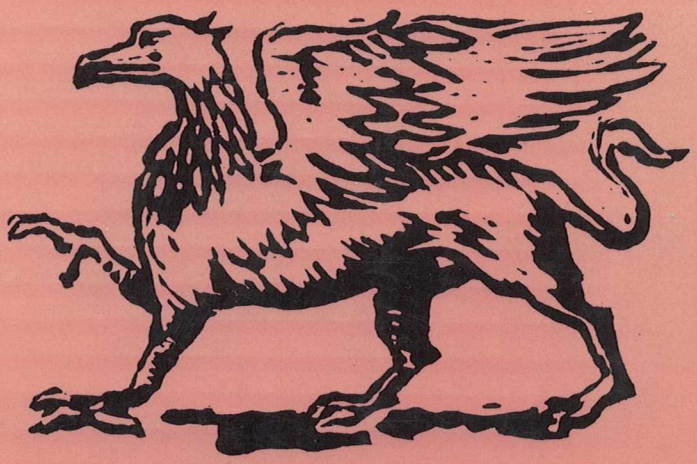
# Dr. Dobb's Journal

For Users of Small Computer Systems

A Reference Journal for Users of Home Computers

## Volume Six



The People's Computer Company Series

# Dr. Dobb's Journal

For Users of Small Computer Systems

## Volume Six

# DR. DOBB'S JOURNAL

For Users of Small Computer Systems

## A Reference Journal for Users of Home Computers

# Volume Six



The People's Computer Company Series



HAYDEN BOOK COMPANY, INC.
Hasbrouck Heights, New Jersey

# Acknowledgments

Many thanks to all authors and readers who have contributed articles, ideas and illumination to the pages of *Dr. Dobb's Journal*. Many of the items found in these pages would have been inaccessible to the microcomputer hobbyist, or available only at great expense, has their creators not felt that immersion in the critical cauldron of the computing community were more important than financial compensation.

Much gratitude also goes to the Friends of PCC. They are a large and loose-knit band united by such ephemeral and fleeting moments as potluck dinners, spontaneous music in the parks, coffeehouse synergisms, and occasional moments of genius. These friends keep us sane, and make microcomputing human.

Editor: MARLIN OUVERSON
Book production: JANE A. MCKEAN
Production assistant: ALANA HUNTER
Index: POLLY KOCH, SUE ELDREDGE
Printed and bound by: MALLOY LITHOGRAPHING, INC.

Cover: Linocut of griffin by Wendy Palmer. The mythological griffin, half lion and half eagle, has long stood as an enemy of ignorance.

*Copyright © 1983 by People's Computer Company.* All rights reserved. No part of this book may be reprinted, or reproduced, or utilized in any form or by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying and recording, or in any information storage and retrieval system, without permission in writing from the Publisher.

*Printed in the United States of America*

| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | PRINTING |
|---|---|---|---|---|---|---|---|---|
| 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | YEAR |

# Editor's Preface

## Editor's Preface

Since 1976, *Dr. Dobb's Journal* had been serving the microcomputing world – a small, almost cloistered group of hackers and left-wing professionals who believed in the concept of personal computing. But the movement gained momentum. A nose for profits lured big companies into the arena. Even Hewlett-Packard, after throwing away the Apple, joined the fray. Publishers like McGraw-Hill and, later, Ziff-Davis took up the scent.

It became obvious that more and more people were developing their microcomputer expertise. More people were graduating to the next stage of understanding, entering systems work instead of simply using or writing applications. In other words, the potential audience for *DDJ* was growing rapidly.

Beginning at about this time, *Dr. Dobb's Journal* began accepting limited amounts of advertising. The editor was charged with the responsibility of not allowing any such source of income bias the textual content of the magazine. Readers of this bound volume will note that certain page numbers are missing; this is because the advertisements were not reproduced here. All text is, however, included. Despite the inclusion of some advertising, and a new cover designed to work well for newsstand sales, the articles continued in the time-honored, reader-defended tradition. Utilities, algorithms, compiler design, operating systems, and lots of useful public-domain software remain the focus of the magazine to this day. By remaining free of corporate publishing interests, we have retained the freedom to publish what is important, not what will simply create the mass market sale.

It is our conviction that the same level of computing power and ease should be available to individuals and small organizations as rests at the fingertips of governments and corporations. What is at the disposal of politicians and financiers should also be used by and for just plain people. In the beginning days of personal computing, there was a strong awareness of the fact that whoever has control of these machines has control of information and computing utilities that directly influence the daily lives of members of entire civilizations. Micro enthusiasts were banded together to improve personal computing power by sharing public-domain software, home-brewing better machines, and working with the physically handicapped.

It will be interesting to note whether the descent of microcomputers into the mainstream of industrial nations will dilute or eradicate that revolutionary spirit. Will the programmer who now can readily sell games or utilities to a receptive public still take the time to tinker with software and hardware controls for a paraplegic's "smart" wheelchair, or fight to integrate the communication protocols for the deaf into the standard which has evolved for the non-hearing-impaired?

At the end of this decade, we must hope to find that we have heeded the call to conscience presented to us by the microcomputing revolution. Perhaps we should remember that this phenomenon truly did begin as a revolution. Whether or not the powerful tools we have created will help to transform this world into an even better one will depend on the people involved and on their priorities. It is still unclear whether our ideals or our bank accounts will prevail.

Either way, *Dr. Dobb's Journal* is providing the tools for advanced computer users. Use them in good health.

**Marlin Ouverson**
**Editor**

# About People's Computer Company

People's Computer Company, the publisher of *Dr. Dobb's Journal*, was founded in 1972 as a non-profit source of education and information about computers. It grew from the same roots as the *Whole Earth Catalog*. Both companies were offshoots of the Portola Institute of Menlo Park, California, an organization investigating and promoting the idea of alternative technology. For the founders of PCC, this meant learning to use computers as a liberating force in society.

To make these ideas manifest, it has been necessary to think far ahead, to be creative and innovative. PCC published the first personal computing periodical in 1972, a time when the technical community at large found the thought of "personal" computers preposterous.

PCC has initiated a number of successful service-oriented projects in order to empower people to take a more active part in a society in which computers are a significant factor. It has published a newspaper, three magazines, and several books on home/personal computing. It has also operated a neighborhood computer center, a mail-order bookstore, a computer van, and has sponsored telecommunications research (PCNET).

The main emphasis of the company has been small, personal computers – the kind anyone can learn to use and many families can afford. ComputerTown, USA! is a major project begun with financial assistance from the National Science Foundation. It provides a model of how to make everyone in a small- to medium-size community "computer literate." It shows how to use local resources – public libraries and museums, schools, community centers, local businesses, and donations of hardware and volunteer time – to give children and adults direct, hands-on experience with computers. ComputerTown shares its experience and resources in a newsletter, in the ComputerTown Books series, in the ComputerTown Guidebooks, and through classes and membership services. Affiliates exist in the United Kingdom, Europe, Australia, and across the United States.

*Dr. Dobb's Journal* was PCC's second publication. The first issue was published in 1976 in order to provide short-term distribution of the newly written Tiny BASIC language. Reader response was so strong that it went immediately into monthly production. It has led the way since then by focusing on important advances, printing public domain software, and fostering vital reader interaction on technical matters. Most readers describe their experience with computers as intermediate to extensive. About ten percent of them live outside the United States. Volumes One through Six include the first six years of *DDJ's* publication, and are available in matching, bound volumes from Hayden Book Company.

The PCC staff members have a wide variety of personal and professional interests and experiences. Their work stems from a conviction that the personal computing revolution is of tremendous significance, and that familiarity with computers fosters confidence and greater control over one's life.

Bob Albrecht, one of the company's founders, explained the inspiration for the company in the first issue of the *PCC Newspaper*:

"Computers are mostly used against people instead of for people, used to control people instead of free them. It's time to change all that. We need a People's Computer Company."

**People's Computer Company**
**P.O. Box E**
**Menlo Park, CA 94025**

# Contents

## VOLUME 6, NUMBER 51, 1981


2 Editorial
3 Letters
5 *Dobb's Ex Machina*
6 What FORTH Is? H. T. GORDON
14 Of Interest
15 Z-80 File Comparator DONALD E. FAULKNER
16 Z-80 Video Drivers ANTHONY SKJELLUM
36 Software Review
*"8080/Z-80 Editor, Assembler, Disassembler"* J. N. DEMAS
41 Book Reviews
42 Intext — An Input Routine for Text Editors GREG SMITH
45 Turing Machine Simulation J. L. LAWRENCE


## VOLUME 6, NUMBER 52, 1981


50 Editorial
51 Letters
54 Parfor — A Theoretical, Parametrized, FORTH-like HLL H. T. GORDON
60 Sinclair's ZX80 JOHN B. PALMER
62 POKE and PEEK to Your Heart's Content T. J. VAN IDERSTINE
63 PAN — One Activity of the PCNET Project DAVE CAULKINS and DAVID C. HARRIS
64 DECONCAT — A De-Bugging Tool for Concatenated Programs GREG SMITH
66 Small-C — An Implementor's Notes and a Bug Corrected P. L. WOODS
69 Z-80 Memory Test RAY DUNCAN
80 Software Reviews
*"DB.BAS — General File Maintenance"* MICHAEL FAVITTA
*"ZDM for CP/M Users"* GENE HEAD
82 *Dobb's Ex Machina*
84 Of Interest
86 Book Reviews
88 Radio Shack Pocket Computer DICK GREENLAW


## VOLUME 6, NUMBER 53, 1981


90 Editorial
91 Letters

95 Computer-Based High Speed Reading DAVE CAULKINS
96 Analysis of the Use of the 6502's Opcodes HOWARD W. WHITLOCK, JR.
99 Book Reviews
100 ERGO: A "Thinking Program" SHELDON SOFTKY and SYLVAN RUBIN
102 Using Word-Master with the Compal-80 ANTHONY SKJELLUM
103 Byte Manipulation and Byte Pointers EDWIN E. FREED
104 Small-C: Current Status P. L. WOODS
122 Software Reviews
*"CLOAD Magazine, Inc."* JOHN HATFIELD
*"MON 3"* STEVEN WEXLER
*"Disk Keyplus"* MARCIA KALISH
124 DOSMOVER for the North Star THOMAS W. CAGE
126 Video Underline Modification DANNIE E. DAVIS
127 TRS-80 Voice Synthesizer on Other Microcomputers MICHAEL HIGHTOWER and PETER B. MAGGS
131 Of Interest

## VOLUME 6, NUMBER 54, 1981

136 Editorial
137 Letters
140 Z-80 Floppy Disk Test RAY DUNCAN
146 Polynomial Evaluation in BASIC R. A. HOFFMAN
148 Algorithms in FORTH S. B. BASSETT
149 A SLIC Label Program ROBERT HALLEM
150 A Cassette I/O Routine for BASIC Arrays ARNE ROHDE
174 Of Interest
177 Important Features of the PCNET Protocol DAVID C. HARRIS
178 Software Review
*"MINCE: Not Just Another Editor"* BARRY A. DOBYNS

## VOLUME 6, NUMBER 55, 1981

184 Editorial
185 Letters
187 Using (and Misusing) the Z-80 Microprocessor RAY DUNCAN
189 A Smart 2716 EPROM Programmer ERIC M. MILLER
191 Of Interest
194 What to Do With Those Extra Function Keys JOHN B. PALMER
197 CAD for Time Delay Circuits HAL KNIPPENBERG
198 Book Reviews
200 Dr. Dobb's Clinic DAVE CORTESI
202 Santa Barbara Tiny BASIC for 6809's JAMES A. HINDS
220 Software Review
*"SCRIBE IX"* GENE HEAD

## VOLUME 6, NUMBER 56, 1981


226 Editorial
227 Letters
230 Cross Assembler for 8080 or Z-80 JONATHAN MILLS
235 COMAL-80 — A New Language? SUSAN SONDERSTRUP and MOGENS PELLE
237 Guest Essay
*"Re-thinking Program Editors"* WILLIAM B. BROGDEN
239 H8 Text Editor to CP/M File Conversion JIM SCOTT
240 A Tiny File System WILLIAM A. GALE
268 Book Reviews
270 Installing a Printer Toggle in North Star DOS 5.2 BERT SMART
271 Dr. Dobb's Clinic DAVE CORTESI
273 Of Interest


## VOLUME 6, NUMBER 57, 1981


276 Editorial
277 Letters
282 Pidgin — A Systems Programming Language WILLIAM A. GALE
287 Time on the Horizon D. J. YATES
291 BASIC Rubik's — A Cube Simulator J. W. PEHOUSHEK
310 Of Interest
312 Dr. Dobb's Clinic DAVE CORTESI
315 Book Reviews


## VOLUME 6, NUMBER 58, 1981


318 Editorial
319 Letters
321 Write Your Own Compiler WILLIAM A. GALE
330 Some Useful C Time Functions BILL BOLTON
354 Dr. Dobb's Clinic DAVE CORTESI
357 Of Interest


## VOLUME 6, NUMBER 59, 1981


362 Editorial
363 The FORTH Philosophy KIM HARRIS
369 A FORTH Assembler for the 6502 WILLIAM F. RAGSDALE
381 Sallying FORTH to Battle R. G. LOELIGER
383 Screen-Oriented Editor in FORTH HENRY LAXEN
398 Elements of a FORTH Data Base Design GLEN B. HAYDON
403 FORTH Decompiler RAY DUNCAN
408 Of Interest
411 The FORTH Alternative MITCHELL E. TIMIN
414 Dr. Dobb's Clinic DAVE CORTESI

## VOLUME 6, NUMBER 60, 1981


418 Editorial
419 Letters
422 Dr. Dobb's Clinic DAVE CORTESI
423 TEK4010 — A Tektronix 4010 Simulator ANTHONY SKJELLUM
427 BASIC-FORTH — A Poor Man's FORTH Computer C. H. TING
429 The Conference Tree — Computer Conferencing JOHN S. JAMES
460 Of Interest


## VOLUME 6, NUMBER 61, 1981


464 Editorial
465 Letters
466 Of Interest
468 ADVEN-80 — An Advanced Adventure Development System PETER D. SCARGILL
487 Terminal Programs for North Star RICHARD BLESSING
492 Small VM — Portable Software Development Environment J. E. HENDRIX
504 Small-C Grows Up RICHARD ROTH
510 Essay on Hacking GANDALF
515 Dr. Dobb's Clinic DAVE CORTESI


## VOLUME 6, NUMBER 62, 1981


518 Editorial
519 Letters
520 Dr. Dobb's Clinic DAVE CORTESI
522 Of Interest
526 Actors, Property Lists, and LISP RONALD L. NICOL
533 Marching the 8080 — A Simple Memory Chip Tester JAMES LAURINO
535 For CP/M BIOS — An Improved MX-80 Printer Driver GARY P. NOVOSIELSKI
545 Connect A Parallel Printer to North Star (More on the MX-80) ALAN BOMBERGER
547 Small-C Expression Analyzer — Patching the Address Arithmetic J. E. HENDRIX
551 Auto. Loading of Assembly Language Subroutines JOHN B. PALMER

# Dr. Dobb's Journal

For Users of Small Computer Systems

## Volume Six

$2.50
In Canada $3.00

# Dr. Dobb's Journal of COMPUTER Calisthenics & Orthodontia

*Running Light Without Overbyte*

Number 51 | January 1981 | Volume 6, Issue 1

## A REFERENCE JOURNAL FOR USERS OF SMALL COMPUTERS

**What FORTH Is?**
*H. T. Gordon*

**Z-80 File Comparator**
*Donald E. Faulkner*

**Z-80 Video Drivers**
*Anthony Skjellum*

**Intext: An Input Routine for Text Editors**
*Greg Smith*

**Turing Machine Simulation**
*J. L. Lawrence*

**and much, much more . . .**

P. O. Box E
Menlo Park
CA 94025

DR. DOBB'S JOURNAL of
COMPUTER Calisthenics & Orthodontia

Volume 6, Issue 1
Number 51
January 1981

# Contents

## Articles

**What FORTH Is?** . . . . . 8
*by H. T. Gordon*
**Z-80 File Comparator** . . . . . 19
*by Donald E. Faulkner*
**Z-80 Video Drivers** . . . . . 20
*by Anthony Skjellum*
**Intext:** *An Input Routine for Text Editors* . . . . . 46
*by Greg Smith*
**Turing Machine Simulation** . . . . . 49
*by J. L. Lawrence*

## Departments

Letters . . . . . 4
*Dobb's Ex Machina* . . . . . 7
Of Interest . . . . . 17
Software Review . . . . . 40
Club News . . . . . 44
Book Reviews . . . . . 45

© 1981 by People's Computer Company. All rights reserved.

*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, California 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025.

Subscription Rates: $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. dollars, drawn on a U.S. bank.

**Writer's Guidelines:** All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

**Publisher**
Ann Merchberger
**Editor**
Marlin Ouverson
**Contributing Editors**
Dennis Allison
Dave Caulkins
Dave Cortesi
**Art Director**
Clifford West
**Artists**
Barbara Ruzgerian
Greg Gordon
Jeannie Barroga
**Typesetter**
Renny Wiggins
**Marketing Director**
Gavin W. Cullen
**Advertising Director**
Michael Madaj
**Circulation Manager**
Peter Clark
**Circulation Assistant**
Leah Dansby

# Editorial

Change has been called the essence of manifestation. *DDJ* is growing continually, and is under the guidance of a new editor. The position carries with it a keen responsibility not only to the readers, but to the ideals of the publisher. These are of such broad impact and general importance that the commitment to them can only be met by calling together many voices to speak within the pages of this publication.

Among this issue's letters to the editor is one which expresses dissatisfaction with the direction followed by *DDJ*. The new subscriber is shocked that the ". . . founding fathers have abandoned ship . . . and in their place a girl journalism student . . . [who cannot] pretend to any technical competence . . . ."

With the most recent change of editors, the issues of feminism and journalism became merely rhetorical. It is, however, vital for each of us to realize that the technical excellence of this journal derives from many sources. The founding fathers have not abandoned ship in any sense of the expression. Each of them contributes directly and indirectly with articles, advice, overall guidance and specific, critical comments. Additionally, the staff relies upon a growing number of technical advisors. It would not be appropriate for any of these fine people to quit their own endeavors to become journalists, for it would mean fewer experts at work on new developments in the field. As H.T. Gordon states in this issue, "Most experts are doers, not teachers."

Most important of all to the realization of *DDJ*'s ideals are its readers. The entire philosophy of the publisher hinges upon the demystification and proliferation of computer expertise. Our contributors are exemplars of this outlook. While each of us benefits from the volatile exchange of information, there is also within each of us at least a half-kindled spark of altruism. It is that which moves us to give, to share a discovery or to provide a new way of looking at an aspect of personal microprocessing.

We are engaged in a concerted effort, in which each of us has an important role. Your letters, thoughts and articles are always welcome.



Marlin Ouverson
Editor

# Letters

Dr. Dobb's Journal *welcomes letters to the editor as a forum for ideas, innovations, irascibility and even idiosyncrasies. Some letters may be edited for clarity and brevity. The Doctor likes hearing from you – keep on writing!*

## Note on Multiple File Transfer

Dear *DDJ*:

I am a new CP/M user, presently with a single drive system (Thinker Toys double density), so the MFT – Multiple File Transfer Utility in the October issue of *DDJ* – came along at just the right time for me. However, I found two minor problems with the way it was written.

First, when the copy function is completed it re-boots immediately with whatever disk is currently in the drive. If it doesn't have a copy of CP/M on it, or has a different version of CP/M (a different memory location, for example) it goes wild. This is easily corrected by adding one more message:

```
MSGF  DB 'Mount System disk,
         press any key',0
```

and replacing the JMP 0 instruction at label MFTG with the following routine.

```
MFTG  LXI H,MSGF
      CALL WASC     ;print message
      CALL RACC     ;wait for key-
                     board input
      JMP 0         ;now re-boot
```

This gives you the opportunity to swap disks again before it re-boots.

Second, I had some problems when trying to copy from single density disks. Adding a CALL RESET instruction between the CALL WEOLC and LDA IBFLG instructions in the MFT1 routine (top of page 26) corrects this.

I hope these modifications will help others; this program works great. (And it saved me from trying to write the same thing from scratch!)

Larry Hudson
2232 Alameda Ave.
Ventura, CA 93003

## One-Byte Opcodes

Dear *Dr. Dobb's*:

H.T. Gordon comes very close to answering how to create a large number of one-byte opcodes in *DDJ* #49. "Opcode Design: 9 or 16 bits" in several places was on the brink of suggesting a microprocessor with a "language register." The contents of this register would specify the machine language currently being executed.

Some of the CPU's languages could be optimized for string manipulation, floating point math, input/output, sorting, file handling, process control, Pascal, BASIC, etc. Other languages might emulate a 6502, 8080, 6800, etc. The assembly language of this multi-lingual microprocessor would include commands to read and write a language register. The actual opcodes generated by these commands would of course depend on which language was in use.

Steps in a similar direction have been taken by circuit board designers. Godbout Electronics offers a Dual Processor Board with an 8085 and an 8088. Microsoft has a Z80 card to plug in with the Apple's 6502. Ohio Scientific has offered several co-resident processors, including the Z80, 6502, 6100, and 6800. But these merely offer the choice of one of several compromises – each chip is designed to do everything. A single microprocessor implementing several languages, each of which was designed to do one thing well, would provide the "quantum leap in performance" H.T. Gordon mentions.

Sincerely,
Charles Floto, Editor
BUSS
325 Pennsylvania Avenue, S.E.
Washington, DC 20003

*H.T. Gordon responds:*

Dear Mr. Floto,

Thanks for sending me a copy of your *DDJ* letter *re* my 9- vs. 16-bit article. I wrote that in a few, hypermanic hours, without the usual long reflection and rewriting such things ought to get. Only later did I wonder whether a single status bit gives designers enough of a "handle" on the circuitry to provide two distinct and meaningful interpretations of a set of 8-bit opcodes. That may be why 8-bit leading bytes are used to "completely redefine" an opcode. Your concept goes very far beyond what I had in mind at the time (enlarging a set with minimal loss of speed and code efficiency), and would not be invalidated even if several status bits were needed for a total reinterpretation. It is possible that, even though I didn't see it, some CPU designers (especially the 2650!) did, but decided the circuit complexity would be too formidable in the then state-of-the-art. Who knows, the 6809 designers may have mused about it but been told not to get too wild! *That* possibility certainly crossed my mind!

Your idea is powerful and may now even be feasible, though doing it might drive designers up the wall. Still, it might be worth sacrificing a few geniuses to the gods (though I'd rather sacrifice a few hundred executives) to get a chip that could acquire magical powers in a few microseconds. It would be a quantum leap, though the distinction between "co-resident processors" and opcode-extension of one super-processor is a subtle one. Some operations, like read/write of memory, might well be identical for all the "processor sets." My impression (that will be developed in forthcoming articles on FORTH-like HLLs) is that designers are now thinking hard about optimizing microprocessor instruction sets to "run" the intermediate-code of various HLLs. Perhaps your "9+ opcode" concept would be useful there.

Sincerely,
H. T. Gordon
College of Natural Resources
University of California
Berkeley, CA 94720

## A Mote Point . . .

Dear *DDJ*:

As a tech writer for a PR agency, with primarily high-technology clients, I read most of the computer and electronics magazines, and I come across occasional gems of technical prose, which I'd like to share with you and your readers.

Herewith please find my submission to *Dr. Dobb's Compendium of Self-Referential Phraseology*:

". . . clear, concise, industrial grade speech."

It is from a Texas Instruments TM990/36 ad.

Referentially,
S. B. Bassett
c/o 151 University Avenue
Palo Alto, CA 94301

## Captured by the Capitalists?

Dear Editor,

In the October issue you requested suggestions and comments. I have a few for you. I am a new subscriber (starting June this year) but my readership dates back considerably farther. Your magazine came highly recommended from several articles and books and early this year when the Hayden Book Company offered

# Letters

a 15 day trial on the first three volumes I took the chance. I expected to be disappointed. The very concept of the People's Computer Company seems doubtful to me, since I am convinced that the capitalist system is the only currently viable one. So I expected that shared software would be cast-offs, half-baked schemes etc. Instead there were indeed useful listings, interpreter-writing aids, and other sytems software level tips. I find those first three volumes a valuable reference. So I subscribed in June. Imagine my shock. The founding fathers have abandoned the ship (contributing editors?) and in their place a girl journalism student whose first suggestion was to have more business oriented practical articles (in a systems software journal?). Neither Suzanne nor yourself can pretend to any technical competence in microsystems software. And that seems to me to be a mistake. Sure, a generalist can edit any text, but in order to publish a journal which was intended to be an exchange point for systems software it seems to me that you need to understand the technical details in depth.

What I am trying to say is that, for me at least, your journal is slowly becoming useless. There are plenty of magazines for non-technical beginners in higher-level languages already. I am looking for details on writing systems utilities such as spoolers, designing compilers, text-editors and so on. Maybe a tutorial on CP/M concepts is useful, but PASCAL? My suggestion? You've been captured by the capitalists. Give up the ads and attempted slick format. Go back to being a systems software journal. Give us more lowdowns on others' problems with new systems. Give up being an expensive imitation of *BYTE* and *Kilobaud*. Go and reread those first issues and get back to the basics. Get someone who's competent in the field to (co-)edit.

I'd also like to respond to a letter in that October issue, in which Prentice-Hall was criticized for their 15-day trial program. Since I don't live in a big city (or California), I have had to use these free trials to review the (usually oversold) new publications. I have had excellent success with both Prentice-Hall and Hayden Publishing Companies. It has never required more than one letter to correct any misunderstandings. There is a cost, however. If you keep the book you end up paying the usual postage and handling fees and if you send it back you pay for the postage (usually around a dollar). Of course, there is no free lunch. Since the book is mailed fourth class and the bill first, it should come as no surprise that you might get the bill first. I have never had a publisher question the length of the trial. They know the mails are slow.

I hope you can use this.

Very truly yours,
Siebert Ickler
3709 Bentwood
Corpus Christi, TX 78415

*Editor's note: Please refer to this month's editorial on page 3. Thank you for airing your views . . . but really,* DDJ *an "expensive imitation?"* DDJ *was the first, and still remains on the cutting edge of innovation, thanks to its contributors, its technical advisors and the founding fathers.*

## Pascal Postscript

Dear *DDJ*:

I recently perused *DDJ* #48, in which Mr. John Culleton discusses (in the Letters column) the "upside down" nature of Pascal. While I personally agree with this characterization, my recent experience modifying a Pascal compiler (to operate on the M68000) has given me a clear appreciation of the problems which lead to this order. I offer the following not as an apology, but as an explanation.

In passing, I would like to clarify one of the diagrams in Mr. Culleton's letter. He refers to a program with main routine 000, calling 100, 200, and 300, which in turn call 110, 120, etc. His diagram of the order "required" by Pascal is correct, but not actually required; it appears on the left, and a truly "required" order appears on the right.

| | | | | | |
|---|---|---|---|---|---|
| 900 | | | 900 | | (utilities) |
| 910 | | | 910 | | |
| 920 | | | 920 | | |
| | | 110 | | | 110 |
| | | 120 | | | 120 |
| | | 130 | | | 130 |
| | | 210 | | 100 | |
| | | 220 | | | 210 |
| | | 310 | | | 220 |
| | | 320 | | 200 | |
| | | 330 | | | 310 |
| | | 340 | | | 320 |
| | | 410 | | | 330 |
| | 100 | | | | 340 |
| | 200 | | | 300 | |
| | 300 | | | | 410 |
| | 400 | | | 400 | |
| 000 | | | 000 | | |

As can be seen, the lower level routines need not all be defined first; in fact, if they are called only from their associated higher level routine, they should not be. Therein lies the problem.

The definition of group (200, 210, 220) seems to indicate that the subroutines are defined before the higher routine. This is not quite the case. The order is more like this:

200's definition;
200's variables;
210's definition;
210's variables;
210's text;
220's definition;
220's variables;
220's text;
200's text

I support the view that this puts the "Mainline" text, which is what most people read when they peruse a program, in a strange order. However, the definition of the routine and its variables appears before the subroutines, which is required by the Pascal-defined "scope of identifiers."

Routines 210 and 220 can refer to 200's variables, and in fact if they do not specifically define their own variables with the same names, they can refer to 000's as well. 200 cannot refer to 210's variables, since they are at a different level (they have a definition in the way). More importantly, since Pascal is a stack oriented system, 210's "variables" are really just offsets into a stack frame which is allocated when 210 is called and de-allocated when 210 finishes. They therefore *do not exist* when 200 is in control. Since 200's frame is underneath 210's, the "upward" reference is possible. (Similarly, 000's frame is further down yet and is thus always accessible.)

Languages like COBOL and FORTRAN differ from Pascal in that their variables are allocated at the beginning of time and stay in the same place forever. FORTRAN does not allow any referencing of variables not explicitly defined in the current routine or function; COBOL does not have subroutines (for the most part; PERFORM is not a proper subroutine call). It may be noted that PL/1, a recursive dynamic-allocation language, has the same definition structure for embedded routines.

I have used FORTRAN compilers which required the routines to be entered in lowest-reference (i.e., upside-down) order; I have also used an assembler

*(continued on page 16)*

# DOBB'S Ex Machina

**Occasionally, a reader of *Dr. Dobb's Journal* will experience poor product performance or what he considers to be a company's failure to provide the quality of service they advertise. If his own honest attempts to resolve the situation fail, he may then take recourse to the impartial (if not heavenly) mediation of *Dobb's Ex Machina*.**

**When writing a complaint to us, please send backup data (e.g., carbons or photocopies of letters you have sent to the company in question, replies received, copies of registered receipts). It is not possible to cover all complaints received, but we will publish a representative sample. *DDJ* has no particular ax to grind with any seller or manufacturer, but we will pay close attention to those companies about which we receive numerous and consistent complaints.**

**Dear Editor,**

You've had quite a few letters about mail order problems, but let me tell you what can happen in playing safe and ordering from a local store.

Toward the end of last year I planned to upgrade a system used for self-funded research into methods of improving children's medical care, basically substituting a CRT terminal for the old faithful VDM-1.

I decided to play it safe and ordered a well-known name from a local store, rather than ordering one of the newer brands from a mail order discounter. It seemed likely that paying a good bit more initially would pay off eventually in better service and support.

However I made two mistakes. I ordered a Hazeltine 1520 from a local Computerland.

The terminal didn't work properly when unpacked. Computerland not only couldn't find the problem, they weren't sure whether or not one existed. The Hazeltine service agency, TRW, did find a problem and fixed it, but when the unit was brought home a different problem was evident, so back again. (The TRW technician said he was not responsible for general fitness of the terminal, but only for the present problem.)

In all, the Hazeltine failed six times within 90 days. Five times it was taken to TRW on an 80 mile round trip and returned for another 80 miles. I kept thinking that this time would see it fixed for good, but at the sixth time I couldn't face another 160 miles of traveling, particularly when the terminal had never worked long enough to do anything useful. I wrote a restrained letter to Hazeltine asking for a refund.

Hazeltine's response has consisted of pointing out that they guarantee only to fix faults, not that they will stay fixed, and accusing my system of being at fault. (This system worked well with a DECWRITER for years and the televideo terminal, in use now, has been flawless for many months.)

Having a bit more of a conscience myself I can't recover part of the nearly $2000 by unloading it on some other sucker. I can't afford to write this kind of money off to experience. It shot a big hole in my research budget, but what can one individual do against a large corporation? Try to warn others I guess.

Yours,
Jim Callas
650 N. San Pedro Rd.
San Rafael, CA 94903
8/11/80

**Dear Editor,**

Thank you for allowing Hazeltine Corporation the opportunity to respond to Dr. Callas' letter. Given that Dr. Callas represents a unique group of consumers it is important that your readers understand the chronology of events and perhaps gain additional insight into the potential pitfalls that can plague an unsuspecting consumer as they relate to purchasing computer peripheral products.

To begin with, let's review the events as they took place:

1) Dr. Callas purchased a Hazeltine 1520 from our customer, Computerland, in November of 1979. We were not advised of a problem until we received a letter on March 3, 1980.

2) Personal contact with Dr. Callas was made on March 6, 1980, at that time we requested additional information, such as the serial number for investigation of his problem.

3) On March 27, Dr. Callas' attorney called Hazeltine requesting a refund. All communications since then have been through his attorney. To date we have yet to discuss with Dr. Callas the actual conditions the unit was expected to operate under.

4) Through his attorney we have learned that he was trying to run a teletype off the Hazeltine 1520 using a "proprietary" circuit. The design of that circuit may not be compatible to our unit causing the many failures he has experienced. There is no way Hazeltine or Computerland can diagnose and find the problem in such circumstances.

5) Hazeltine's 1500 Series carries an "On-site" warranty for commercial customers. Dr. Callas' unit is in his home. T.R.W. cannot service units in private dwellings. If Computerland was closer to Dr. Callas' home than the T.R.W. repair center, arrangements might have been made to do warranty service at Computerland.

6) T.R.W. responded correctly to Dr. Callas. They are not responsible for general fitness of our terminal. That is, they are not responsible for whether a particular Hazeltine terminal will operate in an environment which is unique to a particular customer. Hazeltine cannot warrant that our unit will perform with any particular system, because we do not have control over the systems to which a customer connects our terminal.

As one might assume, Hazeltine has incurred substantial warranty costs in an unsuccessful effort to make the H1520 work on Dr. Callas' system. But that is not the issue. The point to be made here is that there is an abundance of terminals available "off the shelf" from distributors or retail computer stores. Compatibility can never be assumed. Incompatibility and poor product performance are truly two distinct issues; one is a function of a product's technical specifications, the latter is simply a quality problem. T.R.W. stands ready to make repairs should the customer experience a problem due to a failure of quality control. There is a specific burden of risk placed on the buyer when he purchases from distributors or retail computer stores that the product will be compatible with the buyer's system. The store may assume that the buyer knows what he needs and what technical specifications are correct for his system. As a point of information, Hazeltine employs a direct salesforce, an Engineering Department, a Customer Service Department and a Technical Support Department, all of which are at our customer's disposal. Had Dr. Callas first confirmed the compatibility of his system with the H1520, we suspect his decision to purchase the H1520 would have been affected.

It is our opinion that Dr. Callas received good service and the best possible support under these circumstances. We

*(continued on page 48)*

# What FORTH Is?

## by H.T. Gordon

My brief critique of FORTH (*DDJ* #48, p. 19) elicited several positive responses from the FIG (Forth Interest Group). The one that sparked this note was a call from Daniel Conde, a UCB student, telling me that some FORTH literature had been given to our Computer Science reference library by Jim Flournoy. On browsing through this, I was pleasantly surprised to find a copy of an unpublished article: *The FORTH language . . . a detailed study of U.C.S.D.'s version of FORTH*, by Bruce Airy, who is or was a student at the U.C. San Diego campus. He seems to have implemented FORTH on an antique NOVA computer (he has many wry comments on its weaknesses). This odd experience, plus an awareness of how to communicate with novices that is rare in computer science, makes his writing the clearest introduction to FORTH I have so far read. Because its NOVA orientation makes it unsuitable for *DDJ*, I decided to write a shorter and more generalized presentation (the "?" in my title acknowledges its non-expert character).

It may be argued that the recent expert articles on FORTH published in *BYTE* (Aug. 1980) are pre-emptive. Excellent though they are, my own impression is that they probe less deeply into the working of FORTH than Airy's version. Expertise in the cryptics of FORTH code seems to inspire the same aversion to human-oriented documentation that is so common among assembly-language programmers (you're telling the machine what to do, so who cares about dimwitted *people*?). At the human level, the way to dispel subconscious fear of something new and strange is to lay stress on its similarity to what is well-known and understood. That's the secret of one of the rare human achievements: good teaching. Most experts are doers, not teachers.

**An impressionist overview of FORTH.** As I now see it, FORTH is what a simple command-language will evolve into if (1) its creator is unable to resist internal or external compulsion to enhance it endlessly, and (2) it proves so useful that he is driven to adapt it to more than one kind of machine. The first I know from personal experience to be an irresistible temptation. The urge to invent new commands in my EDITHA program (*DDJ* #25) was halted by lack of memory in the KIM-1. In my DISPAG program (*DDJ* #48) the brake was mental fatigue. In both cases, the driving force was the need to enhance performance far beyond the limits of the ROM monitor commands. In his *BYTE* article, Charles Moore (the inventor of FORTH) says that he developed this "unusual" HLL in order to write complex programs at a higher rate than that possible with other HLLs.

A "command" is generated by pressing a key (or a sequence of keys), to send a bit-pattern to an interpreter program. The interpreter "decodes" it to the address of a block of machine-language code, that is jumped to, executed, and terminated by a return to interpreter control. The simplest possible command-language would consist only of addresses, but no such language exists because our minds are not that numerical! Human evolu-

*H. T. Gordon, College of Natural Resources, University of California, Berkeley, CA 94720.*

tion has programmed us to communicate with our own kind, first with sounds and gestures, then with written visual equivalents, then (when machines entered the picture) via ASCII equivalents. So our commands to a machine must originate in our minds as sounds or pictures, with mnemonic values that guide our fingers to the keys of the correct command-code.

Most commands require prior entry of parameters that will be used by the command-execution code. This entry is often done under control of *implied commands* within the interpreter. Bruce Airy perceived that commands are analogous to the built-in operation codes of a CPU, and that parameters are akin to the machine-language "operands." Some commands may have *implied parameters* (existing within the execution-code), and (like complex opcodes such as INC) also imply operation of more primitive (i.e., simpler) commands.

The great creative idea in FORTH is its enhancement of interpreter logic to make possible the construction not only of new "primitives," but of commands that consist of sequences of such primitives, that (like a machine-language program) can be run without interruption. Freeing a command-language from the restriction of single-step execution profoundly influences its development. The *opcode-like* primitives must be designed to *interact in sequence*, since they are the instruction set of a "virtual" machine that is being emulated on a simpler, real machine. The program-like complex commands require the high-level equivalent of a CPU program-counter, and also *branching primitives* to make possible loops and conditional jumps.

**What – and where – are the FORTH primitives?** There is a universal tendency for innovations to develop a special jargon either of neologisms or of pre-existing words whose meaning can be stretched to include something rather different. Since a command-code consists of one or more ASCII characters (not including, but delimited by, spaces) it is, in some sense, a "word." The command-set is therefore like a vocabulary. The list in which the command-codes are stored, with their definitions, has some nebulous kinship to a dictionary. In reality, it is a unitary file, organized as a linked list so that it can be searched through. The basic primitives are in its lower region, and it is extensible at its high end, new user-defined commands being appended in chronological order. The structure of each entry in the list consists of a uniform "head," followed by a "body" (definition) determined by the logic of the "defining word" primitive used to construct it. The body of every primitive command is a program written in the machine-code of the CPU being used by the FORTH system, terminated by a return to whatever control program in the "kernel" happens to be in charge. The kernel machine-code programs, located below the dictionary in memory, control all FORTH operations.

To know what a primitive is, one must know the dynamics of its construction. New-command construction is controlled by the text (outer) interpreter program, working with one of the defining-word primitives. The command-code for building a new primitive is CODE (followed by a space delimiter). Control shifts to the head-constructing routine, which has access to two 16-bit variables in the kernel: HEAD and DP, the addresses of the first and last bytes of the previous entry. The routine expects one or more ASCII characters to be keyed, to be the unique name (command-code) of the new command. The sequence is stored in the dictionary, starting at location DP+2. Keying a space stops the routine. The number ($n$) of characters is stored at DP+1 (the first byte of the new head). To save space, the number of name-characters is truncated to only the first 4. I suspect that some versions compress these into 2 hex bytes, both to save space and to accelerate a dictionary-search. The next 2 bytes are used to store the address of the previous entry, to serve as a link during a dictionary-search. The search routine always starts at the address of the highest (last) entry (saved in HEAD), and if the value of $n$ does not match that of the command-code being sought it uses the link address to find and test the next lower entry. If the $n$ values do match, the character-strings are compared. This means that names such as TEST1 and TEST2 are not unique, since both are encoded as 5TEST.

In Airy's NOVA version, the byte that follows the 4 "name" characters stores the "precedence" of the command in its 2 low-order bits (the other 6 presumably set to zero). So there are 4 possible precedence states (0, 1, 2, or 3), and most commands have a precedence of zero. Within the kernel is a control variable, STATE. A command will be executed only if its precedence is equal to or greater than STATE. In the compiler mode, STATE is set to 2, blocking execution of commands with a lower precedence so that the compiler can fetch their execution-address (required for the construction of "program-like" commands).

Bruce Airy's presentation sometimes falters, forcing me to fall back on personal interpretation. He ignores the case of a single-character name, storable in a single byte. I see no reason not to do this, since the naming-routine has the value of $n$ (in this case, 1). If $n$ is less than 4, it can calculate the storage location for the link address as DP+3+$n$, otherwise as DP+7. A more puzzling element in Airy's version of the head-construction is his storing of the execution-address in the 2 bytes following the storing of the link-address. This address is that of the first byte of the "body" that immediately follows it and contains the logic to execute the command. I see no reason why this logic should not immediately follow the link-address, since its address is calculable by just adding 2 to the algorithm for calculating the location of the link-address.

The construction of the machine-language body of the primitive normally requires that an assembler program have been temporarily loaded (unless it's in a ROM). I've never used an assembler (the 6502 instruction set being being so simple). Were I to adopt FORTH, I would probably prefer a variant of CODE that would just accept a sequence of hex bytes. The program executed by a FORTH primitive ought to be a relatively simple one – complexity being attained by weaving primitives together "in FORTH." That, after all, is why HLLs were invented.

**The role of the "variables" stack.** I see no reason why the program that executes a primitive should not make use of storage locations within itself (or with 6800/6502 systems, zero-page locations) but one of the major FORTH conventions is that the passing of parameters between primitives, or their initialization by user keying-in, always involve pushing them onto or pulling them from a LIFO (last in, first out) stack. Most of the primitives (cf. the Glossary, Aug. 1980 *BYTE*, p. 186) involve stack-manipulations, and the user must at all times be

**"The simplest possible command-language would consist only of addresses, but no such language exists because our minds are not that numerical!"**

aware of what is on the stack. That's the psychological reason why no one primitive either pushes or pulls a large number of variables! The FIG version of FORTH has primitives that play with as many as the 3 most recent stack entries.

If a program plays with a variable in a complex, erratic way, it becomes difficult to ensure that it will appear on TOS (the acronym for top-of-stack) exactly at the right moment. This problem is solved by giving it a name (declaring, i.e., creating, a unique storage location for it), using the defining-word VARIABLE. Like all defining-words, this expects keying-in of a unique ASCII name and creates a new dictionary-entry with the standard head. The body, however, causes execution of machine-code (within the VARIABLE primitive) whose function is to put the address of a 2-byte storage location within the body of the new entry onto the stack. That's what keying in the new name (say XVAL) will cause. To initialize the XVAL storage-location, one must key in a sequence of 3 commands, e.g., 17 XVAL ! The 17 (or any other number) will be interpreted as a command to push it on the stack. XVAL pushes its storage-address on the stack. The ! primitive causes 17 to be stored in the XVAL location, and clears both address and number from the stack. To push the number currently stored back on the stack, one keys in XVAL @. The @ primitive fetches the number located at the address that XVAL has put on the stack, and replaces the address on the stack by that number.

Some readers will wonder (as I did) how any number implies a command to push it on the stack. The FORTH interpreter doesn't "know" that a sequence of ASCII numerical characters is a number. It first searches through the dictionary to see if it's a command-code, but it's not (or ought not to be). When the search is exhausted, control shifts to the NUMBER program in the kernel, which tests to see if it's a valid number in whatever BASE (decimal, hex, or what have you) has been declared. If it is, it's pushed on the stack (otherwise, error-message). Sometimes, the same number will be needed repeatedly in a program, so to ease the burden on human memory it is given a name, by using the defining-word CONSTANT. The sequence 73518 CONSTANT FACTOR will create a new dictionary entry, differing from VARIABLE in that the execution logic will push the number 73518 on the stack whenever the command FACTOR is keyed-in. Unless this number has been carefully chosen because it is a valid address of a RAM storage location (it usually isn't, but could be), primitives like @ and ! will only wreck the operation – certainly they can't alter the numerical value stored in the body of FACTOR. Still, using a constant to push a base-address onto the stack would allow adding a number to it to provide something like indexable access to a table in memory.

I shall not discuss the operation of the long list of primitives, or the deeper question: are these now an optimal macro-instruction set? That needs a lot of thought. My critique of the instruction sets of several microprocessors (*Microcomputing*, #39 and #40, 1980) was based on two years of study, while this note is only a preliminary, superficial exploration of FORTH.

**Construction and execution of program-like commands.** The great creative idea in FORTH is its design of high-level commands that can, without interruption, run a sequence of (ultimately) primitive commands. The : defining-word (colon command) controls the construction process. It expects a new ASCII name and creates a dictionary-entry head in the usual way, but the body is utterly different. Not machine-code, but primarily a *sequence of addresses* of commands that exist in the dictionary. This is not what the user is keying-in at all, but a compilation of the source-code that is being keyed-in. This code is human-level (or as close as FORTH ever gets to it!) and consists of a sequence of command-names and numbers, delimited by spaces. During compilation the STATE variable is reset (to 2 in the Airy version) in order to block execution of most of the keyed-in commands, except for a handful of "compiler imperative" primitives that have a precedence of 2 or 3. The first element in the body is the address of the : primitive, that will (when STATE = 0, allowing the high-level command to run when its name is keyed-in) cause the address (inner) interpreter in the kernel to control execution. When the user has keyed-in the first "word," the space delimiter causes the compiler to search the dictionary to see if it's a known command. If it is, and its precedence is less than 2, the compiler adds its execution-address to the body of the new entry. If it's not, but it is a valid number, the compiler adds the address of the LIT primitive to the sequence, and then adds the numerical value to the sequence. When LIT logic is eventually executed, it will push the numerical value that follows it in the sequence onto the stack, and increment the "command-counter" of the address-interpreter so that the numerical value will be skipped. Construction of the high-level command is terminated by keying-in the ; primitive. This semi-colon command is a compiler-imperative. Its execution first adds the ; execution-address to the new body (in the run state, it will act as a terminator of the address-interpreter operation) and returns control to the text-interpreter.

FORTH has a great variety of primitives which, when inserted in its colon-programs, are the equivalent of branch or jump instructions in assembly languages. Some are conditional, so the equivalent of a status register might be expected. To my surprise, this proved to be rudimentary when compared to the status register of microprocessors. Status can only be false (zero) or true (non-zero), and these numerical values must be on TOS when the conditional-branch primitive is executed. Perhaps this reflects the somewhat infantile state of the CPUs and HLLs existing at the time FORTH was invented. To my mind, it does not justify the belief of Charles Moore (in his *BYTE* article) that FORTH is now so close to perfection that it can be standardized.

There are 4 types of repetitive-looping constructs. Two of them are somewhat like FORTRAN DO-loops. Both are initiated by the DO primitive, whose function is to store 3 items of information for eventual use by a matching LOOP or +LOOP later in the sequence. The first 2 are indexing-range numbers, integers that I think of as Endex and Index, that normally immediately precede the DO (e.g., Endex Index DO) and will therefore be pushed onto the stack by LIT primitives. The third is the *return-address* of the DO, available in the command-counter (the equivalent of a CPU program-counter) at the execution-time of the DO. In principle, this could be "remembered" by also pushing it onto the variables stack, since the logic of the repeating-sequence ought to keep this balanced to prevent an excess of pushes or pulls from "blowing up" the stack. However, FORTH creates a specialized return-stack for

**"The program executed by a FORTH primitive ought to be a relatively simple one – complexity being attained by weaving primitives together . . ."**

the parameters used by its control primitives. Execution of the DO pulls Index from the variables stack and pushes it onto the return-stack, then pulls Endex and puts it on the return-stack, and finally pushes the DO return-address onto the return-stack. When the matching LOOP primitive is executed, its logic increments Index and compares it with Endex. If Endex is greater, it resets the command-counter to the DO return-address. Since the counter auto-increments, execution of the sequence immediately following the DO is repeated. If Endex is not greater, the information on the return-stack is no longer needed, so it is cleared (I presume by resetting the return-stack pointer) and the command-counter auto-increments to execute the sequence following LOOP. The +LOOP primitive expects to find an integer (2 or higher) on TOS, pulls it and adds it to Index, then works like LOOP.

An inner LOOP can be nested within an outer one, but its DO requires its own matching LOOP as well as its own index-range parameters. This ensures that the inner LOOP will be completed, and its information cleared from the return-stack, before the outer LOOP is reached.

A special primitive, I, when inserted in the within-loop sequence, causes the current value of Index to be read and pushed on the stack. The return-stack differs from the variables stack in that information on it is much more often read than subjected to the flux of pushes and pulls characteristic of the variables stack. The FIG version of FORTH has primitives that allow the programmer to push, pull, or copy information using the return-stack – but wisely advises caution since unbalancing it spells disaster! This suggests that memory-management can be a problem. The FORTH environment defines 3 storage-areas in memory: the 2 stacks (which must be kept more-or-less balanced), and the so-far-unused area at the high end of the dictionary. I suspect that primitives will in time be created to exploit the last – simpler than creating one more storage area for the programmer to keep in mind.

There are also two types of "indefinite" loops. Escape from these requires that a generator-logic sequence terminate by pushing a true (non-zero) status onto the variables stack. Both are initiated by the BEGIN compiler-imperative, that causes the compiler to push the return-address (current address in the sequence being compiled) onto the variables stack. Since the execution-address of an imperative is not itself compiled (i.e., added to the sequence), the return-address will actually be that of the next program word after BEGIN, the BEGIN causing no increment of the command-counter and so "vanishing" from the program. In the simpler version of this type of loop, there must be a matching END (re-named UNTIL by the FIG) that will decide to repeat or escape from the loop. END is also an imperative. It first compiles the execution-address of the SKIP? primitive, then pulls the return-address from the stack and compiles it. During execution, the SKIP? pulls the "status" left on TOS by the preceding decision-generating logic. If it is zero (false), it resets the command-counter to the return address that follows it in the sequence. If non-zero (true), it causes the command-counter to skip the return-address and move on to the post-END sequence.

The more complex loop is required when the decision-logic must (if false) be followed by error-correction logic, that must always return to the decision logic sequence to see if the correction has worked. If it has (status becomes true), the correction logic must be by-passed in escaping from the loop. The BEGIN works as before, leaving the address of the start of the decision logic on the stack. At the point where the true-or-false status is pushed on the stack, an IF (renamed WHILE by the FIG) is inserted. This imperative compiles a SKIP?, but unlike END it does not pull the BEGIN return-address. Instead, it pushes its own current address on the variables stack, and causes the compiler to leave the address-location following the SKIP? vacant. The correction logic is then written, and terminated by an AGAIN (REPEAT in FIG). This imperative compiles the SKIP unconditional-jump. It uses the address left on the stack by the IF to store the correct SKIP? jump-address (that of the start of the post-AGAIN logic to be run where escape from the loop occurs). It then pulls the BEGIN return-address and compiles it following its SKIP. Like the indexed DO loops, the indefinite loops can be nested, but each BEGIN needs its own END (or WHILE . . . REPEAT) to allow escape from an inner loop before re-entry into an outer one.

The . . . IF . . . ELSE . . . THEN (ENDIF in FIG) construct allows non-loop branching. If the pre-IF decision logic is true, the post-IF sequence is run. The ELSE is optional, required only if the false condition needs special execution logic, but every IF must have its own THEN. As before, the IF compiles the SKIP? address, creates a vacancy for the as-yet-unknown jump-address, and pushes its own address on the stack so that a subsequent imperative will be able to fill it. If that is an ELSE, the correct SKIP? address is that of the post-ELSE, false logic. The ELSE inserts this, using the address left on the stack by the IF. ELSE then compiles a SKIP, but leaves the jump address vacant, pushing its own address on the stack for use by the (inevitable) THEN. All that the THEN has to do is fill the vacant location with the address of the post-THEN logic, leaving no other trace of its passage. If there was no ELSE, its operation will fill the still-vacant jump-address of the SKIP? compiled by the IF. Like the BEGIN, the THEN exists only in the source code, not the program, since the only function is to provide address-information during compilation.

These constructs are extraordinarily simple when compared with the intricacy of branching possible with assembly langauge. Their rigidity is the reason why FORTHians can point with pride to the structured programming quality of FORTH, without saying what is being traded-off for it. But the user-extensibility would allow a programmer to do some interesting destructuring!

**An elementary – but complexly motivated – example.** The generation of the Fibonacci number sequence is a very simple task. The first two numbers are both ones, and each successive number is the sum of the preceding two numbers. It's one of many algorithms useful to reveal how different HLLs work. My reason for writing it in FORTH is historic. While I was talking to a friend who had acquired FORTH and was demonstrating it on his PET, a bystander asked him to generate the first 10 Fibonacci numbers, saying it was easy to do in BASIC. It can't be difficult in any HLL! Yet my friend's coding of it failed to give the correct sequence. I now realize that he had not yet mastered the visualization of the stack parameters that is the *sine qua non* for writing FORTH.

In the listings is my "easy" program – untested since I

**"To my mind, [this] does not justify the belief . . . that FORTH is now so close to perfection that it can be standardized."**

do not have a FORTH system. The first line has it in typical FORTHian, uncommented format. It ought to be somewhat intelligible if my earlier explanantions were instructive. However, the listing then reiterates it in something like an assembler format, with comments, anything within parentheses being ignored by the FORTH interpreter. This is followed by program-execution and output.

My commented version illustrates why FORTHians tend to avoid comment. Note that I have three "multiple-words" that I see as complex-commands with an inherent unity, described in the parenthetic statement to the right. Some words are also provided with location-identifiers at the left. All this is very tentative! Of greater interest are novel conventions to identify stack locations and operations. $S_0$ is the first "free" location, and a push into it is $\rightarrow S_0$. $S_1$ is the location above $S_0$, $S_2$ above $S_1$, etc. The operation $\rightarrow S_0$ causes what was the $S_0$ location to become $S_1$ (and so automatically alters the numbering of the absolute locations that were previously $S_1$, $S_2$, etc.). When one or more filled stack locations are pulled, this is denoted by brackets. The primitive DROP, that just pulls the TOS without using it, would be commented as ( $[S_1]$ ), causing the previous $S_1$ location to become the new $S_0$, with corresponding alteration of the previous $S_2$ to $S_1$, etc. Note the more complex effect of the + primitive in the FIBO program. This kind of stack notation may be more helpful in keeping track of the status than that used in FIG FORTH. For example, their comment of the OVER primitive is ($n_1$ $n_2$ $\rightarrow$ $n_1$ $n_2$ $n_1$) instead of my ($S_2 \rightarrow S_0$), and DUP is (n $\rightarrow$ n n) instead of my ($S_1 \rightarrow S_0$). Note also that I simplify the comment of the immediately-interacting pair DUP. to its overall effect (print $S_1$), to avoid the confusion of ($S_1 \rightarrow S_0$, print $[S_1]$ ).

FORTHians may object to the space-wasting complexity of such a commented format, and anti-FORTHians may still claim that it's not intelligible enough. All I can say is that, since FORTH is a kind of high-level assembly-language, that may be as intelligible as it can ever get.

As to space-wasting, look at the output! Perhaps an expert could use the primitives to write FIBO so as to format more efficiently. Of course, one could create a new type of output command to do anything one likes. But then, the beautiful simplicity begins to erode! Something like this seems to be transmogrifying the (relatively) simple C language little-by-little into a more powerful (but less easy) D.

Something more fundamental comes to mind if we note the similarity between DUP ($S_1 \rightarrow S_0$) and OVER ($S_2 \rightarrow S_0$). There may be (and surely could be) similarities in the machine code of these primitives, that might be generalized to cases like ($S_3 \rightarrow S_0$), ($S_4 \rightarrow S_0$), and perhaps even to ($S_3 \rightarrow S_1$). Analogously, exchanges of the SWAP type might be generalized to include operations such as ($S_3 \leftrightarrow S_1$). Conservative counterarguments to this might be (1) these operations are not useful, (2) if ever needed, they can be emulated by sequences of existing primitives, (3) they would strain the memory of programmers, and (4) their addition would strain the memory of the FORTH dictionary. Similar arguments exist for limiting the complexity of microprocessor instruction sets. Since these get larger in successive generations, the validity of the small-is-beautiful arguments is open to question!

As to (1), some operations are more often used, and some may be more useful than others, but I doubt that any are useless. (2) It's nice to kill several birds with one stone. (3) Remember and use only the subset you prefer, as the users of a complex assembly-language often do. (4) Not necessarily. Complexity can be achieved in three ways: by accretions, which lead to elephantine bulk, by innovative combinations, or by metamorphoses to a higher level of deep structure and understanding. Examples of all these could be drawn from the evolution of many HLLs, as well as of microprocessors (e.g., the 6502 set is smaller and came much later than the 8080, but is not less powerful). However, I shall explore the possibilites in FORTH.

To implement a generalized operation of the type ($S_n \rightarrow S_0$) means replacing two primitives (DUP and OVER) by a single SPUSH, that requires the value of *n* to locate $S_n$. A more general form like ($S_n \rightarrow S_m$) would need 2 parameters, and perhaps more complex logic. The simpler version (implying that m = 0) might expect *n* on TOS ($S_1$) at execution, pulling it to use as an index for its logic. So the price for its power to do many operations is that each would be a sequence of n SPUSH – two commands and a longer, slower-running program. The gains would be a minimizing of dictionary primitives, probably some coding economy, possibly some reduction in strain on programmer memory. That's the "combinations" concept, not too unlike the DO.

**Metamorphic changes in information-transfer by concatenations of characters.** The word *metamorphosis* means a drastic change to a higher level of complexity. It can occur anywhere. Although it is relatively rare, many books have been written about specific instances of it, and even a few on the phenomenon in general. The instance I hope to develop in this section is the class of picture-symbols we call characters. Like all symbols, they are an arbitrary representation of a packet of information that will be transferred from a generator-transmitter system to a receptor-interpreter system. Chinese characters represent a word, itself a symbol. So do many of our characters (like + or =), while others signify sounds (e, z). Where we differ from the Chinese is our use of concatenations of a small character-set to create complex symbols, while they have expanded their character-set to immense size and complexity.

There has been a tendency (Zipf's "law") for often-used words to become relatively short (monosyllabic), and so they can be written as a string of a few characters. Otherwise, the human brain seems to have been unconcerned about the effiency of information-transfer. Possible words (like "gevi" or "miro") go unused in English. There are some gestures toward economy, like the plural suffix "s" or the occasional use of the negator Latin prefix "un" (though we prefer "bad" to "ungood," and "not bad" is not the semantic equal of "good," while "untrue," "not true" and "false" have much the same meaning). Tolerance of linguistic inefficiency and inconsistency is the sign of a very complex interpreter. No interpreting machine can be so powerful (that's why I believe that artificial intelligence must differ from human intelligence).

The dilemma of every HLL is that it must be comprehensible to both human and a machine interpreter, a bridging compromise between extremely different systems. Hex or octal notations are close to the machine, while assembly language gets nearer the human level. All HLLs are at a higher level of symbolic complexity. Some (LISP, or FORTH) rise little above the assembly-language level, though modifications (SNOBOL4, and perhaps STOIC) may improve intelligibility. Others (BASIC, or Pascal) make greater concessions to the human interpreter.

**"Increasing power – with retention of compatability and human-intelligibility, and without exponential rises in interpreter size and complexity – demands nothing less than genius."**

The price of this trade-off is high. First, these require a more complex machine interpreter, over which the user has no control. Only experts can enhance it. Increasing power – with retention of compatibility and human-intelligibility, and without exponential rises in interpreter size and complexity – demands nothing less than genius. Second, and the consequence of the first, as they approach immutability, there will be a tendency to assume that nothing better is possible or needed. Limitations will not be seen, or if seen will be thought of as insuperables (as we now see the limiting velocity of light). Third, the great complexity and specialization will lower the probability of metamorphic change.

All this pleading for retaining machine-interpreter simplicity has been a prologue for a possibly metamorphic change in the FORTH command-coding. Bruce Airy did not push his analogy with CPU structure down to the bit-level. However, if a FORTH command-code is the analog of an opcode, each character of the code is the analog of a bit. The genii who design CPUs have learned how to cause a very small number of bits to control very complex operations. Since this complexity is not in the opcode bit-pattern, it has to be created by intricate (but highly efficient!) switching networks in the interpreting control-unit. The information in the opcode bit-pattern is used with an inhumanly-optimal efficiency. Can something like this be approximated for the information in a command-code?

The execution-logic requires parameters. Some are transferred to it (via the stack) by previous commands. Others are implied, residing in the logic itself. Thus, the OVER command has no indication within its 4 characters of the parameters used to execute ($S_2 \rightarrow S_0$). *Why not?* Consider the possible command-code ?S20, where ? is intended to signal an unusual operation to the FORTH interpreter, S acts as a human-mnemonic of a stack operation, and the numbers specify stack locations to be used by the S execution-logic. This code structure – *if* it can be rightly machine-interpreted – allows great generality. DUP would be replaced by ?S10, and presently non-existant analogs such as ?S30, ?S70, etc., will work. Furthermore, the execution logic could recognize that a non-zero value for the second parameter means an *exchange* instead of a copy into $S_0$. The existing SWAP would become ?S21. Or, going deeper into complexity, ?S21 could be executed as ($S_2 \rightarrow S_1$), while ?S12 would command a SWAP ($S_1 \leftrightarrow S_2$). The goal is to maximize the information content in the command-code. Since one can play with stack locations from $S_0$ to $S_9$, a very large number of distinct (though similar) operations become possible.

Since existing interpreter logic would see all these variants as completely different words, it would have to be taught that commands of the type ?SXY (where X and Y are different decimal digits) require special handling! In fact, one can imagine a whole class of codes starting with a ? but with characters other than the S to cause execution of quite different logic to use X and Y, even using XY as a 2-digit number if only one parameter (but a greater range) is needed. Existing FORTH suggests that implementation is feasible, since its compiler-imperatives don't work like other commands. I am not concerned here with implementation detail, but with the concept of making command-codes as information-rich as opcodes. Complexity is ultimately essential. Existing FORTH puts it in its dictionary and its programs. Why not put whatever you can in a more powerful kernel?

While the above change begins to break the ice of the now-frozen command-coding, true metamorphosis means exploiting an innovation all the way to its extreme limits. What would the seemingly senseless command ?S11 signify? Why not the existing DROP, that in my new notation is ($[S_1]$)? By extension, ?S22 would be a DROP of $S_2$, etc., reorganizing the stack to erase any one of the 9 specifiable locations. This still leaves structures like ?S02, that might be interpreted as a DROP of ($[S_2,S_1]$), i.e., more than one location. The meaningless ?S00 could be interpreted like the existing R command, copying the top of the return-stack into $S_0$. However, the details of interpretation cannot be made specific at the germinal-idea stage. The machine-code of the parametrized ?SXY primitive would be a complex, optimizable stack-manipulation. Its realistic limits would be revealed only during its creation, and act as a guide to optimal XY interpretations. Equally important would be the views of the designer concerning the options likely to be most frequently needed, influenced by the potential mnemonic value to the user.

During compilation, a ?SXY would first put the execution-address of the machine-code in the sequence, then the numerical XY perhaps compressed into a single byte, in any event not needing more than one location (like the IF, ELSE, and LOOP). This is a significant compression by comparison with the combinations approach, which would have to compile 2 LITs and their numerics, then the address of S stack-manipulation code, that would be more complex since it would have to pull the parameters from the stack.

We are not yet at the metamorphic limit! What of the information content in the S character? It's true that a minimum of one character is needed to find the execution-address of the stack-manipulation machine-code. But we have so far ignored the stack-rotation FORTH primitive, ROT, that in my notation is ($S_3 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3$) linearly representing a "circular" move. Note that "counter-rotation" like ($S_1 \rightarrow S_3 \rightarrow S_2 \rightarrow S_1$) is also conceivable, though it has no primitive in existing FORTH, and that the concept is extensible to more than three contiguous locations. It is impossible to decide *a priori* to what extent the machine-logic could be shared with that of the ?SXY operations. In any case, the limits of existing FORTH command-naming seem to demand a separate dictionary entry. If this were ?RXY, the ASCII numeric equivalent of R is 52, and of S 53. The R logic might enter the S logic with the low-nibble 2 "in its teeth" to modify interpretation to a rotation. E.g., ?R31 would replace the existing ROT, ?R13 would signify its reversed version, etc. Needs much thought! Complexity is being shifted into the machine-logic of fewer (but more information-rich) commands, lowering the requirement for complex sequences of a larger number of simple commands. The ROT must have been created for that very reason.

FORTH-like languages are still far below their potential evolutionary peak. Like other HLLs, FORTH came to life within one highly creative mind. Although he asserts that it "has been honed against the rock of experience," it has not yet been tempered in the fire of other creative minds. One sign that they are interested is the article by the Motorola designers Ritter and Walker on "threaded-code" (*BYTE* Sept. 1980 p. 206), recognizing how HLLs may become similar at the intermediate-code level and that it's there one might optimize a microprocessor to serve more than one HLL. I also expect a major impact on HLLs by the Japanese, with their utterly different spoken and written language. The non-Englishy quality of FORTH may prove to be an advantage at the international level.

**"I am . . . concerned . . . with the concept of making command-codes as information-rich as opcodes."**

**Is FORTH ready for widespread usage?** I start by conceding that the preceding remarks are amateurish. A few weeks of study do not suffice to make anyone an authority on a subject so complex! Perceptive readers will have noticed that my views underwent subtle change as my understanding gradually improved. My present feeling is that existing FORTH is not ready for use by novices. Charles Moore concedes (*BYTE* Aug. 1980 p. 76) that it is more "sensitive to the quality of the programmer" than are BASIC and FORTRAN, and that it is possible (I would say easy) to write "very bad FORTH code." Even a program as simple as my FIBO example (that I hope is good FORTH code!) lays considerable stress on the intelligence of the programmer. Stack-manipulation is trickier than using the fixed-location registers of a CPU (especially a simple one like the 6502). The action caused by a command-code primitive is less intelligible than that of most assembler mnemonics. The development of richer commenting conventions may help correct this, but I feel that greater utilization of the information-carrying capacity of the codes (hinted at in my ? type structures) may prove even more effective. Having in mind a clear image of the architecture of a CPU is invaluable in the writing of assembly-language programs, so I have tried – following in the footsteps of Bruce Airy – to construct one. It may be possible to enhance FORTH at its very core, where complexity and power might be maximized while the user-level constructs become fewer and easier-to-grasp (I cannot help thinking of this as the 8080 → 6502 transformation).

Nevertheless, even the most enhanced version of FORTH will have become a superior high-level assembly-language – machine-oriented, though independent of real-world machines to the extent that an HLL can be. Even if it has resisted the temptation of primitive-proliferation (the 8080 → Z80 direction), it will retain the error-proneness (all things are possible, most of them wrong) that the computer elite thinks is inherent in assembly-language. I shall quote some trenchant comments by G. J. Myers of the IBM Research Center in his book *Software Reliability* (1976, John Wiley & Sons, N.Y.):

*. . . in general, the "higher the level of language, the fewer the errors . . . . If a programmer is using a machine language, he spends most of his time not problem-solving but fumbling with the details of the machine . . . . Programs in high level languages are more understandable, easier to change, somewhat self-documenting, and enhance the compatibility and portability of programs . . . . The biggest advantage of high level languages is their ability to express and manipulate complex data-structures . . . manipulation of data is what programming is all about . . . efficiency is achieved through the intelligent selection of algorithms and data-structures . . . structured programming is* **the attitude of writing code with the intent of communicating with people instead of machines. . .**

These reflections must be based on a lot of bitter experience and ought to be kept in mind in any FORTH enhancement. In fairness to Dr. Myers, he is no less aware of the problems of HLLs that make the machine invisible to the programmer, especially IBM's baby, PL-I.

*The following issue of* DDJ *will feature Mr. Gordon's theoretical extension of the "parametrized command" concept. It will introduce new considerations and further develop some aspects presented in this article.* ■■

**Table of Architectural Analogies between FORTH and a CPU**
**(modified from original idea of Bruce Airy of UCSD)**

| | |
|---|---|
| bit (only 0 or 1) | ASCII character-set |
| opcode byte, 8-bit | command-code (word), variable length |
| instruction set (non-expansible) | command-set (vocabulary) primitives (expansible by user) |
| working registers (fixed number) | variables-stack (number indefinite, no definite accumulator) |
| index registers (fixed number) | temporary return-stack locations (may use evanescent → $S_0$) |
| stack pointer | return-stack pointer (kernel) |
| program counter | command counter (kernel) |
| program | sequence of execution-addresses of command-codes, and numeric operands |
| working memory | mostly within dictionary |
| control unit | control CPU-code programs (kernel) |
| status register (many specific bits) | only evanescent zero or non-zero → $S_0$ |
| JSR and RTS | : and ; primitives |
| load-#immediate | LIT primitive |
| JZ and JMP absolute | SKIP? and SKIP primitives, followed by address numeric |

*DDJ's* responsive readership
can give you invaluable feedback
on your ideas and innovations.
Find out how you can reach them
by contacting us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

**"We are not yet at the metamorphic limit!"**

**FORTH program to generate and print first 10 Fibonacci numbers.**

```
: FIBO 0 1 11 1 DO CR DUP . SWAP OVER + LOOP ;
```

(commented version, cf. text)

```
(FIBO)  : FIBO     (compile new program name)
        0          (#0 → S0, S1 = 0)
        1          (#1 → S0, S1 = 1, S2 = 0)
        11 1 DO    (set up loop, I = 1 to 10)
(LP-1)  CR         (CRLF to start next print line)
        DUP .      (print S1)
        SWAP       (S1 ↔ S2)
        OVER       (S2 → S0)
        +          ([S1 + S2] → S0)
        LOOP       (I = I + 1, if ≠ 11 return to LP-1)
(STOP)  ;          (end compilation)
```

(execution and output)

```
FIBO    (run new command name)
1
1
2
3
5
8
13
21
34
55
```

*(continued from page 6)*

which ran faster and smaller if the variables were defined near the top of the program (with a jump around them, of course). Any such program must keep track of forward references for later fixup, and that takes space and time. Furthermore, it is presumed that defining things first makes the programmer think about his data structures and get organized before coding.

I feel that Pascal has certain severe problems, including the non-obvious order of its writing and reading, and I deplore the fact that it was picked up by many as a panacea. Structured programming is a state of mind, not a set of rules. However, the stack orientation is one of the nicer parts of Pascal; smaller memory use and recursion are obtained for free, and a careful ordering of routines can cut memory use significantly. I think that the matter is still open to discussion.

Truly yours,
Harlan A. Rosenthal
Nicolet Scientific Corp.
245 Livingston St.
Northvale, NJ 07647

Dear Editor:

Number 43 has just hit Yap by slow boat. Without wishing to suggest that *DDJ*'s usual offerings are not high quality, nevertheless Dr. Knuth's article was filet mignon when one had expected ground round!

We linguists, you see, are another group that is constantly plagued with both text-formatting and font-design and -use problems, perhaps more so even than mathematicians. The center of my attention is dictionaries in foreign languages, and I have nearly despaired of writing programs to adequately handle my own data-management problems. Nevertheless, Dr. Knuth's books are my thread of hope, and his mention of a book on TEX is a ray of light in the darkness.

Yours truly,
John Thayer Jensen
P.O. Box 358
Yap, TT 96943

*Correction to* DDJ *#50, page 52:* Marilyn Darling's letter mentioned that Digital Research is compiling a list of CP/M-compatible software vendors for distribution to interested persons who contact them. A *DDJ* production error confused the last sentence, which should have let readers know that questions about this Digital Research project may be directed to Curt Geske at (408) 649-3896. ■■

# Of Interest

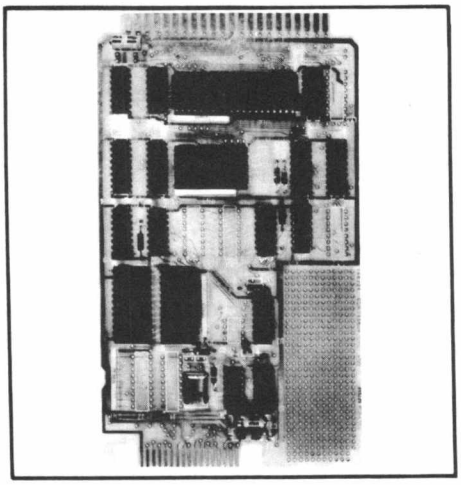
## Cassette Pascal For the Apple II

A cassette-based Pascal for the Apple II is now available from Dynasoft Systems. It is a compact p-code implementation of a Pascal subset which has been available for 6800 and 6809 microprocessors since the beginning of 1980. It is intended for systems which cannot support full-scale implementations such as UCSD Pascal.

The package sells for $50, which includes a user's manual and airmail postage. It is comparable in speed with Apple Integer BASIC and the system, including the compiler, interpreter and a line-oriented editor, occupies approximately 8K bytes. For further information, contact Dr. Allen Jost at Dynasoft Systems Ltd., P.O. Box 51, Windsor Junction, Nova Scotia, Canada, B0N 2V0.

## $199 Computer

RCA Microboard Products has added to its line a "complete computer system" on a 4.5 x 7.5 inch card. It contains a CDP1802 CMOS Microprocessor, a 2-MHz crystal-controlled clock, 512 bytes of read-write memory, parallel I/O ports, power-on reset, an interface expansion area and a socket for 1 or 2 kilobytes of user-selected ROM. The CDP18S604 has the inherent advantages of low-power static CMOS and the COSMAC microprocessor architecture. Powered from a 5-volt supply, it requires only 4 milliamperes when populated with a CMOS ROM and is designed for a variety of low-cost computer applications. It is expandable by use of the 44-pin COSMAC Microboard Universal Backplate. Information may be obtained from RCA Solid State Division, Box 3200, Somerville, NJ 08876.



RCA's "complete computer" for $199.

RCA has also introduced the industry's first 10V CMOS 4K RAM. Its features include silicon-on-sapphire CMOS technology, low standby and operating power, extended temperature range (−40°C to +85°C), wide operating voltage range (4 to 10.5 volts) and industry standard 1024x4 pinouts.

Further information, and a copy of the CDP1825 data sheet, file # 1247, may be obtained by writing to the RCA address listed above.

RCA has reduced prices on a number of other products. Their CDP18S601 Microboard Computer (4-kilobyte RAM; sockets for 4/8-kilobyte ROM/PROM) has been reduced from $549 to $349. Also, their CDP18S623 Microboard (8-kilobyte RAM), that now sells for $495, was previously $650.

## CP/M-86 Operating System

Digital Research has introduced CP/M-86 for Intel 8086/8088-based microcomputers. This is the first of Digital Research's sixteen-bit products, a high-performance, single-user operating system designed to take advantage of the 8086's address space and speed while expanding upon the facilities of standard CP/M. The file format of CP/M, release 2, has been retained for compatibility. CP/M-86 can also serve as a slave node in a CP/NET network.

As with previous versions, logical and hardware-dependent portions are modularized for ease in configuration. For details, contact Harold Elgie at (408) 649-3896.

## PET Users Magazine

PET users may find of interest the British publication, *PRINTOUT*. It is a slick-format magazine, published ten times a year. It is devoted to consumer reports of hardware and peripherals, unbiased software reviews, applications, and software problems. Subscriptions are $33 per year from P.O. Box 48, Newbury, Berkshire RG16 0BD, England.

## IEEE-696 Prototype Card

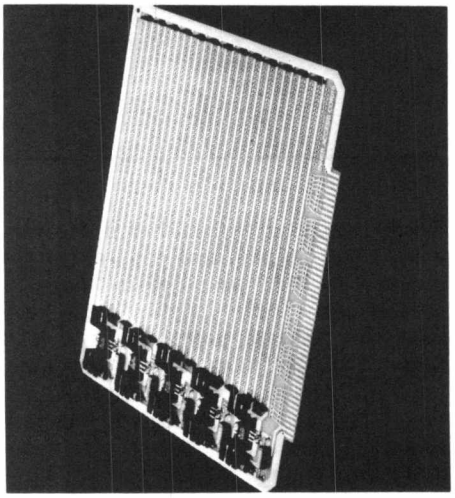
A nine-inch deep, S-100 wire-wrap prototyping card is available from the Inner Access Corporation for the IEEE-696 bus standard, compliance H. The company expects compliance H to become the preferred board size for both industrial and commercial S-100 users since it nearly doubles the area previously available to the designer (potential users should inquire how to fit one onto an older mainframe). It will accommodate over one hundred sixteen-pin wire-wrap sockets or other wire-wrap sockets on 0.3″ or 0.6″ centers.

The board comes complete with on-board regulators which provide 5 volts at 3 amps and ±12 volts at 1 amp. Twenty-six 0.1 ufd decoupling capacitors are distributed at either end of the board. Heretofore, no H-size prototyping boards have been available. The board costs $98 singly, from Inner Access Corporation, P.O. Box 888, Belmont, CA 94002.

## Word Processing Report

"Word Processing on Personal Computers" is the first in a series of evaluation reports published by the Small Systems Group. This report introduces personal computer word processing with sections on software, hardware and applications. It compares Auto Scribe, Electric Pencil, Magic Wand and Wordstar in terms of quality of documentation, ease of learning, editing power and formatting power. To do so, 159 features were analyzed, and the final section tabulates and discusses these in detail.

Single copies are available in North America for $10 and for $12 elsewhere. They may be ordered from the Small Systems Group, Box 5429, Santa Monica, CA 90405.



H-size prototyping board for the S-100.

# Z-80 File Comparator

by Donald E. Faulkner

Many times in the last few years, I've found myself in need of a routine permitting a byte-by-byte comparison between two files in memory. Such comparisons between old and newly purchased "improved" versions of programs can quickly point to the locations of any modifications which may have been made. Then, when the addresses have been ferreted out, the disassembler generally makes quick work of disclosing the nature of such program changes. The routine is very handy also for validating the source code generated by disassemblers. Often, portions of disassembled object code turn out to be data tables, ASCII strings, etc., rather than valid instructions. After correcting the new source listings to properly recreate the tables and strings, the code can be validated by reassembling into object code and comparing with the original machine code.

In the past, these comparisons were handled on my system by a routine written in BASIC. Although the BASIC program was slow, its foot-dragging pace didn't really become obvious on files containing many dissimilarities. The venerable teletype, as it listed the addresses of discrepancies, masked the slowness of the actual comparisons with its own slow-motion gyrations. When two long, virtually identical files were compared, however, the interpreter's slowness became more evident. With no dissimilar address being listed, one could really begin to appreciate just how slowly the wheels of progress ground inside the computer cabinet.

It was partly this which caused me to write the accompanying program. Actually, the bulk of the stimulation to do so resulted from my own housekeeping problems; I lost the BASIC routine.

This program is written to run on a Z-80 CPU. My own system is a Digital Group Z-80 with 50K of RAM using digital tape storage running under the PHIMON operating system. The video display uses a 64 character by 16 line screen format. For convenience, several of the PHIMON operating system's subroutines have been used. However, these have been accessed through a series of "equates" at the beginning of the program to make adapting it to another system somewhat easier.

In operation, the routine requests beginning and ending addresses of the first file in RAM and the beginning address of the second file to which it is to be compared. The printer routine may be activated by preceding the entry of these addresses with a CONTROL/P. CONTROL/O may later be used at this same point to "turn off" the printer. The addresss entries must be followed by a carriage return to initiate the comparison process. (Entry errors may be corrected with the usual RUB-OUT.) The routine lists the split-octal addresses, if any, within the first file at which dissimilar bytes have been located. The last split-octal address listed is followed by the message "Term. ." indicating that the comparison is complete.

After listing a block of fourteen lines of eight addresses each, the routine pauses. Pressing the spacebar clears the screen and causes the listing to resume; pressing ESCAPE terminates the listing and exits to the operating system. Other keystrokes are ignored. When the listing of addresses is complete, the operator may repeat the process, selecting new addresses, by pressing the spacebar. ESCAPE will, instead, exit to the operating system.

Now, looking at the subroutines called from the PHIMON operating system:

**OPRSYS** Must be the entry point of the computer's own operating system.

**KEYBDO** Is the address of a keyboard entry subroutine which, when called, accepts a character from the keyboard and returns it in the A-register.

**TV** Is a subroutine which, when called, outputs the character in the A-register to the screen. The A-register then returns a null.

**ERASE** Must clear the screen and arrange for the next character to be at the screen's upper left (i.e., "Home" the cursor).

**LINBUF** Is the address of any 32-byte-long area in RAM which may be used as a command buffer.

**DELAY** Is a subroutine which causes a time delay, when called, which is proportional to the constant pre-loaded into the A-register. If you have no time delay routine in your system, a simple delaying subroutine is given in listing two.

**TVEDT** Outputs to the screen the ASCII character pointed to by the HL register pair. It then increments HL and repeats the process until a null byte is encountered. At this point it returns to the calling routine. ASCII characters must have most-significant-bits set. Bytes with msb's not set are interpreted as multiple spaces. (An octal 101 produces 65 spaces, while an octal 301 prints the character "A".) A subroutine which does this is given in listing three.

If you're not running a Digital Group system, replace the TOGGLE subroutine in the program with one which will toggle the cursor position back 20 spaces on your screen. If you don't have this backward-cursor-movement capability, remove the subroutine (lines 440-520) and enter a data byte of 054 octal between lines 1970 and 1980 of the source code. This modification will allow the addresses entered to fall below the format-prompting string on the screen rather than overwriting it. Finally, if your printer doesn't automatically generate a line-feed with each carriage-return it receives, then load the A-register with octal 212 and call HCOPY between lines 1500 and 1510 of the source code.

If you don't already have a comparison utility on your system, now is the time to add one. I hope that you'll find it as useful as I have, perhaps even taking its place right alongside your search utility as an indispensable software tool.

■■

(LISTING ON PAGE 37)

*Donald E. Faulkner, 1245 Springdale Drive, Jackson, Mississippi 39211.*

# Z-80 Video Drivers


## Abstract

Memory-mapped video displays are widely used in microcomputer systems. They afford considerable output flexibility and are generally less expensive than a standard CRT unit. The major advantage of the memory-mapped display is the fact that the contents of the display may be accessed like any other memory in the microprocessor. This capability is lacking in CRT's. However, standard terminal functions must be provided for by software. This software has to be designed carefully and with the desired application in mind, in order to provide maximum capability and efficiency.

In this article several video drivers are included in their complete form. First a basic implementation is presented to exemplify the general concepts behind the design of such a driver. Then, a full implementation is presented along with discussion of the features and approaches used to implement them. As part of the discussion, a special purpose driver is included that handles cursor positioning in a Life simulation program.


## Introduction

Memory-mapped video displays are a common element to many microcomputer systems. This variety of display devices offers significant versatility in the presentation of information and still remains a low-cost alternative to CRT displays. However, unlike CRT's, memory-mapped displays require non-trivial software drivers to provide the normal display functions such as scrolling, backspacing and cursor positioning. Many factors must be considered in the design stage in order to provide a useful and efficient video driver. Among these design considerations are display speed limitations, output formatting and the scope of included features.

## Strategy in Design

The drivers presented in this article utilize a main routine that also serves as the entry point for calling routines. The main routine provides common information about the cursor position to each subroutine, which is used to execute the display features. It is responsible for decoding commands, for executing appropriate subroutines and for handling normal character output and scrolling or paging. As well, it handles removal and replacement of the cursor character on the screen. The subprograms change the cursor address and manipulate the video memory in various ways. They are required to follow a set convention for parameter passing. The only parameter passed is the cursor address. In Figure 1, the layout of the video drivers is outlined. Each utilizes the same essential structure including a main loop and support routines.


by Anthony Skjellum



*Copyright © 1980 by Anthony Skjellum, 1695 Shenandoah Road, San Marino, CA 91108.*


## A First Attempt

A design specification is useful in producing drivers of any kind. It is wise to outline the functions to be included in the primary implementation. Additional features may be included once the basic driver is fully functional. Table Ia lists a set of minimal functions for a first implementation. These functions will be required to handle normal TTY emulation.

Example I is a listing for a minimum driver called MNDRIVR. It supports the features enumerated in Table Ia as well as those of Table Ib. As it stands, this driver is very useful. Its main virtue is speed. MNDRIVR was intended to be an example of a first implementation. Expandability and maximum throughput were considered equally important in its design.

In order to make MNDRIVR as efficient as possible, the parameter passing conventions were made simple. The HL register contains the cursor address on entry to the support routines; they are expected to update this value in a meaningful way. Because the other registers may be modified at will, there is no need for cumbersome push-pop sequences at the beginning and end of support routines. This makes interfacing support routines much more straightforward and avoids the additional overhead that would otherwise be required.

## Scrolling

Scrolling is the rate dependent step in text display, therefore extra attention has been put toward minimizing the execution time of SCRL and to decoding the linefeed character (i.e., by looking for it first). If scrolling could be removed, the display speed would be much faster. This can be done readily. Instead of scrolling, the cursor is made to home. No display memory need be moved. The only disadvantage is a slight loss in readability which I feel is made up for by the gain in throughput. Through conditional assembly, the paging option may be selected in MNDRIVR.

For a first attempt at a video driver, MNDRIVR has provided the essential functions necessary for emulating TTY operation and has also addressed a design tradeoff between scrolling and display speed.

## A General Purpose Driver

Because of MNDRIVR's modular construction, additional features can be added in a logical way while the efficient inner loop is maintained. Many useful functions may be added with little additional overhead. In the next sections, several specific features are discussed including negative scrolling (and negative linefeeds), line insert-delete, cursor addressing as well as "linefeed collection," a process by which scrolling is made more efficient. All of these features are included in the comprehensive program, GPDRIVR, in Example II. It is not intended that all of these features be incorporated in all applications. Therefore, desired options are selectable via conditional assembly toggles.

## Cursor Motion Functions

It is often necessary to move the cursor around on the display. This is facilitated by four-directional cursor motion. The standard linefeed function provides the down motion; up, left and right are trivial to implement as well.

For special purpose drivers, it may be useful to move the cursor diagonally as well. Take, for example, a program that performs a Life simulation. Such a program could allow the user to place "cells" on the display by moving the cursor and pressing a key. Such a driver is included in Example III. It allows the cursor to be moved in any of eight directions.

## Reverse Scrolling

Reverse scrolling provides a logical way to handle negative linefeeding, backspacing or cursor positioning, which causes the cursor to move above the home position and off the screen. In MNDRIVR this condition is handled by placing the cursor at the home position. In GPDRIVR, either reverse scrolling or homing can be selected by conditional assembly.

**TABLE Ia**

I. Carriage return
II. Linefeed
III. Backspacing
IV. Scrolling
V. Top of page detection

**TABLE Ib**

I. Clear screen
II. Home cursor
III. Scrolling or paging option

## Linefeed Collection

In normal text output there are occasions that require multiple carriage return-linefeed sequences without intervening characters. In such a case it is possible to "save up" the scrolling and perform it all at once. Each scroll requires that 960 bytes of memory be moved and that the bottom line be filled with blanks. A multi-line scroll moves fewer characters and correspondingly fills more lines. Linefeed collection (or "scroll-control" as it is called in GPDRIVR) provides this feature. This feature is not available when the paging option is selected. Furthermore, linefeed collection only occurs when the cursor is at the bottom left corner of the display.

## Other Special Functions

For sophisticated text display, there are several other key features that need to be included. They include delete to end-of-line, delete to end-of-screen, cursor addressing, line insert and line delete. These features require a greater overhead than some of the more conventional functions including a GPDRIVR. Therefore, an "escape sequence" is necessary to access these functions. An escape character (ESC) is sent prior to the command character. By using this technique, the main routine need not decode these additional commands but instead controls a smaller overhead for the escape mode.

## Summary

In this article, many aspects of CRT emulation for memory-mapped displays have been explored. Other sophisticated features, such as display windows and the like, were considered beyond the scope of a routine that was to be driven via character-by-character input. Features such as point and line plotting could be included as part of the escape sequence functions in GPDRIVR. Such an undertaking is quite feasible but was considered too specific for inclusion here.

## Implementation Notes

MNDRIVR and GPDRIVR were both assembled under MAC. They utilize the standard Z80 library (Z80.LIB). They also require SUPPORT.LIB, which is presented in Listing I. The macro TEST is a short test program that echoes console input to the video driver for test purposes. In a normal application, this would be eliminated.

CRT in Example III was written for the TDL MACRO ASSEMBLER. ■■

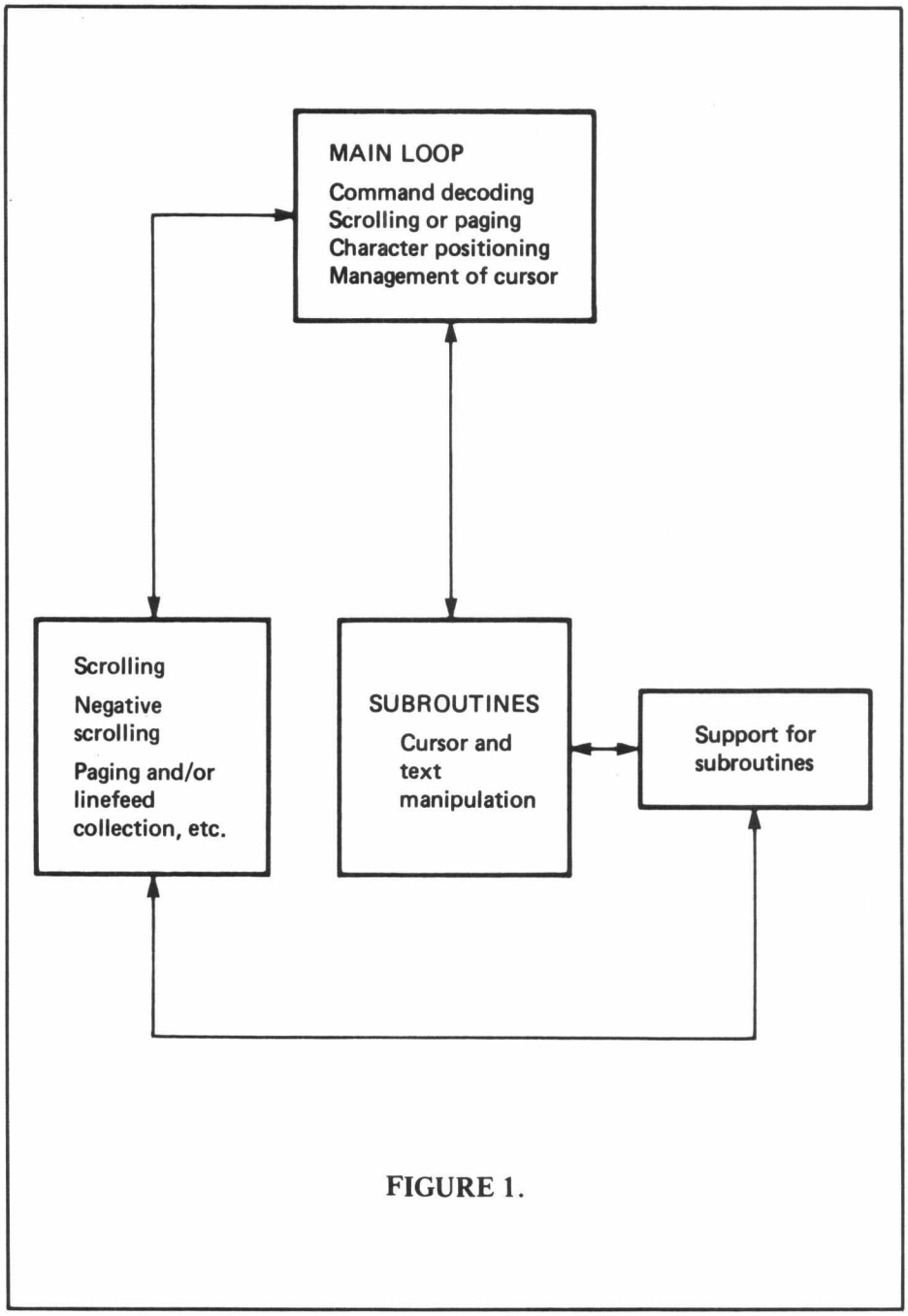



FIGURE 1.

# Z-80 Video Drivers

## Example I

```
                        **********************
                        ********MNDRIVR*******
                        **********************



0100                    ORG     0100H           ; TPA START
                        ;
                        MACLIB  Z80
                        MACLIB  SUPPORT
                        ;
                MAIN:   TEST                    ; EXEC TEST PROGRAM
0100+315C01
0103+2A0100
0106+110600
0109+19
010A+220E01
010D+CD0000
0110+CBBF
0112+FE03
0114+CA0000
0117+CD5C01
011A+18F1
011C+
                        ;
                        ;
                        ;
                        ;
                        ;
FFFF =          SCROL   EQU     ON      ; SELECT SCROLLING
0000 =          SOLID   EQU     OFF     ; SOLID CURSOR OPTION
                        ;
                        ;
                        ;
                        ;
                        ;
                ;*******************************************************
                ;*                                                     *
                ;*      MNDRIVR.ASM                                    *
                ;*                                                     *
                ;*      SOFTWARE DRIVER FOR VECTOR GRAPHIC'S           *
                ;*      FLASHWRITER 16X64 DISPLAY.                     *
```

```
0165 D5                 PUSH    D       ; SAVE IT ON STACK
                        ;
                        ;
0166 FE0A               CPI     LF      ; IS INPUT CHAR A LINEFEED?
                        JRZ     LFEED   ; IF SO, THEN JUMP...
0168+2849
016A FE0D               CPI     CR      ; IS INPUT CHAR A CARRIAGE RETURN?
                        JRZ     RTN     ; IF SO, THEN JUMP...
016C+2840
016E FE08               CPI     BK      ; IS INPUT CHAR A BACKSPACE?
                        JRZ     DELT    ; IF SO, THEN JUMP...
0170+2856
0172 FE7F               CPI     DEL     ; IS INPUT CHAR A DEL?
                        JRZ     DELT    ; IF SO, THEN JUMP...
0174+2852
0176 FE06               CPI     CHCLS   ; IS INPUT CHAR A CLEAR SCREEN REQUEST?
                        JRZ     CLEAR   ; IF SO, THEN JUMP...
0178+283E
017A FE05               CPI     CHHME   ; IS INPUT CHAR A HOME REQUEST?
                        JRZ     HOME    ; IF SO, THEN JUMP...
017C+2846
                        ;
                        ;
017E D1                 POP     D       ; REMOVE SPURIOUS TOP OF STACK
                        ;
                        ;
                                ;
                        ;
017F 77                 MOV     M,A     ; STUFF CHAR INTO DISPLAY MEMORY
0180 23                 INX     H       ; INCR. CURSOR
0181 7C         VIDR:   MOV     A,H     ; CHECK IF SCROLL NEEDED
0182 FED0               CPI     DPAGE   ; SEE IF HL TOO SMALL
                        JRC     TOP     ; JUMP IF TOO SMALL
0184+380C
0186 FED4               CPI     EPAGE   ; SEE IF HL TOO BIG
0188 D49701             CNC     SCRL    ; YES
                VIDR2:
                        SETB    7,M     ; REVERSE VIDEO CURSOR COND.
018B+CBFE
                                        ;
                        ;
018D 22CC01             SHLD    CURSOR  ; SAVE CURSOR ADDRESS
0190 F1         VIDR3:  POP     PSW
0191 C9                 RET
                        ;
0192 2100D0     TOP:    LXI     H,DSPL
                        JMPR    VIDR2   ; REENTER MAIN PROGRAM
0195+18F4
                        ;
                        ;
                        ;
                        IF      SCROL   ; SCROLLING OPTION
                        ;
                        ;
```

```
                ;*                                                    *
                ;*                                                    *
                ;*      BY ANTHONY SKJELLUM                           *
                ;*      1695 SHENANDOAH RD.                           *
                ;*      SAN MARINO, CA   91108                        *
                ;*                                                    *
                ;*      COPYRIGHT 1980 (C) ANTHONY SKJELLUM.          *
                ;*      ALL RIGHTS RESERVED.                          *
                ;*                                                    *
                ;*                                                    *
                ;*      VERSION 1.00    --> JANUARY 1979              *
                ;*      VERSION 2.00    --> JULY    1980              *
                ;*                                                    *
                ;*      THIS IS A MINIMAL DRIVER FOR THE FLASHWRITER  *
                ;*      DISPLAY.                                      *
                ;*                                                    *
                ;*      ASSEMBLE WITH MAC AND SUPPORT.LIB.            *
                ;*                                                    *
                ;******************************************************
                ;
                ;
                ;
                ;******************************************************
                ;*                                                    *
                ;*      VIDEO:                                        *
                ;*                                                    *
                ;*      EXPECTS:        ACCUMULATOR W/ CHAR TO OUTPUT *
                ;*                      OR CONTROL CODE.              *
                ;*                                                    *
                ;*      RETURNS:        CURSOR POSITIONED OR CHARACTER*
                ;*                      OUTPUT TO SCREEN              *
                ;*                                                    *
                ;*      CONSUMES:       BC,DE,HL                      *
                ;*                                                    *
                ;******************************************************
                        ;
015C F5         VIDEO:  PUSH    PSW     ; SAVE INPUT CHAR
                        ;
015D 2ACC01             LHLD    CURSOR  ; CURSOR VALUE
                        ;
                        IF      SOLID   ; IF SOLID CURSOR OPTION
                        MOV     B,A     ; SAVE INPUT CHAR
                        LDA     UNDR    ; GET CHAR UNDER CURSOR
                        MOV     M,A     ; PLACE IN VIDEO RAM
                        MOV     A,B     ; REVIVE INPUT CHAR
                        ELSE
                        RES     7,M     ; IF REVERSE OPTION, TURN OFF REVERSE
0160+CBBE                                                       MODE
                        ENDIF
                        ;
                        ;
                                        ;
0162 118101             LXI     D,VIDR  ; POINT TO STANDARD RETURN ADDRESS
```

*(continued on top of page 22)*

```
0197 2140D0     SCRL:   LXI     H,DSPL+64       ; SOURCE
019A 1100D0             LXI     D,DSPL          ; DESTINATION
019D 01C003             LXI     B,960           ; COUNT
                        LDIR
01A0+EDB0
                        ;
01A2 EB                 XCHG                    ; HL HAS DEST.
01A3 012040             LXI     B,4020H         ; B:= 64, C:=SPACE
01A6 71         SCR:    MOV     M,C             ; FILL
01A7 23                 INX     H               ; POINT TO NEXT
                        DJNZ    SCR             ; CONT. UNTIL DONE
01A8+10FC
01AA 21C0D3             LXI     H,DSPL+960      ; RESET CURS. ADDRESS
01AD C9                 RET                     ; AND EXIT.
                        ;
                        ;
                        ELSE
                SCRL    EQU     HOME            ; ELSE PAGE OPTION IS ON
                        ;
                        ENDIF
                        ;
                        ;
                ;******************************************************
                ;*                                                    *
                ;*      SUPPORT ROUTINES.                             *
                ;*                                                    *
                ;******************************************************
                        ;
01AE 7D         RTN:    MOV     A,L             ; GET LOW ORDER OF CURS. ADDR
01AF E6C0               ANI     0C0H            ; MASK OUT LINE POS.
01B1 6F                 MOV     L,A             ; REPLACE IT.
01B2 C9                 RET                     ; AND RETURN
                        ;
01B3 114000     LFEED:  LXI     D,64            ; DO CURS+= 64;
01B6 19                 DAD     D               ; ADD QUANT IN HL
01B7 C9                 RET                     ; AND EXIT.
                        ;
                        ;
                        ;
                        ;
01B8 2100D0     CLEAR:  LXI     H,DSPL          ; POINT TO START OF SCRN
                                                ; MEMORY
01BB 0120D4             LXI     B,ESPL+SPACE    ; B:=EPAGE, C:=SPACE;
                        ;
                                                        ; C:=SPACE.
01BE 71         CL1:    MOV     M,C
01BF 23                 INX     H               ; PERFORM LOAD AND INCR.
01C0 7C                 MOV     A,H             ; DO END CHECK
01C1 B8                 CMP     B               ; SEE IF WE'RE DONE
                        JRNZ    CL1             ; CONTINUE IF NOT DONE
```

*(continued on next page)*

# Z-80 Video Drivers

```
01C2+20FA
01C4 2100D0     HOME:   LXI     H,DSPL          ; SET CURS ADDR.
01C7 C9                 RET                     ; AND EXIT.
                        ;
                        ;
                        ;
                DELT:
01C8 3620               MVI     M,SPACE ; AND CLEAR W/ SPACE
01CA 2B                 DCX     H
01CB C9                 RET
                        ;
                        ;
                        ;
                ;*********************************************************
                ;*                                                       *
                ;*      RAM AREA                                         *
                ;*                                                       *
                ;*********************************************************
                        ;
01CC 00D0       CURSOR: DW      DSPL            ; POINT TO START OF SCRN
                        IF      SOLID           ; SOLID CURSOR OPTION
                UNDR:   DB      0
                        ENDIF
                        ;
                        ;
                        ;
                        ;
                        ;
01CE                    END
```

```
                        **********************
                        *****SYMBOL TABLE*****
                        **********************
0000 BC         0008 BK         0019 CHADR      0016 CHCLN      0017 CHCLOZ
0006 CHCLS      0014 CHEOL      0013 CHEOS      001B CHESC      0005 CHHME
000C CHLFT      0018 CHOPEN     0012 CHRGHT     0015 CHUP       01BE CL1
01B8 CLEAR      000D CR         01CC CURSOR     0002 DE         007F DEL
01C8 DELT       D3C0 DLINE      00D0 DPAGE      D000 DSPL       00D4 EPAGE
D400 ESPL       00D8 FPAGE      0004 HL         01C4 HOME       0004 IX
0004 IY         000A LF         01B3 LFEED      0100 MAIN       0000 OFF
FFFF ON         01AE RTN        0197 SCRL       FFFF SCROL      01A6 SCR
0000 SOLID      0020 SPACE      0192 TOP        015C VIDEO      0181 VIDR
018B VIDR2      0190 VIDR3
```

**end**

```
                ;
CS      EQU     05FH    ; DEFINE SOLID CURSOR AS UNDERLINE CHAR
                ;
                ENDIF
                ;
                ;
                ;
                ;
                ;
                ;
                ;
;*********************************************************
;*                                                       *
;*      GPDRIVR.ASM                                      *
;*                                                       *
;*      SOFTWARE DRIVER FOR VECTOR GRAPHIC'S             *
;*      FLASHWRITER 16X64 DISPLAY.                       *
;*                                                       *
;*                                                       *
;*      BY ANTHONY SKJELLUM                              *
;*      1695 SHENANDOAH RD.                              *
;*      SAN MARINO, CA   91108                           *
;*                                                       *
;*      VERSION 1.00:   JULY 1980                        *
;*                                                       *
;*      COPYRIGHT 1980 (C) ANTHONY SKJELLUM.             *
;*      ALL RIGHTS RESERVED.                             *
;*                                                       *
;*      THIS IS A FULL VIDEO DRIVER FOR THE              *
;*      FLASHWRITER DISPLAY.                             *
;*                                                       *
;*                                                       *
;*      ASSEMBLE WITH MAC AND SUPPORT.LIB.               *
;*                                                       *
;*********************************************************
;
;
;
;*********************************************************
;*                                                       *
;*      VIDEO:                                           *
;*                                                       *
;*      EXPECTS:        ACCUMULATOR W/ CHAR TO OUTPUT,   *
;*                      CONTROL CODE OR ESCAPE CODE.     *
;*                                                       *
;*      RETURNS:        CURSOR POSITIONED OR CHARACTER   *
;*                      OUTPUT TO SCREEN                 *
;*                                                       *
;*      CONSUMES:       BC,DE,HL,X                       *
;*                                                       *
;*********************************************************
                ;
```

# Example II

```
                        ******************
                        *****GPDRIVR*****
                        ******************




0100                    ORG     0100H           ; TPA START
                        ;
                        MACLIB  Z80
                        MACLIB  SUPPORT
                        ;
                MAIN:   TEST                    ; EXEC TEST PROGRAM
0100+315C01
0103+2A0100
0106+110600
0109+19
010A+220E01
010D+CD0000
0110+CBBF
0112+FE03
0114+CA0000
0117+CD5C01
011A+18F1
011C+
                        ;
                        ;
FFFF =          @CNTRL  EQU ON          ; SCROLL-CONTROL TOGGLE
FFFF =          @SCROL  EQU ON          ; SELECT SCROLLING
0000 =          @SOLID  EQU OFF         ; SOLID CURSOR OPTION
FFFF =          @ESCMOD EQU ON
0000 =          @SECOND EQU OFF         ; PUT SECOND LEVEL FNS IN ESC MODE.
FFFF =          @NEGSC  EQU ON          ; NEGATIVE SCROLLING ENABLED
                        ;
                        ;
                        ;
                        ;
                        ;
                        IF   NOT @SCROL
                        IF   @CNTRL  '--<Inconsistency in assembly toggles>--'
                        ENDIF
                        ENDIF
                        ;
                        IF   @SOLID
```

*(continued on top of page 24)*

```
015C F5   VIDEO:        PUSH    PSW     ; SAVE INPUT CHAR
                        ;
015D 2A2D03             LHLD    CURSOR  ; CURSOR VALUE
                        ;
                        IF      @SOLID
                        MOV     B,A     ; SAVE INPUT CHAR
                        LDA     UNDR    ; GET CHAR UNDER CURSOR
                        MOV     M,A     ; STORE IT BACK IN DISPLAY MEMORY
                        ;
                        ELSE
                        RES     7,M     ; REMOVE REVERSE VIDEO STATUS
0160+CBBE
                        ENDIF
                        ;
                        ;
0162 11A101             LXI     D,VIDR  ; POINT TO STANDARD RETURN ADDRESS
0165 D5                 PUSH    D       ; SAVE IT ON STACK
                        ;
                        ;
                        ;
                        ;
                        IF      @ESCMOD ; IF CURSOR ADDRESS OPTION
                        ;
                        IF      NOT @SOLID
0166 47                 MOV     B,A     ; SAVE INPUT CHAR
                        ENDIF
                        ;
0167 3A3203             LDA     ECHK    ; GET ESCAPE MODE FLAG
016A B7                 ORA     A       ; SET FLAGS
                        JRZ     NOESC   ; NO ESCAPE CONDITION
016B+280D
016D FA5F02             JM      ESCSET  ; SET ESCAPE MODE
0170 3D                 DCR     A       ; DECREMENT COUNT
0171 323203             STA     ECHK    ; UPDATE
                        LIXD    EWORD   ; GET ADDRESS TO EXECUTE.
0174+DD2A
0176+3003
                        PCIX            ; GO THERE
0178+DDE9
017A 78   NOESC:        MOV     A,B     ; REVIVE INPUT CHAR
                        ENDIF
                        ;
                        ;
                        ;
                        ;
017B FE0A               CPI     LF      ; IS INPUT CHAR A LINEFEED?
                        JRZ     LFEED   ; IF SO, THEN JUMP...
017D+2839
017F FE0D               CPI     CR      ; IS INPUT CHAR A CARRIAGE RETURN?
                        JRZ     RTN     ; IF SO, THEN JUMP...
0181+2830
```

*(continued on next page)*

# Z-80 Video Drivers

```
                        ;
                        IF      @CNTRL
                        ;
                        IF      NOT @ESCMOD
                        MOV     B,A     ; SAVE TEMP COPY.
                        ENDIF
                        ;
0183 3A2F03             LDA     CNT     ; NEED SCROLLS?
0186 B7                 ORA     A       ; SET FLAGS
0187 C4E201             CNZ     MLSCRL  ; PERFORM MULTI-LINE SCROLL
                        ;
018A 78                 MOV     A,B     ; REVIVE COPY
                        ENDIF
                        ;
                        ;
018B FE08               CPI     BK      ; IS INPUT CHAR A BACKSPACE?
                        JRZ     DELT    ; IF SO, THEN JUMP...
018D+284F
018F FE7F               CPI     DEL     ; IS INPUT CHAR A DEL?
                        JRZ     DELT    ; IF SO, THEN JUMP...
0191+284B
                        ;
                        IF      @SECOND ; IF SECOND LEVEL FNS RUN DIRECTLY...
                        CPI     CHCLS   ; IS INPUT CHAR A CLEAR SCREEN REQUEST?
                        JRZ     CLEAR   ; IF SO, THEN JUMP...
                        CPI     CHHME   ; IS INPUT CHAR A HOME REQUEST?
                        JRZ     HOME    ; IF SO, THEN JUMP...
                        ;
                        CPI     CHUP    ; IS INPUT CHAR AN UP REQUEST?
                        JRZ     NLFEED  ; IF SO, THEN JUMP...
                        CPI     CHLFT   ; IS INPUT CHAR A LEFT REQUEST?
                        JRZ     LEFT    ; IF SO, THEN JUMP...
                        CPI     CHRGHT  ; IS INPUT CHAR A RIGHT REQUEST?
                        JRZ     RIGHT   ; IF SO, THEN JUMP
                        ;
                        ENDIF
                        ;
0193 D1                 POP     D       ; REMOVE SPURIOUS TOP OF STACK
                        ;
                        IF      @ESCMOD ; IF ESCAPE MODE...
0194 FE1B               CPI     CHESC   ; IS INPUT CHAR AN ESCAPE CHARACTER?
                        JRNZ    NOCODE  ; IF NOT, THEN JUMP...
0196+2007
0198 3E80               MVI     A,128   ; SET FOR LATER IDENTIFY
019A 323203             STA     ECHK    ; STORE IT
                        JMPR    VIDR    ; LEAVE AS QUICKLY AS POSSIBLE...
019D+1802
019F 70         NOCODE: MOV     M,B     ; PLACE INPUT CHAR DIRECTLY IN DISPLAY MEM.
```

```
01BD 11C0FF     NLFEED: LXI     D,-64   ; DO CURS-= 64;
01C0 19                 DAD     D       ; ADD QUANT IN HL.
01C1 C9                 RET             ; AND EXIT.
                        ;
01C2 23         RIGHT:  INX     H       ; DO ++CURS;
01C3 C9                 RET             ; AND EXIT.
                        ;
01C4 2B         LEFT:   DCX     H       ; DO --CURS;
01C5 C9                 RET             ; AND EXIT.
                        ;
                        ;
                        ;
                        ;
                        IF      NOT @ESCMOD
                CLEAR:  LXI     H,DSPL          ; POINT TO START OF SCREEN MEMORY
                        LXI     B,ESPL+SPACE    ; B:=EPAGE, C:=SPACE.
                CL1:    MOV     M,C             ; FILL
                        INX     H               ; PERFORM INCR.
                        MOV     A,H             ; DO END CHECK
                        CMP     B               ; SEE IF WE'RE DONE
                        JRNZ    CL1             ; CONTINUE IF NOT DONE
                        ENDIF
01C6 2100D0     HOME:   LXI     H,DSPL          ; NEW CURSOR POS.
01C9 C9                 RET                     ; AND EXIT.
                        ;
                        IF      @ESCMOD
01CA 2100D0     CLEAR:  LXI     H,DSPL          ; START FOR CLEAR AND NEW ADDRESS
                                                ; DELETE TO END OF SCRN FOR CURSOR
01CD E5         DEOS:   PUSH    H               ; SAVE IT
01CE CDD301             CALL    CL2
01D1 E1                 POP     H
01D2 C9                 RET                     ; PERFORM OPERATION AND EXIT.
01D3 0120D4     CL2:    LXI     B,ESPL+SPACE    ; C:= SPACE, B:= EPAGE;
01D6 78         BFILL:  MOV     A,B     ; GET TEST VALUE IN ACC.
01D7 71         BFILL2: MOV     M,C     ; FILL
01D8 23                 INX     H       ; POINT TO NEXT
01D9 7C                 MOV     A,H     ; GET HIGH ORDER OF HL
01DA B8                 CMP     B       ; SEE IF WE'RE DONE
                        JRNZ    BFILL2
01DB+20FA
01DD C9                 RET             ; AND EXIT ON COMPLETION.
                        ;
                        ;
                        ;
01DE 3620       DELT:   MVI     M,SPACE ; CLEAR WITH SPACE
01E0 2B                 DCX     H       ; DECREMENT THE CURSOR
01E1 C9                 RET             ; AND EXIT
                        ;
                        ;
                        IF      @SCROL  ; SCROLLING OPTION
                        ;
                        IF      @CNTRL  ; SCROLL-CONTROL
                        ;
```

```
                        ELSE
                        MOV     M,A     ; PLACE INPUT CHAR INTO DISPLAY MEMORY FROM
                        ENDIF                                                   ACC.
                        ;
                        ;
                        ;
                        ;
01A0 23                 INX     H       ; INCR. CURSOR
01A1 7C         VIDR:   MOV     A,H     ; CHECK IF SCROLL NEEDED
01A2 FED0               CPI     DPAGE   ; SEE IF HL TOO SMALL
01A4 DA3D02             JC      TOP     ; JUMP IF TOO SMALL
01A7 FED4               CPI     EPAGE   ; SEE IF HL TOO BIG
                        IF      @CNTRL
01A9 D41602             CNC     CONT    ; YES
                        ELSE
                        CNC     SCRL    ; PERFORM NORMAL SCROLL OR PAGE
                        ENDIF
                        ;
                VIDR2:
                        IF      @SOLID
                        MOV     A,M     ; GET CHAR UNDER CURSOR
                        STA     UNDR    ; SAVE IT
                        MVI     M,CS    ; WRITE CURSOR CHAR TO DISPLAY MEMORY
                        ELSE
                        SETB    7,M     ; MAKE RESERVE VIDEO CONDITION
01AC+CBFE
                        ENDIF
                        ;
                        ;
                        ;
01AE 222D03             SHLD    CURSOR  ; SAVE CURSOR ADDRESS
                        ;
01B1 F1                 POP     PSW
01B2 C9                 RET
                        ;
                ;*******************************************************
                ;*                                                     *
                ;*      SUPPORT ROUTINES.                              *
                ;*                                                     *
                ;*******************************************************
                        ;
01B3 7D         RTN:    MOV     A,L
01B4 E6C0               ANI     0C0H
01B6 6F                 MOV     L,A     ; CLEAR OUT CURRENT LINE COUNT
01B7 C9                 RET
                        ;
                        ;
                        ;
01B8 114000     LFEED:  LXI     D,64    ; DO CURS+= 64;
01BB 19                 DAD     D       ; ADD QUANT IN HL
01BC C9                 RET             ; AND EXIT.
                        ;
```

*(continued on top of page 26)*

```
                        ;
                        ; MLSCRL:       MULTI-LINE SCROLLING
                        ;
                        ;
01E2 3D         MLSCRL: DCR     A       ; IF A= 1, DO NORMAL SCROLL
01E3 322F03             STA     CNT     ; DO DUMMY STORE TO DO HOUSEKEEPING FOR
                                        ; SCRL.
                        JRZ     SCRL
01E6+2841
01E8 3C                 INR     A       ; OTHERWISE...
                        ;
01E9 C5                 PUSH    B       ; MUST PRESERVE INPUT CHAR HIDDEN IN B
01EA 6F                 MOV     L,A     ; SAVE NUMB. LINES TO SCRL
01EB AF                 XRA     A       ; ZERO ACC.
01EC 322F03             STA     CNT     ; UPDATE FOR AFTER SCROLLING
01EF 2600               MVI     H,0     ; GET # OF SCROLLS INTO HL
01F1 29                 DAD     H       ; HL = HL *2;
01F2 29                 DAD     H       ; *4;
01F3 29                 DAD     H       ; *8;
01F4 29                 DAD     H       ; *16;
01F5 29                 DAD     H       ; *32;
01F6 29                 DAD     H       ; *64;
01F7 EB                 XCHG
01F8 210004             LXI     H,MAXL*MAXW     ;
                        DSUB    D
01FB+B7
01FC+ED52
                        DMOV    B,H     ; MOVE VALUE TO CONT
01FE+44
01FF+4D
                        ;
0200 2100D0             LXI     H,DSPL  ; DEST ADDRESS
0203 EB                 XCHG            ; STORE DEST ADDRESS IN DE
0204 19                 DAD     D       ; FORM SOURCE ADDRESS
                        LDIR
0205+EDB0
0207 EB                 XCHG            ; HL IS DEST
0208 0E20               MVI     C,SPACE ; FILL CHAR
020A 71         MSCRL2: MOV     M,C     ; FILL
020B 23                 INX     H       ; POINT TO NEXT
020C 7C                 MOV     A,H
020D FED4               CPI     EPAGE
                        JRNZ    MSCRL2
020F+20F9
0211 21C0D3             LXI     H,DLINE
0214 C1                 POP     B
0215 C9                 RET
                        ;
                        ;
0216 3A2F03     CONT:   LDA     CNT     ; SCROLL COUNT
0219 FE10               CPI     16      ; MAX. LINES
```

*(continued on next page)*

# Z-80 Video Drivers

```
                        JRNC    ZERC    ; ZERO COUNT AND CLEAR SCREEN
021B+3005
021D 3C                 INR     A       ; BUMP CONT
021E 322F03             STA     CNT     ; SAVE IT
0221 C9                 RET
                        ;
0222 CDCA01     ZERC:   CALL    CLEAR           ; CLEAR SCREEN
0225 21C0D3             LXI     H,DLINE ; PROPER CURSOR LOC
0228 C9                 RET                     ; AND EXIT.
                        ;
                        ENDIF           ; END (IF SCROLL)
                        ;
                        ;
                        ;
                        ;
                        ;
                SCRL:
                        IF      @CNTRL
0229 C5                 PUSH    B               ; SAVE INPUT CHAR HIDDEN IN B
                        ENDIF
                        SOURCE  DSPL+64         ; SOURCE FOR MOVE
022A+2140D0
                        DEST    DSPL            ; PUT AT BEG. ON DISPLAY
022D+1100D0
                        COUNT   PCOUNT          ; SET MOVE COUNT
0230+01C003
                        LDIR
0233+EDB0
                        ELSE
                SCRL    EQU     HOME            ; PERFORM PAGING
                        ;
                        ENDIF
                        IF      @SCROL          ; IF SCROLLING IS ACTIVATED
                        ;
                        ;
                        ;
0235 CD5302             CALL    FILL
0238 21C0D3             LXI     H,DLINE
                        IF      @CNTRL
023B C1                 POP     B
                        ENDIF
023C C9                 RET                     ; EXIT
                        ENDIF
                        ;
                        ;
                        ;
                TOP:
                        ;
                        IF      @NEGSC  ; IF NEGATIVE SCROLL
023D CD4602             CALL    NSCRL
026B+EDA1
                        JRZ     ESCFND          ; LEGAL COMMAND
026D+2803
026F 19                 DAD     D               ; ADD REST OF OFFSET
                        JMPR    ESCLP           ; AND REPEAT
0270+18F5
0272 5E         ESCFND: MOV     E,M             ; GET LOW ORDER OF ADDR.
0273 23                 INX     H               ; POINT TO HIGH ORDER
0274 56                 MOV     D,M             ; GET HIGH ORDER OF ADDR.
0275 23                 INX     H               ; POINT TO COUNT BYTE
0276 7E                 MOV     A,M             ; GET IT
0277 323203             STA     ECHK            ; SAVE IT
027A B7                 ORA     A               ; SET FLAGS
                        JRZ     ESCGO           ; NO PARAMETERS SO EXEC NOW
027B+2806
                        SDED    EWORD           ; SAVE IT FOR LATER
027D+ED53
027F+3003
0281 E1         ESCOUT: POP     H               ; RECOVER CURSOR VALUE
0282 C9                 RET                     ; RETURN TO VIDR
                        ;
                        ;
0283 EB         ESCGO:  XCHG                    ; HL HAS EXEC. ADDR
0284 E3                 XTHL                    ; HL IS RESTORED, TOP HAS EXEC.
0285 C9                 RET                     ; GO THERE...                ADDR
                        ;
                        ;
0286 AF         ESCEOT: XRA     A               ; ZERO ACC.
0287 323203             STA     ECHK            ; TERM. ESCAPE SEQUENCE.
                        JMPR    ESCOUT          ; LEAVE...
028A+18F5
                        ;
                        ;
                        ;
                ;*******************************************************
                ;*                                                     *
                ;*      ESCAPE FUNCTIONS                               *
                ;*                                                     *
                ;*******************************************************
                        ;
                        ;
                        ;
                        ;
                ;*******************************************************
                ;*                                                     *
                ;*      XY ADDRESSING                                  *
                ;*                                                     *
                ;*      MOVES CURSOR TO THE ADDRESS (X,Y) SPECIFIED.   *
                ;*      IF THE VALUES ARE OUT OF BOUNDS, THE           *
                ;*      COMMAND ABORTS.                                *
                ;*                                                     *
                ;*******************************************************
```

```
                        ENDIF
                        ;
0240 2100D0             LXI     H,DSPL
0243 C3AC01             JMP     VIDR2           ; RESET CURSOR AND CONTINUE...
                        IF      @NEGSC          ; IF NEGATIVE SCROLL
                NSCRL:  SOURCE  DLINE-1
0246+21BFD3
                        DEST    ESPL-1
0249+11FFD3
                        COUNT   PCOUNT
024C+01C003
                CRL:    LDDR
024F+EDB8
                                                ; MOVE 'EM
                        JMPR    FILL1           ; SKIP OVER EXCHANGE
0251+1801
                        ;
                        ;
                        ENDIF
                        ;
                        IF      @NEGSC OR @SCROL
0253 EB         FILL:   XCHG                    ; GIVE HL FILL LOC.
0254 012040     FILL1:  LXI     B,4020H         ; FILL ONE LINE W/ SPACES.
0257 71         FILL2:  MOV     M,C             ; FILL IN.
0258 23                 INX     H               ; POINT TO NEXT ONE...
                        DJNZ    FILL2           ; ...AND REPEAT 'TILL DONE.
0259+10FC
025B 2100D0             LXI     H,DSPL          ; POINT TO FIRST LINE FOR NSCRL
025E C9                 RET                                                 FN.
                        ;
                        ENDIF
                        ENDIF
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        IF      @ESCMOD
                        ;
                        ;
025F 78         ESCSET: MOV     A,B             ; REVIVE INPUT CHAR
0260 E5                 PUSH    H               ; SAVE CURSOR ADDRESS
0261 210003             LXI     H,ESCTBL        ; ESCAPE COMMAND TABLE
0264 110300             LXI     D,3             ; OFFSET FOR ENTRIES MINUS ONE
0267 34         ESCLP:  INR     M
0268 35                 DCR     M               ; TEST FOR EOT.
                        JRZ     ESCEOT          ; ILLEGAL COMMAND...
0269+281B
                        CCI                     ; COMPARE AND ++HL;
```

*(continued on top of page 28)*

```
                ;*************************************************************
                        ;
                        ;
028C B7         XYADDR: ORA     A               ; SET FLAGS
028D 78                 MOV     A,B     ; REVIVE INPUT CHAR
                        JRZ     YADDR   ; DECODE Y
028E+2808
0290 FE40               CPI     64
                        JRNC    ECODE   ; ERROR OUT OF ADDRESSING MODE-->
0292+3019
0294 323303             STA     XAD     ; SAVE IT
0297 C9                 RET
                        ;
                        ;
0298 FE10       YADDR:  CPI     16
                        JRNC    ECODE   ; ERROR OUT OF ADDRESSING MODE-->
029A+3011
029C 6F                 MOV     L,A     ; FORM NEW ADDRESS IN HL:
029D 2600               MVI     H,0     ; ZERO OUT HIGH ORDER
029F 29                 DAD     H       ; HL:=64*HL;
02A0 29                 DAD     H
02A1 29                 DAD     H
02A2 29                 DAD     H
02A3 29                 DAD     H
02A4 29                 DAD     H
02A5 3A3303             LDA     XAD     ; GET X COMPONENT
02A8 5F                 MOV     E,A     ; PUT IN DE
02A9 16D0               MVI     D,DPAGE ; MAKE OFFSET FOR SCRN ADDRESS
02AB 19                 DAD     D       ; ADD THEM TOGETHER
02AC C9                 RET             ; RETURN WITH NEW CURSOR ADDRESS
                        ;
                        ;
                        ;
                ;*************************************************************
                ;*                                                           *
                ;*      ERROR EXIT ROUTINE FOR ESCAPE FUNCTIONS.             *
                ;*                                                           *
                ;*************************************************************
                        ;
                        ;
02AD AF         ECODE:  XRA     A       ; ZERO ACC.
02AE 323203             STA     ECHK    ; TERMINATE MODE
02B1 C9                 RET
                        ;
                        ;
                ;*************************************************************
                ;*                                                           *
                ;*      CLEAR LINE FUNCTION                                  *
                ;*                                                           *
                ;*************************************************************
                        ;
```

*(continued on next page)*

# Z-80 Video Drivers

```
02B2 CDB301     CLLINE: CALL    RTN             ; POSITION CURSOR
02B5 E5         FILLER: PUSH    H               ; SAVE CURSOR ADDRESS
02B6 CD5402             CALL    FILL1           ; FILL LINE PORTION
02B9 E1                 POP     H               ; RECOVER CURSOR VAL.
02BA C9                 RET
                        ;
                        ;
                ;*******************************************************
                ;*                                                     *
                ;*      DELETE TO END-OF-LINE FUNCTION                 *
                ;*                                                     *
                ;*******************************************************
                        ;
                        ;
02BB 7D         DEOL:   MOV     A,L             ; GET LOW HALF OF ADDRESS
02BC E63F               ANI     3FH             ; GET POS ON LINE.
02BE 47                 MOV     B,A             ; STUFF VAL. IN B
02BF 3E40               MVI     A,64            ; MAX. VALUE
02C1 90                 SUB     B               ; FIND ACTUAL COUNT
02C2 47                 MOV     B,A             ; PUT COUNT WHERE IT'S NEEDED
02C3 0E20               MVI     C,SPACE         ; FILL WITH SPACES
02C5 E5         FLLER2: PUSH    H               ; SAVE CURSOR ADDRESS
02C6 CD5702             CALL    FILL2           ; FILL LINE PORTION
02C9 E1                 POP     H               ; REINSTATE CURSOR ADDR
02CA C9                 RET                     ; AND EXIT...
                        ;
                        ;
                        ;
                        ;
                ;*******************************************************
                ;*                                                     *
                ;*      CLOSE LINE FUNCTION                            *
                ;*                                                     *
                ;*******************************************************
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
02CB CDB301     CLOLNE: CALL    RTN             ; HL TO BEGINNING OF LINE
02CE E5                 PUSH    H               ; SAVE VALUE
02CF EB                 XCHG                    ; DE:= CURS. ADDR
02D0 214000             LXI     H,64            ; CREATE SOURCE ADDRESS
02D3 19                 DAD     D               ; HL:= SOURCE
02D4 CDDF02             CALL    BMOVE           ; MOVE 'EM.
02D7 21C0D3             LXI     H,DLINE
02DA CD5402             CALL    FILL1           ; FILL BOTTOM LINE
```

```
                ;*******************************************************
                ;*                                                     *
                ;*      ESCAPE COMMAND TABLE                           *
                ;*                                                     *
                ;*******************************************************
                        ;
                        ;
ESCTBL: ESCOM   CHADR,XYADDR,2  ; XY ADDRESSING

0300+19
0301+8C02
0303+02

        IF      NOT @SECOND     ; THEN PUT SECOND LEVEL FNS |
        ESCOM   CHCLS,CLEAR,0   ; CLEAR SCREEN              |HERE

0304+06
0305+CA01
0307+00

        ESCOM   CHHME,HOME,0    ; HOME CURSOR

0308+05
0309+C601
030B+00

        ESCOM   CHLFT,LEFT,0    ; CURSOR LEFT

030C+0C
030D+C401
030F+00

        ESCOM   CHRGHT,RIGHT,0  ; CURSOR RIGHT

0310+12
0311+C201
0313+00

        ESCOM   CHUP,NLFEED,0   ; CURSOR UP

0314+15
0315+BD01
0317+00

        ENDIF
        ESCOM   CHEOS,DEOS,0    ; DELETE TO END OF SCREEN

0318+13
0319+CD01
031B+00

        ESCOM   CHEOL,DEOL,0    ; DELETE TO END OF LINE

031C+14
031D+BB02
031F+00

        ESCOM   CHCLOZ,CLOLNE,0 ; CLOSE LINE FUNCTION

0320+17
0321+CB02
0323+00

        ESCOM   CHOPEN,OPNLNE,0 ; OPEN LINE FUNCTION

0324+18
0325+E902
0327+00

        ESCOM   CHCLN,CLLINE,0  ; CLEAR LINE FUNCTION

0328+16
0329+B202
032B+00
```

```
02DD E1                 POP     H               ; RECOVER CURS POS.
02DE C9                 RET
                        ;
                        ;
                BMOVE:  LOAD
02DF+7E
02E0+12
02E1 23                 INX     H
02E2 13                 INX     D               ; POINT TO NEXT
02E3 7C                 MOV     A,H
02E4 FED4               CPI     EPAGE
                        JRNZ    BMOVE
02E6+20F7
02E8 C9                 RET
                        ;
                        ;
                        ;
                ;*************************************************************
                ;*                                                           *
                ;*      OPEN LINE FUNCTION                                   *
                ;*                                                           *
                ;*************************************************************
                        ;
                        ;
                        ;
                        ;
                        ;
02E9 CDB301     OPNLNE: CALL    RTN     ; HL:= START OF LINE.
02EC E5                 PUSH    H       ; SAVE VALUE
02ED EB                 XCHG            ; DE HAS ADDR
02EE 2100D4             LXI     H,ESPL  ; MAX. VALUE
02F1 E5                 PUSH    H       ; SAVE VALUE FOR LATER
                        DSBC    D       ; CREATE COUNT
02F2+ED52
                        DMOV    B,H     ; GET COUNT IN BC
02F4+44
02F5+4D
                        SOURCE  DSPL+959
02F6+21BFD3
02F9 D1                 POP     D       ; RECOVER VALUE USED BEFORE
02FA 1B                 DCX     D       ; DECREMENT IT
                        LDDR
02FB+EDB8
02FD E1                 POP     H
                        JMPR    FILLER
02FE+18B5
                        ;
                        ;
                        ;
                ;*************************************************************
                ;*                                                           *
                ;*      ESCAPE COMMAND TABLE                                 *
                ;*                                                           *
                ;*************************************************************
```

*(continued on top of page 30)*

```
032C 00                 DB      0               ; EOT
                        ;
                        ENDIF
                        ;
                        ;
                ;*************************************************************
                ;*                                                           *
                ;*      RAM AREA                                             *
                ;*                                                           *
                ;*************************************************************
                        ;
032D 00D0       CURSOR: DW      DSPL            ; POINT TO START OF SCRN
                        IF      @CNTRL
032F 00         CNT:    DB      0               ; SCROLL COUNT
                        ENDIF
                        IF      @SOLID          ; SOLID CURSOR CONDITION
                UNDR:   DB      0               ; CHAR HIDDEN UNDER CURSOR
                        ENDIF
                        ;
                        IF      @ESCMOD
0330 0000       EWORD:  DW      0
0332 00         ECHK:   DB      0
                        ;
0333 00         XAD:    DB      0               ; TEMP X ADDRESS VARIABLE
                        ENDIF
                        ;
                        ;
                        ;
                        ;
0334                    END
                        ******************
                        ***Symbol Table***
                        ******************
0000 BC         01D6 BFILL      01D7 BFILL2     0008 BK         02DF BMOVE
0019 CHADR      0016 CHCLN      0017 CHCLOZ     0006 CHCLS      0014 CHEOL
0013 CHEOS      001B CHESC      0005 CHHME      000C CHLFT      0018 CHOPEN
0012 CHRGHT     0015 CHUP       01D3 CL2        01CA CLEAR      02B2 CLLINE
02CB CLOLNE     032F CNT        0216 CONT       000D CR         024F CRL
032D CURSOR     0002 DE         007F DEL        01DE DELT       02BB DEOL
01CD DEOS       D3C0 DLINE      00D0 DPAGE      D000 DSPL       0332 ECHK
02AD ECODE      00D4 EPAGE      0286 ESCEOT     0272 ESCFND     0283 ESCGO
0267 ESCLP      0281 ESCOUT     025F ESCSET     0300 ESCTBL     D400 ESPL
0330 EWORD      0253 FILL       0254 FILL1      0257 FILL2      02B5 FILLER
02C5 FLLER2     00D8 FPAGE      0004 HL         01C6 HOME       0004 IX
0004 IY         01C4 LEFT       000A LF         01B8 LFEED      0100 MAIN
0010 MAXL       0040 MAXW       01E2 MLSCRL     020A MSCRL2     01BD NLFEED
019F NOCODE     017A NOESC      0246 NSCRL      0000 OFF        FFFF ON
02E9 OPNLNE     03C0 PCOUNT     01C2 RIGHT      01B3 RTN        0229 SCRL
0020 SPACE      023D TOP        015C VIDEO      01A1 VIDR       01AC VIDR2
0333 XAD        028C XYADDR     0298 YADDR      0222 ZERC       FFFF @CNTRL
FFFF @ESCMOD    FFFF @NEGSC     FFFF @SCROL     0000 @SECOND    0000 @SOLID
```

**end**

# Z-80 Video Drivers

## Example III

```
;*************************************************
;*                                               *
;*      CRT:                                     *
;*                                               *
;*      Driver for Life simulation program       *
;*      LIFEZ by A. Skjellum.                    *
;*                                               *
;*      Copyright 1980 (C) A. Skjellum.          *
;*                                               *
;*      All rights reserved.                     *
;*                                               *
;*      Written for use with TDL MACRO           *
;*      ASSEMBLER vs. 2.21.                      *
;*                                               *
;*************************************************
        ;
        ; Macros:
        ;
        .DEFINE JUSTFY =
        [ MOV   A,E
          ANI   0C0H
          MOV   E,A  ]
        ;
        ;
        ;
        ;
        .DEFINE FN[BYTE,WORD]   =
        [ .BYTE BYTE
          .WORD WORD   ]
        ;
        ;
        ;
        .DEFINE FORMAT[QUANT]   =
        [ LXI   H,QUANT
          DAD   D
          XCHG
          JMP   CRT4    ]
        ;
        ;
        ;
        ;
```

```
CRT:    PUSH    H
        PUSH    D
        PUSH    B
        PUSH    PSW             ; Save registers for sake of calling
                                ; routines.
        LXI     D,SCRN          ; Point to start of screen
        LHLD    CURS            ; Get cursor position.
        DAD     D               ; Make absolute address
        PUSH    PSW             ; Save input char
        LDA     UNDR            ; Get char under cursor
        MOV     M,A             ; Store it in video memory
        POP     PSW             ; Recover input char
        LXI     X,CRTBL         ; Point to command table
        LXI     B,CRTLEN        ; Count
CRT1:   CMP     0(X)            ; Do compare
        JRZ     CRT2            ; if true compare
        INX     X
        INX     X
        INX     X               ; IX+=3;
        DJNZ    CRT1            ; Check all entries
        JMPR    CRT3            ; and jump when done
        ;
CRT2:   PUSH    H               ; HL has valid cursor addr
        MOV     L,1(X)          ; Get address to execute
        MOV     H,2(X)
        XTHL                    ; Recover HL and put exec. addr
                                ; on the stack.
        RET
        ;
        ;
        ;
CRT3:   ORI     COLOR           ; add in bias
        MOV     M,A             ; put in video memory
        INX     D               ; increment cursor value
CRT4:   MOV     A,D             ; Check high order of curs. addr
                                ; for overflow
        CPI     4               ; is DE>1023?
        CNC     ONSCR           ; if so, then go
        LXI     H,SCRN          ; form absolute address
        DAD     D
        MOV     A,M             ; get char under cursor
        XRI     COLOR           ; flip color bit
        MOV     M,A             ; store it back
        SDED    CURS            ; update cursor
        POP     PSW
        POP     B
        POP     D
        POP     H
        RET
        ;
        ;
```

*(continued on top of page 32)*

```
CURUP:  FORMAT  UM
        ;
        ;
CURHO:  LXI     D,CENTR
        JMP     CRT4
        ;
        ;
CLEAR:  LXI     H,SCRN          ; Point to start of display
        MVI     M,SPACE         ; Fill char
..NXT:  MOV     M,B             ; fill byte
        INX     H               ; point to next one
        MOV     A,H             ; see if done:
        CPI     EPAGE           ; are we?
        JRNZ    ..NXT
        RET
        ;
        ;
        ;
        ;
        ;
;***********************************************
;*                                             *
;*      Service routines for CRT4              *
;*                                             *
;***********************************************
        ;
        ;
        ;
ONSCR:  LXI     D,960           ; point to bottom of display
        RET
        ;
;*********************************
;*          DATA AREA            *
;*********************************
        ;
        ;
        ;
CURS:   .WORD   0
UNDR:   .BYTE   SPACE
        ;
CRTBL:  FN       6,     FF
        FN      12,     CURLF
        FN      18,     CURRT
        FN      21,     CURUP
        FN      10,     LINE
        FN      13,     RETN
        FN       1,     CURDL
        FN       2,     CURDR
        FN       3,     CURUL
        FN       4,     CURUR
        FN       5,     CURHO
```

*(continued on next page)*

# Z-80 Video Drivers

```
        FN      11,     CCENI
        FN      14,     RRETN
        ;
        ;
        ; there must be no intervening code between the end of this table
        ; and the CRTLEN equate as that equate automatically defines the
        ; length of this table.
        ;
CRTLEN  =       (.-CRTBL)/3
        ;
        ;
        ;
;**********************************
;*                                *
;*          EQUATES               *
;*                                *
;**********************************
        ;
        ;
SCRN    =       0D000H          ;  address of screen memory

CENTR   =       543
UL      =       -65
UM      =       -64
UR      =       -63
ML      =        -1
MI      =         0
MR      =         1
LL      =       +63
LM      =       +64
LR      =       +65
SPACE   =       32+bias
STAR    =       '*'+bias
KILO    =       0400H
SPAGE   =       0B4H
COLOR   =       080H            ; reverse video bit seven

EPAGE   =       0D8H            ; for other displays epage = spage.
BIAS    =       0               ; bias necessary for correct display
                                ; 0 for flashwriter,but other displays
                                ; may need one.


        ;
        ;
        .END
```

**end**

```
        ora     a
        if      nul reg
        dsbc    h
        else
        dsbc    reg
        endif
        endm
        ;
        ;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+                                                            +
;+      dmov                                                  +
;+                                                            +
;+      Loads the contents of one sixteen bit                 +
;+      register with another.                                +
;+                                                            +
;+      b, d, h are valid arguments.                          +
;+                                                            +
;+                                                            +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ;
dmov    macro   reg1,reg2
        mov     reg1,reg2
        mov     reg1+1,reg2+1
        endm
        ;
        ;
        ;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+                                                            +
;+      Topld                                                 +
;+                                                            +
;+      Loads the spec. register with the contents            +
;+      of the top of stack.                                  +
;+                                                            +
;+                                                            +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ;
        ;
topld   macro   reg
        pop     reg
        push    reg
        endm
        ;
        ;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+                                                            +
;+      xchgb                                                 +
;+                                                            +
;+      Exchanges the contents of bc and hl (bc <-> hl)       +
;+                                                            +
```

# Listing I.

```
                        **********************
                        ******SUPPORT.LIB******
                        **********************



;*********************************************************
;*                                                       *
;*      Support library of macros and equates            *
;*                                                       *
;*      by A. Skjellum                                   *
;*                                                       *
;*      Contents organized in this library:              *
;*      July 22, 1980.                                   *
;*                                                       *
;*      Copyright 1980 (C) Anthony Skjellum.             *
;*      All rights reserved.                             *
;*                                                       *
;*********************************************************
        ;
        ;
        ;
;*********************************************************
;*                                                       *
;*      Macros                                           *
;*                                                       *
;*********************************************************
        ;
        ;
        ;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+                                                       +
;+      Dsub                                             +
;+                                                       +
;+      Does hl = hl - reg;                              +
;+                                                       +
;+      Reg defaults to hl.                              +
;+                                                       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ;
        ;
        ;
dsub    macro   reg
```

*(continued on top of page 34)*

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+                                                       +
;+      xchgb                                            +
;+                                                       +
;+      Exchanges the contents of bc and hl (bc <-> hl)  +
;+                                                       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ;
xchgb   macro
        push    b
        xthl
        pop     b
        endm
        ;
        ;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+                                                       +
;+      ixchg                                            +
;+                                                       +
;+      Exchanges the contents of x or y with the        +
;+      specified register pair.                         +
;+                                                       +
;+                                                       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ;
ixchg   macro   ireg,reg
        push    reg
        xti&ireg
        pop     reg
        endm
        ;
        ;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+                                                       +
;+      Source                                           +
;+      Dest                                             +
;+      Count                                            +
;+                                                       +
;+      These set up the registers hl,de and bc          +
;+      prior to block moves.  They were introduced      +
;+      to emphasize the nature of the block             +
;+      mode transfers in the code.                      +
;+                                                       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ;
source  macro   value
        lxi     h,value
        endm
dest    macro   value
        lxi     d,value
        endm
```

*(continued on next page)*

# Z-80 Video Drivers

```
count   macro   value
        lxi     b,value
        endm
        ;
        ;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+                                                          +
;+      Load                                                +
;+                                                          +
;+      Does (de):=(hl);                                    +
;+                                                          +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ;
load    macro
        mov     a,m
        stax    d
        endm
        ;
        ;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+                                                          +
;+      com                                                 +
;+                                                          +
;+      Used for making byte-word sequence tables.          +
;+                                                          +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ;
com     macro   byte,word
        db      byte
        dw      word
        endm
        ;
        ;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+                                                          +
;+      Escom                                               +
;+                                                          +
;+      Used to make byte-word-byte sequence tables.        +
;+                                                          +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ;
        ;
escom   macro   byte,word,attr
        com     byte,word
        db      attr
        endm
        ;
        ;
```

```
dspl    equ     0D000H          ; screen address
espl    equ     dspl+400H       ; first pos. of background display
pcount  equ     (maxl-1)*maxw   ; points to last line relative
dline   equ     dspl+pcount     ; commonly used address
        ;
dpage   equ     (dspl shr 8)    ; start page
epage   equ     dpage+4         ; start of background memory page
fpage   equ     dpage+8         ; page after background memory
        ;
        ;
;************************************************************
;*                                                          *
;*      Test macro                                          *
;*                                                          *
;************************************************************
        ;
test    macro
        local   temp,stor
        lxi     sp,temp
        lhld    1
        lxi     d,6
        dad     d
        shld    stor+1
        ;;
stor:   call    0
        res     7,a
        cpi     3
        jz      0
        call    video
        jmpr    stor
        ds      64
temp:
        endm
```

**end**

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+                                                             +
;+      jmpr                                                   +
;+                                                             +
;+      Z80 library expects "jr" command.  This makes          +
;+      both valid.                                            +
;+                                                             +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ;
jmpr    macro   ?addr
        jr      ?addr
        endm
        ;
        ;
        ;
        ;
;*************************************************************
;*                                                           *
;*      Equates                                              *
;*                                                           *
;*************************************************************
        ;
        ;
off     equ     0       ; logical off
on      equ     0FFFFH  ; logical on
        ;
        ;
maxw    equ     64      ; 64 character lines
maxl    equ     16      ; 16 lines on the display
        ;
        ;
        ;
space   equ     32      ; space char
cr      equ     13      ; carriage return
lf      equ     10      ; line feed
del     equ     7fh     ; del char
chcls   equ     6       ; clear screen code
chrght  equ     18      ; ctrl R char: curs. right
cheos   equ     19      ; ctrl S char: del. eos
cheol   equ     20      ; ctrl T char: del. eol
chup    equ     21      ; ctrl U char: curs up
chcln   equ     22      ; ctrl V char: clear line
chcloz  equ     23      ; ctrl W char: close line
chopen  equ     24      ; ctrl X char: line open
chadr   equ     25      ; ctrl Y char: cursor address
chlft   equ     12      ; cursor left char
bk      equ     8       ; backspace
chhme   equ     5       ; cursor home
chesc   equ     27      ; escape mode char
        ;
        ;
```

*(continued on top of page 36)*

# Z-80 File Comparator

## Listing

```
001000                          0100 ************************************************************
001000                          0110 *                                                          *
001000                          0120 *    'COMPA' -  A routine permitting the comparison        *
001000                          0130 *               of two files in RAM.    COMPA lists        *
001000                          0140 *               the  addresses  at  which the files        *
001000                          0150 *               contain  dissimilar  data  bytes...        *
001000                          0160 *                                                          *
001000                          0170 *    Command Format: Calls for AAAAAA-BBBBBB,CCCCCC        *
001000                          0180 *               AAAAAA -> Begin. addr. of file no.1        *
001000                          0190 *               BBBBBB -> Term.  addr. of file no.1        *
001000                          0200 *               CCCCCC -> Begin. addr. of file no.2        *
001000                          0210 *                                                          *
001000                          0220 *                                        8 - 10 - 80       *
001000                          0230 *                                                          *
001000                          0240 ************************************************************
001000                          0250 * OPERATING SYSTEM SUBROUTINES *
001000                          0260 OPRSYS EQU  340000     ; Operating system entry point.
001000                          0270 KEYBDO EQU  343162     ; Keyboard input.
001000                          0280 TV     EQU  343202     ; Outputs char. in A-reg to display.
001000                          0290 ERASE  EQU  343243     ; Clears display.
001000                          0300 LINBUF EQU  346335     ; Keyboard line buffer (in Phimon).
001000                          0310 TVEDT  EQU  343210     ; TV editor.
001000                          0320 DELAY  EQU  345201     ; Delays (A-reg)x(100) milliseconds.
001000                          0330 *
001000                          0340 * BEGIN  -  SET UP SCREEN, LINE BUFFER, AND DISPLAY
001000                          0350 *           START-UP MESSAGE...
001000                          0360 *
001000 041 337 002               0370 BEGIN  LD   HL,COUNT   ; Count-112D = no. of error listings
001003 066 160                  0380        LD   (HL),160   ;    on video display.
001005 315 243 343              0390        CALL ERASE      ; Clear video display.
001010 315 322 002              0400        CALL ROLLUP     ; Roll up paper if printer active.
001013 061 377 002              0410        LD   SP,STACK   ; Set stack pointer.
001016 041 372 001              0420        LD   HL,MESS0   ; Put start-up message on
001021 315 210 343              0430        CALL TVEDT      ;    video display only.
001024                          0440 *
001024                          0450 * TOGGLE  -  BACKSPACES CURSOR POSITION 18 SPACES ON VIDEO
001024                          0460 *            MONITOR IMMEDIATELY AFTER DISPLAYING MESSAGE
001024                          0470 *             'MESS0'...
001024                          0480 *
001024 006 024                  0490        LD   B,024      ; After command format is displayed,
001026 076 002                  0500 TOGGLE LD   A,2         ;    toggle cursor position back so
001030 315 202 343              0510        CALL TV         ;    that address entry overwrites
001033 020 371                  0520        DJNZ TOGGLE     ;    format string.
001035                          0530 *
001035                          0540 * KEYIN  -  ACCEPT COMMAND FROM KEYBOARD...
001035                          0550 *
001035 041 335 346              0560        LD   HL,LINBUF  ; Point to keyboard command buffer.
001040 315 162 343              0570 KEYIN  CALL KEYBDO     ; Enter command from keyboard.
001043 376 220                  0580        CP   220        ; CTL/P activates printer.
001045 050 033                  0590        JR   Z,PRTON    ;
001047 376 217                  0600        CP   217        ; CTL/O de-activates printer.
001051 050 046                  0610        JR   Z,PRTOFF   ;
001053 376 233                  0620        CP   233        ; Exit to Phimon on 'ESCAPE'.
001055 312 000 340              0630        JP   Z,OPRSYS
001060 167                      0640        LD   (HL),A     ; Place each keybd char. in buffer.
001061 376 215                  0650        CP   215        ; C/R signals end of command string.
001063 050 045                  0660        JR   Z,CONTIN   ; At end of command, process buffer.
001065 376 377                  0670        CP   377        ; On rub/out, erase command string
001067 050 307                  0680        JR   Z,BEGIN    ;    and begin again.
001071 043                      0690        INC  HL         ; Advance in buffer for next char.
001072 315 000 003              0700        CALL HCOPY      ; Print number if printer is active.
001075 315 202 343              0710        CALL TV         ; Output char. entered to display.
001100 030 336                  0720        JR   KEYIN      ; Accept another character entry.
001102 345                      0730 PRTON  PUSH HL         ; Place proper first instruction
001103 041 000 003              0740        LD   HL,HCOPY   ;    'PUSH AF' at beginning of
001106 066 365                  0750        LD   (HL),365   ;    print routine to make it
001110 315 322 002              0760        CALL ROLLUP     ;    functional.  Roll paper.
001113 315 322 002              0770        CALL ROLLUP     ;
001116 341                      0780        POP  HL         ;
001117 030 317                  0790        JR   KEYIN      ;
001121 345                      0800 PRTOFF PUSH HL         ; De-activate print routine by
001122 041 000 003              0810        LD   HL,HCOPY   ;    replacing first byte with
001125 066 311                  0820        LD   (HL),311   ;    a 'RETURN'.
```

*(continued on next page)*

# Z-80 File Comparator

## Listing

```
001127 341                      0830        POP  HL        ;
001130 030 306                  0840        JR   KEYIN     ;
001132                          0850 *
001132                          0860 * CONTIN  -  COMMAND COMPLETE.  NOW STRIP OFF THE
001132                          0870 *            REQUIRED ADDRESSES...
001132                          0880 *
001132 315 243 343              0890 CONTIN CALL ERASE     ; Clear command string from display.
001135 315 322 002              0900        CALL ROLLUP    ; Roll up paper if printer active.
001140 041 335 346              0910        LD   HL,LINBUF ; Locate beginning of command buffer.
001143 315 230 002              0920        CALL ASMADR    ; Take file#1 st. addr. from buffer.
001146 042 333 002              0930        LD   (LOWER),HL  Save file#1 st. addr. in 'LOWER'.
001151 041 335 346              0940        LD   HL,LINBUF ; Search buffer for '-'.  Address
001154 016 255                  0950        LD   C,'-'     ;    following is ending address
001156 315 217 002              0960        CALL LOADER    ;    for file#1.
001161 257                      0970        XOR  A         ; Clear carry flag for subtraction.
001162 355 113 333 002          0980        LD   BC,(LOWER)  Recall file#1 starting address.
001166 355 102                  0990        SBC  HL,BC     ; Calculate length of file#1; add
001170 043                      1000        INC  HL        ;    1 to include last byte in file.
001171 042 335 002              1010        LD   (STEPS),HL  Save file length in 'STEPS'.
001174 041 335 346              1020        LD   HL,LINBUF ; Search buffer for file#2 starting
001177 016 254                  1030        LD   C,','     ;   address following ','.
001201 315 217 002              1040        CALL LOADER    ;
001204 355 133 333 002          1050        LD   DE,(LOWER)  Point to first byte of file#1.
001210                          1060 *
001210                          1070 * COMPAR  -  COMPARE CORRESPONDING BYTES IN FILE#1
001210                          1080 *            AND FILE#2...
001210                          1090 *
001210 032                      1100 COMPAR LD   A,(DE)    ; Load A-reg with byte from file#1.
001211 106                      1110        LD   B,(HL)    ; Load B-reg with byte from file#2.
001212 220                      1120        SUB  B         ; Compare two bytes by subtraction.
001213 304 242 001              1130        CALL NZ,FAILS  ; Not identical - display address.
001216 345                      1140        PUSH HL        ; Save file#2 pointer and decrement
001217 052 335 002              1150        LD   HL,(STEPS)    the no. of comparisons remaining
001222 053                      1160        DEC  HL        ;   by 1.
001223 042 335 002              1170        LD   (STEPS),HL  Save remaining no. of comparisons.
001226 257                      1180        XOR  A         ; Check for zero remaining steps by
001227 274                      1190        CP   H         ;    comparing both H and L regs with
001230 040 001                  1200        JR   NZ,NOTZER ;    zero.
001232 275                      1210        CP   L         ;
001233 341                      1220 NOTZER POP  HL        ; Restore file#2 pointer in case
001234 050 066                  1230        JR   Z,FINISH  ;    there's more to be done.
001236 043                      1240        INC  HL        ; Advance file#2 pointer.
001237 023                      1250        INC  DE        ; Advance file#1 pointer.
001240 030 346                  1260        JR   COMPAR    ; Compare another pair of bytes.
001242                          1270 *
001242                          1280 * FAILS   -  BYTES NOT IDENTICAL.  OUTPUT FILE#1 ADDRESS
001242                          1290 *            OF DISCREPANCY IN OCTAL FORM...
001242                          1300 *
001242 325                      1310 FAILS  PUSH DE        ; Save file#1 pointer.
001243 103                      1320        LD   B,E       ; Save E-reg. (OCTAL modifies E-reg).
001244 172                      1330        LD   A,D       ; Convert high order part of address
001245 315 171 002              1340        CALL OCTAL     ;    to ascii digits for display.
001250 170                      1350        LD   A,B       ; Convert low order part of address
001251 315 171 002              1360        CALL OCTAL     ;    to ascii digits for display.
001254 076 002                  1370        LD   A,2       ; Output 2 spaces to display for
001242                          1300 *
001242 325                      1310 FAILS  PUSH DE        ; Save file#1 pointer.
001243 103                      1320        LD   B,E       ; Save E-reg. (OCTAL modifies E-reg).
001244 172                      1330        LD   A,D       ; Convert high order part of address
001245 315 171 002              1340        CALL OCTAL     ;    to ascii digits for display.
001250 170                      1350        LD   A,B       ; Convert low order part of address
001251 315 171 002              1360        CALL OCTAL     ;    to ascii digits for display.
001254 076 002                  1370        LD   A,2       ; Output 2 spaces to display for
001256 315 306 002              1380        CALL MLTSPC    ;   proper display formatting.
001261 321                      1390        POP  DE        ; Restore file#1 pointer.
001262 345                      1400        PUSH HL        ; Save file#2 pointer.
001263 041 337 002              1410        LD   HL,COUNT  ; Decrement no. of remaining display
001266 065                      1420        DEC  (HL)      ;    positions for error listings.
001267 050 016                  1430        JR   Z,HOLDIT  ;    Z-flag set - display is filled.
001271 076 007                  1440        LD   A,7       ; Mask off all but least sig. 3
001273 246                      1450        AND  (HL)      ;    bits of count. If these are 0
001274 314 301 001              1460        CALL Z,CARRET  ;    then line is full. Print C/R.
001277 341                      1470        POP  HL        ; Restore file#2 pointer.
```

```
356 344 240 345 362
362 357 362 256 256
256 240 322 305 255
305 316 324 305 322
241
002170 000                      2060        DB   0
002171                          2070 *
002171                          2080 * OCTAL   -  ROUTINE (TAKEN FROM DG Z-80 OP/SYS) WHICH
002171                          2090 *            CONVERTS 8-BIT BYTE INTO 3 ASCII DIGITS
002171                          2100 *            FOR OCTAL FORMATTED OUTPUT TO DISPLAY...
002171                          2110 *
002171 036 003                  2120 OCTAL  LD   E,3        ; Convert byte into 3 ascii digits.
002173 267                      2130        OR   A         ; Reset carry flag for rotation.
002174 027                      2140 SHIFT  RLA            ; Brings 3 bits into pos. for mask.
002175 027                      2150        RLA            ;
002176 027                      2160        RLA            ;
002177 365                      2170        PUSH AF        ; Save copy of A-reg.
002200 346 007                  2180        AND  7         ; Mask all but last 3 bits.
002202 366 260                  2190        OR   260       ; Convert to ascii equivalent.
002204 315 000 003              2200        CALL HCOPY     ; Print number if printer active.
002207 315 202 343              2210        CALL TV        ; Display ascii digit.
002212 361                      2220        POP  AF        ; Restore A-reg.
002213 035                      2230        DEC  E         ; Check whether all 3 digits have
002214 040 356                  2240        JR   NZ,SHIFT  ;    been formed - if not, continue.
002216 311                      2250        RET            ;
002217                          2260 *
002217                          2270 * LOADER  -  RETRIEVES THE ADDRESSES ENTERED INTO THE
002217                          2280 *            KEYBOARD (COMMAND) BUFFER...
002217                          2290 *
002217 176                      2300 LOADER LD   A,(HL)    ; Retrieve character from buffer.
002220 267                      2310        OR   A         ; A null (end-of buffer) before
002221 050 042                  2320        JR   Z,ERROR   ;    char. sought signals error.
002223 271                      2330        CP   C         ; Is this the char. we're seeking?
002224 043                      2340        INC  HL        ;
002225 302 217 002              2350        JP   NZ,LOADER ; If not, try again.
002230 315 243 002              2360 ASMADR CALL CONVRT    ; Take msb from two byte address in
002233 127                      2370        LD   D,A       ;    buffer and store in D-reg.
002234 043                      2380        INC  HL        ; Advance to second half of address.
002235 315 243 002              2390        CALL CONVRT    ; Take lsb from two byte address in
002240 137                      2400        LD   E,A       ;    buffer and store in E-reg.
002241 353                      2410        EX   DE,HL     ; Place two byte address in HL-pair.
002242 311                      2420        RET            ; Ret. address to comparison routine.
002243 006 003                  2430 CONVRT LD   B,3       ; Set counter for two passes through
002245 036 000                  2440        LD   E,0       ;    ASCII to binary byte forming
002247 257                      2450        XOR  A         ;    subroutine & clear carry flag.
002250 176                      2460 CONV01 LD   A,(HL)    ; Load an ASCII digit.
002251 346 007                  2470        AND  7         ; Mask all but least signif. 3 bits.
002253 263                      2480        OR   E         ; Combine part of binary byte already
002254 005                      2490        DEC  B         ;    formed with current ASCII digit.
002255 310                      2500        RET  Z         ; Rotation to place ASCII digits into
002256 027                      2510        RLA            ;    proper relative position in byte
002257 027                      2520        RLA            ;    only necessary for first two
002260 027                      2530        RLA            ;    digits of each 3 digit group.
002261 137                      2540        LD   E,A       ; Store portion of binary byte formed
002262 043                      2550        INC  HL        ;    in E-reg and pick up next digit.
002263 030 363                  2560        JR   CONV01    ;
002265                          2570 *
002265                          2580 * ERROR   -  SIGNAL COMMAND FORMAT ERROR...
002265                          2590 *
002265 315 243 343              2600 ERROR  CALL ERASE     ; Command format error.  Clear screen
002270 041 132 002              2610        LD   HL,MESS2  ;    and display error message for 3
002273 315 210 343              2620        CALL TVEDT     ;    seconds.
002276 076 036                  2630        LD   A,30D     ;
002300 315 201 345              2640        CALL DELAY     ;
002303 303 340 001              2650        JP   BUFZER    ; Clear command buffer and try again.
002306                          2660 *
002306                          2670 * MLTSPC  -  OUTPUTS MULTIPLE SPACES TO BOTH VIDEO DISPLAY
002306                          2680 *            AND TO PRINTER, IF ACTIVE...
002306                          2690 *
002306 107                      2700 MLTSPC LD   B,A       ; Generate no. of spaces held in
002307 076 240                  2710        LD   A,240     ;    A-reg. to both printer and
002311 315 000 003              2720        CALL HCOPY     ;    video display.
002314 315 202 343              2730        CALL TV        ;
002317 020 366                  2740        DJNZ MLTSPC+1  ;
002321 311                      2750        RET            ;
002322                          2760 *
002322                          2770 * ROLLUP  -  ROLLS UP PAPER ON PRINTER (IF ACTIVE)
002322                          2780 *            TWO LINES BY USING PRINT ROUTINE'S
002322                          2790 *            AUTOMATIC L/F WITH RECEIPT OF C/R...
002322                          2800 *
002322 076 215                  2810 ROLLUP LD   A,215     ; Roll up paper two lines if
```

```
001300 311                1480        RET             ; Display space remaining - continue.
001301 076 215            1490 CARRET LD   A,215      ; Print C/R to begin new printed line
001303 315 000 003        1500        CALL HCOPY      ;    on each 64th printed character.
001306 311                1510        RET             ;
001307                    1520 *
001307                    1530 * HOLDIT  -  FREEZE DISPLAY AND CHECK KEYBOARD FOR
001307                    1540 *            OPERATOR OPTION OF HALT/CONTINUE...
001307                    1550 *
001307 315 355 001        1560 HOLDIT CALL CHEKIN     ; Check keyboard option.
001312 066 160            1570        LD   (HL),160   ; Allow another 112 error listings.
001314 341                1580        POP  HL         ; Restore file#2 pointer.
001315 315 243 343        1590        CALL ERASE      ; Clear video display.
001320 315 322 002        1600        CALL ROLLUP     ; Roll up paper on printer
001323 311                1610        RET             ;    and continue.
001324                    1620 *
001324                    1630 * FINISH  -  COMPARISONS ARE COMPLETED.  PUT "TERM..."
001324                    1640 *            MESSAGE ON SCREEN AND CHECK OPERATOR
001324                    1650 *            OPTION.  EMPTY COMMAND BUFFER BEFORE
001324                    1660 *            CONTINUING...
001324                    1670 *
001324 041 122 002        1680 FINISH LD   HL,MESS1   ; Put "Term..." on screen to signal
001327 315 210 343        1690        CALL TVEDT      ;    end of comparisons.
001332 315 322 002        1700        CALL ROLLUP     ; Roll up paper if printer active.
001335 315 355 001        1710        CALL CHEKIN     ; Freeze display - check keyboard.
001340 041 335 346        1720 BUFZER LD   HL,LINBUF  ; Clean out command buffer by
001343 006 040            1730        LD   B,040      ;    filling it with 32 NOPs.
001345 257                1740        XOR  A          ;
001346 167                1750 FILZER LD   (HL),A     ;
001347 043                1760        INC  HL         ;
001350 020 374            1770        DJNZ FILZER     ;
001352 303 000 001        1780        JP   BEGIN      ; Repeat the entire process.
001355                    1790 *
001355                    1800 * CHEKIN  -  KEYBOARD CHECK FOR OPERATOR'S OPTION...
001355                    1810 *
001355 315 162 343        1820 CHEKIN CALL KEYBD0     ; Check keyboard for operator option.
001360 376 233            1830        CP   233        ; "Escape" exits back to Phimon.
001362 312 000 340        1840        JP   Z,OPRSYS
001365 376 240            1850        CP   240        ; "Space" allows process to continue.
001367 040 364            1860        JR   NZ,CHEKIN  ; Loop until "esc" or "space" chosen.
001371 311                1870        RET             ; Option chosen - continue process.
001372                    1880 *
001372                    1890 * MESSAGE AREA...
001372                    1900 *
001372 140                1910 MESS0  DB   140        ; Generates 96 spaces.
001373 140                1920        DB   140        ;       ... 96 spaces.
001374 100                1930        DB   100        ;       ... 64 spaces.
001375                    1940        DW   'Enter Lower File Address Range & Upper File
 305 356 364 345 362                                                          Starting
 240 314 357 367 345
 362 240 306 351 354
 345 240 301 344 344
 362 345 363 363 240
 322 341 356 347 345
 240 246 240 325 360
 360 345 362 240 306
 351 354 345 240 323
 364 341 362 364 351
 356 347 240 301 344
 344 362 345 363 363
 256 256
002073 130                1950        DB   130        ;       ... 88 spaces.
002074 100                1960        DB   100        ;       ... 64 spaces.
002075                    1970        DW   'AAAAAA-BBBBBB,CCCCCC'
 301 301 301 301 301
 301 255 302 302 302
 302 302 302 254 303
 303 303 303 303 303

002121 000                1980        DB   0
002122                    1990 MESS1  DW   'Term...'
 324 345 362 355 256
 256 256
002131 000                2000        DB   0          ;
002132 177                2010 MESS2  DB   177        ;       ...127 spaces.
002133 177                2020        DB   177        ;       ...127 spaces.
002134 177                2030        DB   177        ;       ...127 spaces.
002135 026                2040        DB   026        ;       ... 22 spaces.
002136                    2050        DW   'Command error... RE-ENTER!'
 303 357 355 355 341
```

*(continued on top of page 38)*

```
002324 315 000 003        2820        CALL HCOPY      ;    printer is active.
002327 315 000 003        2830        CALL HCOPY      ;
002332 311                2840        RET             ;
002333                    2850 *
002333                    2860 * DATA STORAGE AREA...
002333                    2870 *
002333                    2880 LOWER  DS   2          ; Starting address of file#1.
002335                    2890 STEPS  DS   2          ; No of remaining comparisons.
002337                    2900 COUNT  DS   1          ; 112D-no. of errors listed.
002340                    2910        DS   037        ; Stack area.
002377                    2920 STACK  DS   1          ; Top of stack.
003000 311                2930 HCOPY  DB   311        ; Beginning of hardcopy routine.
003001                    2940 ************************************************************
003001                    2950 *                                                          *
003001                    2960 *   The custom print routine  should replace lines       *
003001                    2970 *   2930  forward of this source code.  Registers        *
003001                    2980 *   should be "PUSHED" on entry and "POPPED" prior       *
003001                    2990 *   to returning to "COMPA". The first instruction       *
003001                    3000 *   of the print routine should be a "PUSH AF"....       *
003001                    3010 *   The character to be printed is carried  in the       *
003001                    3020 *   A-register.                                          *
003001                    3030 *                                                          *
003001                    3040 ************************************************************
```

# Symbol

```
OPRSYS 340000   KEYBD0 343162   TV     343202   ERASE  343243
LINBUF 346335   TVEDT  343210   DELAY  345201   BEGIN  001000
TOGGLE 001026   KEYIN  001040   PRTON  001102   PRTOFF 001121
CONTIN 001132   COMPAR 001210   NOTZER 001233   FAILS  001242
CARRET 001301   HOLDIT 001307   FINISH 001324   BUFZER 001340
FILZER 001346   CHEKIN 001355   MESS0  001372   MESS1  002122
MESS2  002132   OCTAL  002171   SHIFT  002174   LOADER 002217
ASMADR 002230   CONVRT 002243   CONV01 002250   ERROR  002265
MLTSPC 002306   ROLLUP 002322   LOWER  002333   STEPS  002335
COUNT  002337   STACK  002377   HCOPY  003000
```

# Subroutine

```
0100 * DELAY - TIME WASTER TO ALLOW ERROR MESSAGE TO BE
0110 *         DISPLAYED A FEW SECONDS BEFORE BEING ERASED.
0120 *
0130 LOOP   LD   B,200      ; Three nested loops.
0140 LOOP1  LD   C,320      ;
0150 LOOP2  DEC  C          ;
0160        JR   NZ,LOOP2   ;
0170        DJNZ LOOP1      ;
0180        DEC  A          ;
0190        JR   NZ,LOOP    ;
0200        RET             ; Back to calling routine.
```

# Subroutine

```
0100 * TV EDITOR ROUTINE - FROM DIGITAL GROUP'S PHIMON
0110 *                     OPERATING SYSTEM.
0120 *
0130 START  LD   A,(HL)     ; Load accum. with byte from string.
0140        OR   A          ; Check for a null, indicating
0150        RET  Z          ;    end of text.
0160        CP   200        ; If value<200 then generate
0170        CALL C,VIDSPC   ;    multiple spaces.
0180        OR   A          ; VIDSPC returns a null - Don't
0190        CALL NZ,TV      ;    print char. in this case.
0200        INC  HL         ; Advance pointer.
0210        JR   START      ; Try next character in string.
0220 VIDSPC LD   B,A        ; Video-only multispacer.
0230        LD   A,240      ; Generate a space.
0240        CALL TV         ;
0250        DJNZ VIDSPC+1   ; Output spaces until count=0.
0260        RET             ;
```

**end**

# Software Review

# 8080/Z-80 Editor, Assembler,

by J.N. Demas

*Copyright © 1980 by J. N. Demas, Chemistry Department, Univeristy of Virginia, Charlottesville, VA 22901.*

If you have had your North Star floppy disc system for a while, you are beginning to recognize that there are a lot of things which can be done either much faster or only in machine code. You may have tried entering machine language programs in hexadecimal through the monitor; this effort probably lasted only 20 or 30 minutes—unless you were truly desperate. Now that you are ready to select one of the many disc-based assemblers, the following thoughts should be kept in mind.

First, like good books and fine wine, no piece of software is suitable for everyone. Further, no piece of software will do exactly what you want, in exactly the way you want it done. I describe here what I consider to be a delightful disc-based assembly language software development package from XL Computer Products (P.O. Box 805, Mesa, AZ 85202). The package consists of an EDITOR, ASSEMBLER, and DISASSEMBLER. I describe the strengths, weaknesses, and overall operating characteristics of the system so that you can determine whether this package satisfies your needs.

The XL package is available in an 8080 and Z-80 based version. By software standards the cost is modest: $39.95 for the 8080 version and $59.95 for the Z-80 one. Updated versions such as the current multi-disc drive version can be purchased by owners of the earlier releases for the reasonable copying fee of $9.95 and $14.95, respectively. In view of its performance, I rate the XL package as an excellent buy. Indeed, I am adopting it to use in teaching the software portion of an interfacing course for chemists.

The XL package is not a general purpose DOS; it is oriented only toward assembly language program development. The EDITOR is for creating or modifying assembly language source files; the ASSEMBLER is for generating machine executable code files, and the DISASSEMBLER converts raw machine code into assemblable assembly language files. The power of the system is shown by the following example. Using a single density North Star disc drive (~ 84K/disc) and a 24K RAM system, I disassembled a 9K machine code program to generate 80K of linked assembly language source disk files. I then edited the source files and reassembled the entire 9K object code program in a single pass. This could have been done somewhat more tediously using only 16K of RAM. With one double density disc system, an object code program of about 20K could be disassembled, edited, and reassembled in a single pass. Machine language programs this large are more than adequate for all but the most dedicated (or inefficient) programmers.

If you need, or like, the incredible range of commands of CP/M with the ability to submit chains of commands, to change file names, copy one file into another, etc., then the XL package may not be the one for you. An exception to this may be if you find that you do not have a large enough disc system to handle your programs in the context of a CP/M operating system.

## Overview

The EDITOR, ASSEMBLER, and DISASSEMBLER are all transient programs that overlay with no more than one resident in RAM at any one time. The EDITOR uses meaningful four character mnemonics for commands in the tradition of Processor Technology (indeed, most are the same) and is trivial to learn to use. The programs are all very chatty, giving the operator good cues rather than cryptic ones. The entire package is easy to learn to use if you know assembly language; you should be using the system adequately in one or two hours and proficiently in a day. Most of my problems arose because I just scanned rather than read through the manual.

All disc and input/output operations are performed through the North Star DOS so that if you have a functional North Star DOS, there are no bothersome patches to make to bring the XL programs up. Also, if you have a single density or double density disc controller the same software will work with both. The XL package is supplied in single density format, which you can easily convert to double density if you wish. The EDITOR, which also functions a monitor, is used for creating files and saving them on disc and for fetching and editing existing disc files.

An interesting and unadvertised feature of the Z-80 package is that it is written entirely in 8080 machine code. This means that the XL programs will run on either a Z-80 or an 8080 developmental system. While this feature may incense the purist, I am delighted with it since I have a great deal of money in 8080A based SOL and MITS computers and want to develop software for Z-80 single board systems without having to set up a complete Z-80 developmental system. The XL package permits me to do this without a major new expenditure. The only penalties are that the XL software has somewhat larger memory requirements and executes more slowly than if it had been done in optimized Z-80 code. I have talked with Robert Walden at XL and both versions will also run on a Z-80 system.

The EDITOR created source files can be no larger than the RAM space following the EDITOR, which is ~17K for a 24K system starting at 2000H (2000H/7FFFH). The ASSEMBLER is a standard two pass assembler that operates on disc source files rather than RAM stored files. The ASSEMBLER works on the disc file piecemeal, and it never has more than a small portion of the source in RAM at any one time. Thus, the only RAM required during assembly is for the DOS, the ASSEMBLER, 1280 bytes of disc buffer, and the complete symbol table. In a 16K system this limits you to about 1000 symbols—a number which I find inconceivably large—my disassembled 9K object code file yielded only about 600 labels. Programs that are longer than the RAM file space may be assembled by linking several source files together at assembly time; I will return to this feature later.

From raw object code the DISASSEMBLER generates either a simple assembly language listing or assembly language source disc files. Unlike many popular but inferior disassemblers, the XL automatically generates labels for all referenced lines. The generated assembly language files are thus immediately ready for reassembly. As with other disassemblers this one generates improper code on ASCII strings and jump tables, so these do have to be picked over manually.

## EDITOR

The EDITOR is a combination monitor- and line-oriented editor; line numbers are used to maintain proper order, and to list or modify lines. The EDITOR commands are listed in Table I. In case you forget the commands, COMS lists all

# Disassembler

valid ones. The DUMP and ENTR are standard monitor commands for displaying memory contents in hexadecimal or for entering data into memory. Arguments may be in hexadecimal (hex), octal, or decimal. Hex numbers must be followed by an H, octal by a Q, and decimal by nothing or a D. Since my SOL monitor assumes hex arguments with no suffix H, I am always inadvertently entering decimal values.

**FIGURE 1.**

Commands recognized by the XL EDITOR. These are printed out on typing COMS.

| | | | |
|---|---|---|---|
| ENTR | DUMP | JUMP | FILE |
| LOAD | SAVE | ASEQ | RSEQ |
| LIST | SCAN | DELT | APND |
| EXIT | MOVE | CONV | ASSM |

An important feature of this XL program, which should be emulated by others, is that you cannot enter data into the ASSEMBLER, the DOS, or the file area. Programmers, especially new ones, find this to be a serious error source and this simple human-designed feature will save innumerable hours of retyping a destroyed file.

The FILE command is used to name the current file (only one can be in memory at a time) or to display the file parameters (start and end RAM addresses, the number of disc blocks required to store it, and the last line number). The command LOAD name1, will load the named file and make it the current file; this overwrites any file in memory. SAVE and SAVE name2, save the current file on disc either under the current name or name2. If a file with the same name exists, it is overwritten. If no file of that name is available then a new one is created with that name.

The command APND name2 appends the file name2 to the end of the current file and updates the file information. If you have misfigured and the composite file consisting of name1 plus name2 exceeds the memory available to the EDITOR, a memory overflow error message is issued and the original file is kept intact. Thus an APND can almost never lose an existing file. The APNDed file is always placed at the end even if its current line numbers would imply embedding it in the middle of the current file. The line numbers are just updated, if required, to reflect their current position. The only time I have destroyed a file in RAM is when I APNDed a file that was so long it overwrote my MONITOR's scratchpad RAM, which is at high memory. Any reasonable attention paid to file and RAM size would avoid this problem.

RSEQ arg1, arg2 renumbers the entire file from the beginning. Arg1 is the desired starting number. Arg2 is optional and is the step size. ASEQ arg1, arg2 is the autoline number sequence that automatically inserts line numbers while entering programs. Arg1 is the start value and arg2 is the step size. By use of RSEQ and ASEQ new typed material is easily added to the interior of an existing file. Unlike the Processor Technology assembler, line numbers are not limited to 9999 but can be as large as 65535 (16 bits).

CONV arg1, arg2 takes the difference of arg1 and arg2 and displays the result in hex, decimal, and octal. If arg2 is omitted, it is treated as 0. This function is invaluable for calculating addresses for the assembler or just as a hex, octal, decimal calculator. For someone with as poor an arithmetic sense as I have, this function has avoided countless programming errors.

The most delightful feature of the EDITOR, however, is SCAN. This is the search and search/replace command. For example SCAN"NAME" would list all lines on which the string NAME occurred. SCAN "LABEL" "NEWLABL" will search for and replace every occurrence of LABEL with NEWLABL and print out every line as it appears after it has been changed. The old and new label may be shorter, longer, or the same length as the old label. SCAN is invaluable for replacing disassembly "L" labels with meaningful ones and for modifying labels in subroutines so that label conflicts are eliminated when building up programs from a set of libraries.

The line editor uses most of the same commands as the North Star BASIC line editor. Therefore, for those proficient with the North Star, editing will be easy. DELT arg1, arg2 is the standard line delete command. MOVE arg1, arg2 copies a 256 byte block of memory from address arg2 to address arg1. Again you cannot overwrite or damage your DOS, EDITOR, or file with a MOVE command as the EDITOR senses this impending disaster and refuses to do it.

ASSM or ASSM file2 loads the ASSEMBLER and begins assembly of either the currently named file or of file2. All assembly is done on disc files, and the current file in RAM is destroyed. The EDITOR, however, protects you from loading a file, spending an hour editing and then destroying it by an ASSM. If you change the file, the EDITOR will not permit assembling it until you SAVE it.

## ASSEMBLER

The ASSEMBLER can be entered either as a GO program using the DOS command or from the EDITOR. If entered from the DOS, the ASSEMBLER asks you the file name to be assembled. The ASSEMBLER is chatty and asks very specifically for what it wants. If you read the cues, you are hard pressed to err. Assembly time options include generating: (1) object code disc file, (2) complete assembled listing, (3) assembly error listing only, and (4) object code stored in memory. The object code stored in memory is not very useful since it is stored in the ASSEMBLER file area and this has, in general, no relationship to the actual assembly location.

Being exclusively a disc-based system, the disc activity is nearly continuous during assembly. For those whose only experience is with assemblers that work entirely on RAM files such as ALS8, the enormous reduction in speed necessitated by the disc activity will be a surprise. A 16K source file (about 2K of object code) without comments, requires about 90 seconds to assemble. This appears to be comparable to, or faster than, the CP/M assembler.

One of the most significant features of this assembler is the inclusion of the new pseudo op LNK "name." This is the link disc files command. When encountered in a file during assembly, the pseudo op LNK NEXTFIL instructs the assembler to continue the assembly process on the next disc file (NEXTFIL in this case). Any number of disc files may be linked or chained together by inserting the LNK command at the end of each piece. Forward and backward references to labels in other disc files are all properly handled. The only limitation is that all files must be on accessible disc space at any one time; in a multidisc drive system, the link can be specified to a file on any of the available drives. Irritating, but not fatal, operator errors are to forget that any source code in a file following the LNK is ignored and LNKing a chain of files back on itself – the snake devouring itself problem.

Another unusual feature of the ASSEMBLER is that symbolic labels can be up to 10 characters long and are unique to 10 characters. Thus, ANOTHERNM1

and ANOTHERNM2 are different. With 10 characters, meaningful labels can profitably be used to make a program largely self-documenting.

The ASSEMBLER has the equivalent of an ALS8 ASC statement, which permits embedding ASCII text strings in the program. This is done merely by putting quotes around the string. For example, "PLEASE PUSH BUTTON #1" will embed the enclosed 20 character ASCII string in the machine code. Comments are, of course, permitted in the source listings.

Multiple statements per source line are permitted. For example:

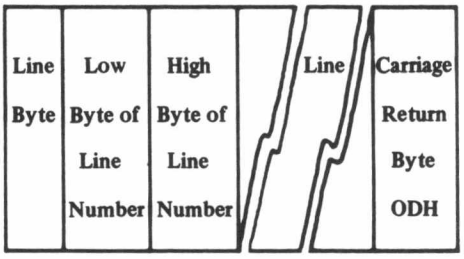
```
1000  LOOP INRA/INX H/ MOV
          M/ JNZ LOOP
```

would replace four lines of source code in many other assemblers. The "/" functions as a delimiter between op codes. The only limitation is that no labels are permitted except at the beginning of a line. Thus,

```
1000  INR A / LOOP MOV M,A /
          INX H / JNZ LOOP
```

is not permissible, because the assembler tries to interpret LOOP as an op code rather than a label.

The file structure is interesting. It is similar to, but more efficient than, Processor Technology's Software Package 1 or ALS8. In the XL files, the form is:




The end of file marker is a 01, and the line byte counter is the total number of bytes in the line including the counter and the CR just as in ALS8. The XL line number is always stored as a two-byte pure binary number and is converted to decimal when printed. In ALS8 the line number is always a four digit ASCII number (9999 maximum line number) stored in four bytes. Thus, the XL format saves two bytes per line over ALS8 and permits lines numbered to 65535. The XL format of multiple statements per line adds further memory savings since one byte counter, one line number, and a CR serve in several statements. Thus, substantially more program can be stored in XL format than in ALS8 format (perhaps 20-50% more). The XL format is close enough to ALS8 format, however, that conversion between the two is straight forward. Thus, if you have reams of ALS8 or Software Package I programs, conversion to XL format is easy (I will describe such a program elsewhere).

## DISASSEMBLER

A disassembler is an essential element of any software development system. In many cases machine language software suppliers do not give adequate information to modify their software to accommodate your own special needs such as I/O drivers. In other cases, as has unfortunately happened to me, the single assembly language source has been lost and only the functioning machine language code is available. In both cases, a disassembler is invaluable to properly modify or recover the software. A word of warning is in order, however. The newcomer who thinks that he is going to go out and promptly tear down and correct or modify someone else's undocumented machine code is in for a rude awakening. For all but the most trivial programs, determining how someone else's software works and modifying it is a long and tedious process – although if done on good software, it is an excellent way to learn good programming.

The DISASSEMBLER disassembles type 1 GO disc files. The GO address is treated as the origin. Simple disassembly with listing and/or generation of a source disc file are permitted. Figure 2 is a printed portion of an original source and a sample disassembly of this program. The lines "ASCII >> . . . <<" are the ASCII equivalent of the above block of code; this ASCII listing is not put into the disc file (I rather wish that it had been since it would be easy to edit out), but is useful for locating jump tables and messages. A careful study of a listing reveals the interesting trade-offs that the authors made. Overall I think that they made excellent choices, but intelligent study of the listing is always essential.

First, note the problems that the DISASSEMBLER has with the ASCII string; it generates functionally meaningless code that, if reassembled at the same origin, would regenerate the original machine code. If the program were relocated results could be disastrous although not in the current case. Note also that all jumps and calls are given symbolic labels even though the reference addresses are outside the disassembled region. An attempt to reassemble this source yields undefined label errors for the call to the monitor routine in line 1 and 28; these are easily corrected using an EQU statement. The advantage of this procedure is that it aids program relocation at a new address. All internal labels are handled properly. Further, note that the LXI H, . . .'s end up with a symbolic label on the loaded word since the 16 bit addresses loaded into HL are within the disassembled region. In the sample shown this is correct since HL is used as a memory pointer; you want HL to be loaded using a symbolic address in case you want to relocate the program. In many cases, however, double register pairs are loaded with pure numbers. In those cases when an "L" label is assigned (e.g., using an LXI D,0 to zero D and E), reassembly at another region of memory would reassign the numeric value – with disastrous results to the program function. It is thus essential to go through and determine by following program function which LXI rp, LXXXX value should have the "L" removed. The SCAN function searching for the string ",L" will turn up all of these quickly and efficiently.

Another fine feature is the way the disassembler works when the generated source files are too big to fit in the disc file set aside for it. After filling the first disc file, the disassembler stops and asks the name of the next disc file to be filled during disassembly. After you give it the new name "file2," the disassembler inserts at the end of the first disc file the necessary LNK file2 line and continues the disassembly on to disc file "file2." This process is repeated until the disassembly is complete. Thus, even when more than one source file is generated by a disassembly, the files are already linked for the assembly process. In the 8080 version you must precreate these files using the DOS. The Z-80 version does it right and will create the files automatically.

### System Disadvantages

The disc-based nature is a speed disadvantage. In disassembly about 5 minutes are required for a 9K object code file that yields about 70-80K of source code. In assembly about 4-5 minutes are required for four LNKed source files of about 70K total length to yield 9K of object code.

Although the North Star DOS supports up to 8 I/O devices the XL assem-

bler recognizes only two: the console and the serial printer. These can be changed manually during listing by the operator who can watch his video terminal, and change over to the printer in regions of interest. The two I/O device restriction makes writing custom input or output drivers harder.

Since the XL software uses the North Star DOS, all the limitations of the North Star DOS are carried over to the XL software. The DOS can only append new files to the free space at the end of the disc even if several big holes exist in lower disc space. Thus, you could run out of room on a nearly empty disc. This space can be regained by using the North Star compact utility, but this destroys the current run.

The SCAN and replace is strictly a local rather than global function. Only the file in RAM is affected. For changes that span linked files, it is necessary to read in and change each piece. A batch SUBMIT function as in CP/M would be useful for circumventing this problem.

The formatting of source listing is nonexistent—exactly the way you type it in. The assembled listing is better with proper indentation.

Short of printing an entire assembly, there is no quick way to go back and look at a piece of the assembly. Unlike CP/M the XL software cannot generate a print disc file of the assembly process for later inspection.

*(continued on next page)*

## FIGURE 2.

**Facsimile of printed portion of an original source showing how it will be assembled (A) and the disassembly (B) of the result and machine code. In the disassembly, addresses 37H to 3CH are not part of the actual program and would be ignored.**

**(A)**

```
0000 CD 1F C0    10 INLOOP CALL SYSTEMINP
0003 CA 00 00    11        JZ INLOOP
0006 FE 45       12        CPI "E"
0008 CA 04 C0    13        JZ ENDOFRUN
000B FE 4D       14        CPI "M"
000D C2 19 00    15        JNZ WHAT
0010 21 28 00    16        LXI H,OKMESSAGE
0013 CD 2F 00    17        CALL MESSOUT
0016 C3 00 00    18        JMP INLOOP
0019 21 22 00    19 WHAT LXI H,WHATMESSAG
001C CD 2F 00    20        CALL MESSOUT
001F C3 00 00    21        JMP INLOOP
0022 57 48 41    22 WHATMESSAG "WHAT?"
0027 00          22        DB 0
0028 41 4E 20    23 OKMESSAGE "AN M! !"
002E 00          23        DB 0
002F 7E          24 MESSOUT MOV A,M
0030 AF          24        XRA A
0031 C8          24        RZ
0032 47          25        MOV B,A
0033 CD 19 C0    26        CALL SYSOUTPUT
0036 C9          27        RET
0037             28 SYSTEMINP EQU 0C01FH
0037             29 SYSOUTPUT EQU 0C019H
0037             30 ENDOFRUN EQU 0C004H
```

**(B)**

```
0000 CD 1F C0     1 L0000   CALL   LC01F
0003 CA 00 00     2         JZ     L0000
0006 FE 45        3         CPI    45H
0008 CA 04 C0     4         JZ     LC004
000B FE 4D        5         CPI    4DH
000D C2 19 00     6         JNZ    L0019
0010 21 28 00     7         LXI    H,L0028
0013 CD 2F 00     8         CALL   L002F
0016 C3 00 00     9         JMP    L0000
0019 21 22 00    10 L0019   LXI    H,L0022
001C CD 2F 00    11         CALL   L002F
  ASCII >>    E    M    !<    /    !"    / <<
001F C3 00 00    12         JMP    L0000
0022 57          13 L0022   MOV    D,A
0023 48          14         MOV    C,B
0024 41          15         MOV    B,C
0025 54          16         MOV    D,H
0026 3F          17         CMC
0027 00          18         NOP
0028 41          19 L0028   MOV    B,C
0029 4E          20         MOV    C,M
002A 20          21         DB     20H
002B 4D          22         MOV    C,L
002C 21 21 00    23         LXI    H,L0021
002F 7E          24 L002F   MOV    A,M
0030 AF          25         XRA    A
0031 C8          26         RZ
0032 47          27         MOV    B,A
0033 CD 19 C0    28         CALL   LC019
0036 C9          29         RET
0037 00          30         NOP
0038 7E          31         MOV    A,M
0039 AF          32         XRA    A
003A C8          33         RZ
003B 47          34         MOV    B,A
003C CD 19 C0    35         CALL   LC019
  ASCII >>    WHAT? AN M! !    G    G    <<
003F C9          36         RET
```

The ASSEMBLER supports addition and subtraction, but not multiplication, division, byte and word shifting, nor upper-lower byte conversions of an address. Particularly in large, frequently relocated programs these features are very bothersome, although not insurmountable, problems. Conditional assembly is also not supported.

The chatty nature of the ASSEMBLER and DISASSEMBLER is convenient and useful with a video display. If you use a printer, however, the time and paper wasted printing the instructions every time would become bothersome. I see no obvious way to defeat this feature.

Finally, after using the ALS8 and the ELECTRIC PENCIL editors (superb memory-mapped video ones), I find the strictly line-oriented North Star editor cumbersome and slow. I have now written programs to interconvert ALS8, XL, and ELECTRIC PENCIL I files, but more on that elsewhere. Also, the XL package is not designed as a stand alone package. None of the system's utilities, such as in ALS8, are available for the user nor can custom commands be set up. Great flexibility in printout is lacking also; for example, there is not "list on" and "list off" commands that can be embedded in a program for unattended printing out of segments of the program during assembly. A simulator or debugging package is not included and would be a quite valuable addition; I use the ALS8 simulator for this stage of development.

### Summary

Overall I find the XL package powerful, easy to use and versatile. I wish I'd had it two years ago; it would have saved countless hours. I am currently converting all my ALS8 programs over to XL files. The APND and SCAN and replace make building up and using a software library trivial since perhaps 80% of a machine language project can consist of routines you have had to write before. If you have these routines as XL disc files, you can just APND them together to build the final program – after using the SCAN and replace if common labels had occurred between modules. For really big programs where the entire source program cannot fit in RAM at once, the LNK command can be used. While this approach is not as powerful as an assembler that generates relocatable load modules, and uses a linking loader to build the final program, it is easier to learn to use and will probably satisfy all but the most avid programmers.

Although not described by XL, another valuable use of the SCAN and replace is a pseudo-macro capability. For example during the writing of a program, you might want to save by PUSHing and then later restore by POPing in the inverse order all of the registers. A common programming error is to restore them in a different order; the hours spent tracking down this trivial error are countless. To simplify this task every time you want to save all registers you type in the line SAVEALL and every time you want to restore them you type in the line RESTORALL. Then before you assemble just type in the two lines:

```
SCAN "SAVEALL" "PUSH H/PUSH
    D/PUSH B/PUSH PSW"
SCAN "RESTORALL" "POP
    PSW/POP B/POP D/POP H"
```

Then every occurrence is correctly changed.

Finally, I will conclude with a few examples of how I am using the XL package. We are currently using SOL-20's and, while the ROM based monitor is excellent, many of the built-in functions are not used by us. Processor Technology supplied the documented source printout. I have disassembled the monitor, replaced all of the "L" labels with the Processor Technology ones, and we are now throwing out the parts we do not want and adding features that we need. When we are finished we will burn a new set of PROM's that do just what we want. In another area, my copy of the ELECTRIC PENCIL I has a number of small but irksome bugs. I have disassembled it and am slowly locating and correcting them as well as adding new features. We also have Processor Technology's powerful, compact PROMable Software Package I assembler object code and the 40 page source listing. Rather than try to type in the source listing, I have disassembled the object code and we are changing the labels with SCAN to agree with the original source. Then we will add new functions, relocate it in memory so that it does not interfere with any of our other system software, and PROM it so that we can have a permanent resident assembler in non-disc based systems. Try some of these projects on limited money, limited memory, limited disc space or limited time. I think that you will then agree that the XL package is indeed a real bargain which will pay for itself almost immediately. ■■

# Club News

## Eastern Iowa Computer Club

The EICC has been active by reaching out to the local community. They recently offered hands-on North Star experience, IMSAI synthesized music and club information on SWTP. Their March meeting is scheduled to be on the subject of displays. For information, call club president Sam Dillon (319) 377-0889.

## Portland Computer Society

PCS is steadily progressing towards incorporation as a non-profit corporation. Contributing members ($25 per year) help them to pool and spread public information about microcomputers. Subscribers to the PCS newsletter ($10 per year) receive the latest club news and a full calendar of events. PCS may be contacted at P.O. Box 17371, Portland, OR 97217.

## Northwest PET Users Group

The new address for the NWPUG is c/o Richard Ball, 2565 Dexter N., #203, Seattle, WA 98109. They now meet on the second Tuesday of each month, 7:30 p.m. at the University of Washington's Academic Computer Center, 3737 Brooklyn, Seattle, WA.

## New England Computer Society

Like many local clubs, NECS offers special interest groups and "microsessions." Some of their users groups focus on CP/M, 6502, TRS-80, 6800, PET, APPLE and Z-80. They have about 200 paid members who sponsor semi-annual swap-shops, computer and electronic flea markets. For information, write to NECS, P.O. Box 198, Bedford, MA 01730. ■■

## Donating Subscribers

**Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives five one-year subscriptions).

**Contributing Subscribers:** Robert L. Edens, Eric P. Hanson, J. G. Zeratsky, Mervin E. Frank, Edward W. Carr, Steven Fisher, Thomas H. Grohne, Richard Washburn, Les Hancock, G. V. Elkins, C. A. Forsberg, Daniel R. Lunsford, J. D. Griffith, Michael Slade, Thomas M. Schaefges, General Electric, Louis L. Odette, Robert M. Connors, William J. McEown, Robert C. Luckey, Kim Leary, Robert N. Stabler, Dewitt S. Brown, John B. Palmer.

**Retaining Subscribers:** Mike Faithful and Osamu Shimizu.

**Sustaining Subscribers:** Gordon Eubanks, Jr. and Robert C. Birss, Ph.D.

# Book Reviews

**A PRIMER ON STRUCTURED DESIGN**
**By Richardson, Butler, Tomlinson**
**Published by PBI-Petrocelli**
**232 pages, $17.50**
**Reviewed by Charles K. Ballinger**

Anyone interested in the "what is it?, why?, and how?" of structured program design should read this book. Having just completed a week-long structured design class in July, I can only say that far more would have been assimilated if this book had been the text for the course.

The three authors are to be commended for their writing clarity and ease of getting the subject matter across to the reader without the introduction of several hundred new buzzwords. While there is some new terminology, it is set forth in an easy-to-understand manner, and a complete glossary of terms is provided as backup.

The book covers structured design from its early background through the evolution of various design methodologies from top-down design to current design techniques, while maintaining an excellent mixture of text and examples to reinforce the subject matter. One area not many home users may use, but none the less very important to the professional programmer, is the information in Chapter 6 dealing with the use and philosophy of Pseudo-code. Although the examples are in FORTRAN, COBOL and PL/1, any language can be substituted to some degree, even BASIC!

Even if structured design does not yet strike your fancy, Chapter 7 on program style and debugging makes the book an excellent source for the budding programmer or the seasoned veteran. I do, however, take exception to the comment on page 143, "Readability is more important than efficiency. (10% of the codes typically consumes 50% or more of the execution time.)" I say more like 10% of the code executes 90% of the time, and certain trade-offs must be made between efficiency and readability.

Each chapter has a summary and review questions along with the notes and references used, so further study or reference material may be obtained if a particular topic is of special interest to the reader.

Two additional chapters of great interest are Chapter 8 on Database Considerations, and Chapter 9 on Managing the Programming Process. Since these two areas have significant impact on program design, it is most useful to see how a database environment affects program design, and why managers inherently get nervous when both this week and last week you reported your program to be 90% complete. Knowing both sides of the fence makes one keenly aware why you do or don't want to stay on your side of it!

In summary, this book covers not only structured program design, but all the facets involved in the "structured" concept. Even if you are not yet into structured anything, you should consider it on its own merit, since industry appears to be heading down the structured road to new horizons. This book is an excellent reference guide, a valuable learning tool, and is well worth the price. It should occupy a prominent spot in your library, to be read and used wherever possible.

**THE CP/M HANDBOOK WITH MP/M**
**By Rodney Zaks**
**Published by Sybex**
**321 pages, $13.95 paperbound**
**Reviewed by George Reeves**

CP/M has become the *de facto* standard operating system for a substantial portion of the microcomputer systems in existence today – those systems utilizing the 8080 and its successors. Rodney Zaks, in this handbook, intends to teach the user how to use the CP/M system and its resources. No prior knowledge of computers is assumed (though it is assumed that the reader has access to a computer system equipped with CP/M). This reviewer, in reading the handbook, had the knowledge but not the CP/M system. While it would be nice to have, it is not necessary for a good understanding of CP/M.

The first chapter is an introduction to CP/M and MP/M. It gives basic definitions of computer systems and compares CP/M to some of its look-alikes, such as Cromemco's CDOS. This chapter shows the reader how to bring up CP/M and to use its built-in transient commands.

Chapter 2 goes into much more detail about the CP/M commands and the control characters recognized by the system.

The PIP command is briefly introduced in Chapter 1, and Chapter 3 details its capabilities, such as how to concatenate files, how to print a group of files with just one command, and how to read a file up to a specific word, without using the assembler.

The CP/M editor, ED, is explained in Chapter 4. It tells in general terms how an editor operates, what it can do, and, in specific terms, how this relates to ED.

For system programmers, Chapter 5 describes the internal operation of CP/M. You will want to read the chapter if you wish to know how an operating system works, or if you intend to modify or use some of CP/M's routines.

Chapter 6 is a good reference guide to CP/M and MP/M commands and programs. It specifies which commands are available in the two versions of CP/M and version 1.0 of MP/M, and how each operates.

Chapter 7 is a compilation of practical hints in using CP/M or MP/M, and Chapter 8 is a short discussion of their future.

The last 30 pages of the book consist of 15 appendices intended for reference use. Some are helpful (the reference guide to the commands of ED, PIP, and CP/M); others are duplications of what you probably have elsewhere (hexadecimal and ASCII conversion tables); and others are useful depending upon your experience (checklist of supplies).

There are the usual number of typographical errors, but these are minor and fairly evident. There is a point of confusion as to what will happen if, in copying files from disk A to disk B, there is already a file of the same name on B: on page 37 it says the copy operation will stop, while on page 38 it says the file will be deleted. Such a condition would be easy to check out.

For those who intend to read this handbook from cover to cover, it is helpful, though repetitious in places; but for those users of CP/M 1.4 or 2.2 and MP/M, it is an excellent reference guide and I heartily recommend it. ■■

## Good News!

*Dr. Dobb's Journal* is now published twelve times each year! Subscribers will still receive the number of issues originally requested. Watch your mailing label for the new expiration date.

# Intext

## *An Input Routine for Text Editors*


## by Greg Smith

*Greg Smith, Commonwealth Department of Transportation, Ground Facilities Division, P.O. Box 367, Canberra 2601, Australia.*


Anyone who has attempted to write a text editor in North Star BASIC is almost certain to have come across the problem of generating a Line Length Error when inputting text. The following passage shows how I overcame this problem using North Star BASIC Release 4.

The problem begins with the maximum line length allowed in North Star BASIC (normally 132 characters). If more than 132 characters are input from the terminal in succession, without being interrupted by a carriage return, a Line Length Error is generated and the program will crash. A typical input sequence which will cause a Line Length Error is given in Listing 1.

This error can now be avoided by utilizing the INCHAR$ function, available in Release 4 BASIC. This function waits for a character to be input from the terminal and then returns that character as a single character string. In effect INCHAR$ is the same as INPUT except:

1) Only one character is returned.
2) No carriage return is necessary.
3) The character is not "echoed" to the terminal.

Therefore it can be seen that INCHAR$ overcomes the problem of Line Length Error, since only one character at a time is input. This is not to say that our problems are over, for INCHAR$ generates some headaches of its own:

1) Since the input character is not echoed to the terminal you cannot see what you are typing.

2) As only one character is input at a time, the BASIC Line Editor cannot be used to backspace over typing errors.

3) Any character typed on the terminal (except Control C) will be accepted as a legitimate character. This may cause problems if a non-printing character is accidently typed and added to the text string, as it may disrupt printout.

The first problem can be overcome simply by printing the single character string into which the INCHAR$ function has placed the input character (Listing 2).

The problem of editing the text as it is input is slightly more difficult to overcome and involves detecting a backspace character before it is entered into the string. This can be done by utilizing the ASC function. This function returns the ASCII value of the designated string. Therefore, if the ASCII value of each character is examined as it is input, then a backspace character can be intercepted before it is added to the text string. It is then a simple matter to reduce the length of the text string by one character for each backspace received from the terminal. At the same time it is possible to print a backspace on the terminal. Since some terminals will not accept a backspace, the user may select the character he wishes to print for a backspace and insert the ASCII value for this character in line 50 (Listing 3).

There is one minor problem which arises when a backspace is printed. The operating system treats a backspace as a "normal" character and does not reduce the character count for the line. Consequently, the operating system provides a carriage return before the line is complete, i.e., before the line is "full." The only real problem with this is that it looks untidy. The solution is to provide a character counter and line feed/carriage return within the routine. This mechansim has been incorporated in INTEXT. The number of characters allowed per line may be varied to suit your requirements by changing the value of B in line 20 (Listing 3).

Examining the ASCII values of the input characters is also the solution of the third problem outlined above. Since the ASCII values of the "printing" characters lie between 32 and 126 inclusive, it is a simple matter to eliminate spurious characters before they reach the text string. INTEXT also detects carriage returns and will print them at the terminal but will not add them to the text string. If two carriage returns are performed in succession, the routine terminates. The resulting text is contained in the string A$.

INTEXT is not a particularly complex routine (refer to flowchart Figure 1 on page 48). For this reason it is relatively fast, and consequently the time it takes from when a character is input until the next character can be input is quite short. The program has sufficient speed to keep pace with a normal typing rate.

### Additional Feature

When you incorporate INTEXT into your text editor, you may wish to utilize another capability it possesses. As pre-

```
LOAD LIST1
READY
LIST

10 REM , , , , , LISTING 1. . . .
20 REM IF MORE THAN 132 CHARACTERS ARE
30 REM INPUT IN SUCCESSION, A LINE LENGTH
40 REM ERROR WILL OCCUR IN LINE 80
50 LINE 132
60 DIM A$ (2000)
70 A$=""
80 INPUT "",T$
90 A$=A$+T$
100 IF LEN(A$)>=2000-25 THEN 120
110 GOTO 80
120 !"STRING FULL"
130 END
READY
```

```
LOAD LIST2
READY
LIST


10 REM . . . . . LISTING 2. . . . .
20 LINE 64
30 DIM A$ (2000)
40 A$=""
50 T$=INCHAR$(0)
60 !T$
70 A$=A$+T$
80 IF LEN(A$)=2000 THEN 100
90 GOTO 50
100 !"STRING FULL"
110 END
READY
```

sented in Listing 3, INTEXT makes use of only two of the ASCII characters below character 32, i.e., the "non-printing" characters, namely character 8 (backspace) and character 13 (carriage return). This means there are another 29 characters that can mean something to your text editor. For example, the version of INTEXT that I have customized for my text editor recognizes Control P (character 16) as a paragraph marker.

**Conclusion**

INTEXT is a fast-acting text input routine in North Star BASIC Release 4. It provides a means of inputting a continuous text into a string without generating a Line Length Error. The number of characters per line can be easily varied to suit the user's terminal. Carriage returns are automatic at the completion of a line and backspacing is allowed.

The routine has the potential to accept control characters and to perform assigned functions on receipt of those characters. ■■

```
LOAD LIST3
READY
LIST


2 REM LISTING 3 INTEXT. . . . . .
3 REM G. L. SMITH. . . . . .
4 REM 4 WILLICOTT STREET . . . . . .
5 REM GEEBUNG . . . . . .
6 REM BRISBANE, 4034 . . . . . .
7 REM QUEENSLAND. . . . . .
8 REM AUSTRALIA . . . . . .
10 A=0\REM. . . . . A = CARRIAGE RETURN INDICATOR
20 B=64\REM . . . . . B = NUMBER OF CHARACTERS PER LINE (MAX. 131)
30 C=500\REM . . . . . C = STRING DIMENSION
40 D=0\REM. . . . . D = LINE LENGTH INDICATOR
50 E=8\REM. . . . . E = ASCII VALUE FOR BACKSPACE CHARACTER
55 LINE 132
60 DIM A$ (C)
70 A$=""
80 REM . . . . . .
90 REM                  IF CHARACTER 8 NOT USED FOR BACKSPACE
100 REM                 THEN CHANGE LINE 150 TO READ D=D+1
110 REM . . . . . .
120 T$=INCHAR$(0)\REM . . . . . COLLECT CHARACTER FROM TERMINAL
130 IF ASC(T$)<>8 THEN 180
140 IF D<>1 THEN A$=A$(1,LEN(A$)-1) ELSE A$=""
150 D=D-1
160 !CHR$(E),\REM . . . . . PRINT BACKSPACE CHARACTER
170 GOTO 120
180 IF ASC(T$)<32 THEN 190 ELSE 200
190 IF ASC(T$)<>13 THEN 120
200 REM
210 IF (A=1) AND (ASC(T$)=13) THEN 330 ELSE A=0
220 !T$,\REM. . . . . "ECHO" INPUT CHARACTER TO TERMINAL
230 D=D+1
240 IF D=B THEN !CHR$(13) ELSE 260
250 D=0
260 IF ASC(T$)=13 THEN !CHR$(10), ELSE 290\REM. . . . . PERFORM LINE FEED
270 A=1
280 GOTO 120
290 A$=A$+T$\REM . . . . . ADD NEW CHARACTER TO EXISTING STRING
300 REM . . . . . CHECK FOR FULL STRING
310 IF LEN(A$)>=C-25 THEN !"YOU HAVE ",C-LEN(A$)," CHARACTERS REMAINING"
320 GOTO 120
330 END
READY
```

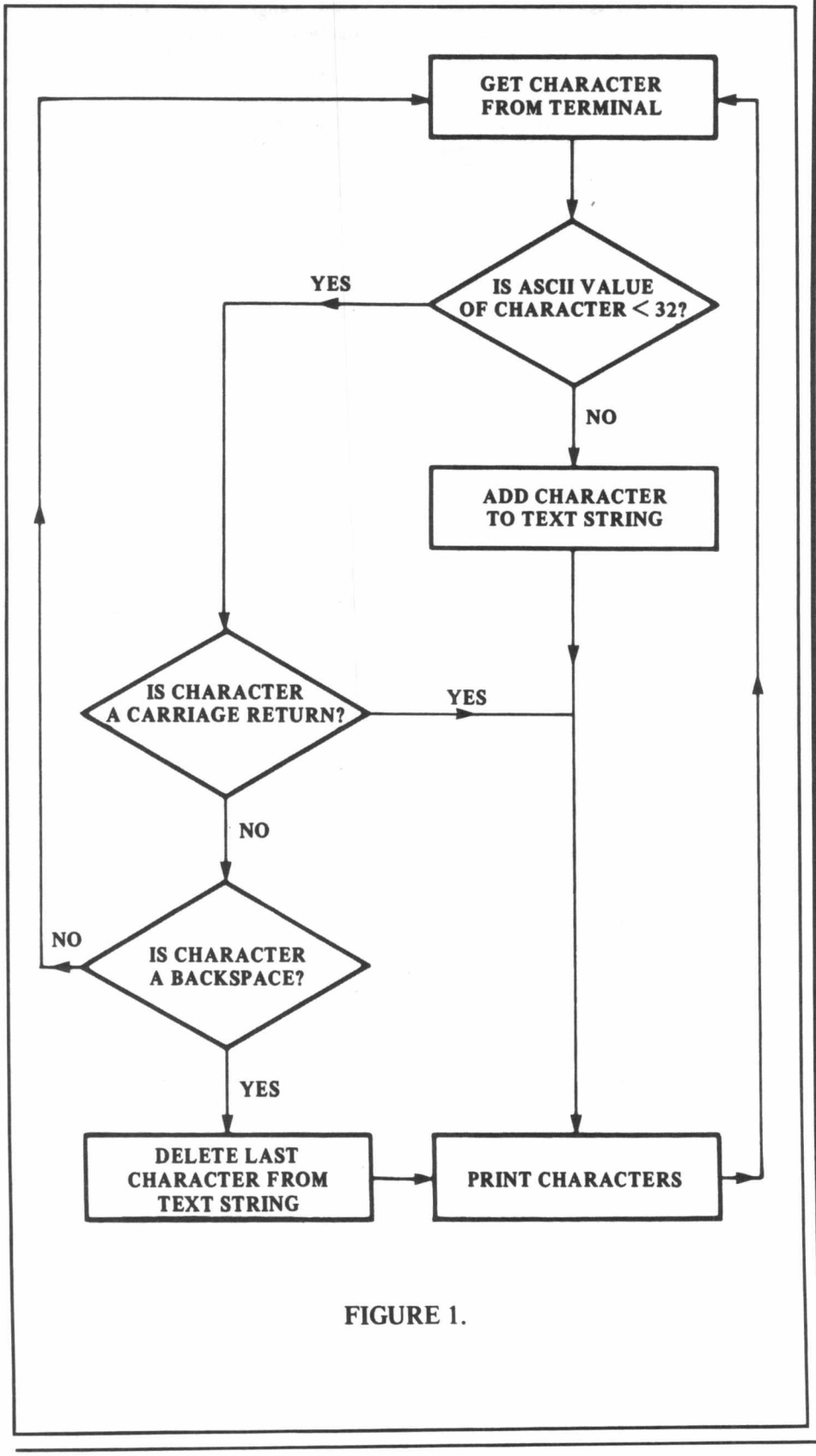



FIGURE 1.

*(continued from page 7)*

are confident that the Hazeltine 1500 series, with a predicted M.T.B.F. of 8192 hours and a demonstrated reliability of more than double that prediction, are reliable when compatible with a system. Unfortunately, this is no consolation to Dr. Callas because we do not conclude that the inability to perform on Dr. Callas' system is a product defect. We can and have offered our services by trying to evaluate the cause. This has been made somewhat difficult by the fact that Dr. Callas has a unique environment.

We would also like to state that we have sold thousands of units through the Computerland stores and thousands more through our distributors. On the whole, they all do an excellent job. The percentage of problems is miniscule. They take full advantage of Hazeltine's services and have never hesitated to enlist the aid of our total resources.

Our last communication with Dr. Callas' attorney was in August. We promised to continue trying to solve the problem with the limited information supplied. He promised not to do anything until a final determination was reached. We were quite surprised to find that Dr. Callas had written to your magazine.

Thank you again for the opportunity to reply.

Yours truly,
Edward Jeran
Director of Sales
Hazeltine Corporation
Industrial Products Division
Greenlawn, NY 11740
October 24, 1980

*Editor's note: Mark Garetz of* InfoWorld *reported the comment "Hazeltine terminals seem to work longer if you don't unpack them." Whether the circumstances which led to that remark are related to Dr. Callas' problem, we do not know. If any* DDJ *readers are aware of similar difficulties, please send us a letter with details.* ■■

# Turing Machine Simulation

## *An Introduction to a Universal Language*


## by J.L. Lawrence

*J. L. Lawrence, Mathematics & Computer Science Department, Western Kentucky University, Bowling Green, Kentucky 42101.*


There is a tendency among programmers, especially beginners, to want to learn only one or two programming languages and force all problems to fit into the framework provided by the language used. I suppose this attitude is justifiable in part if the programmer constantly works on only one class of problems such as business or scientific problems. With the advent of microcomputers, BASIC has become sort of a standard for small machines and users may be left wondering why software vendors are beginning to promote languages such as Pascal, FORTRAN, and FORTH. This article is an attempt to illustrate one reason for this desire for multilingual micros and to provide a very brief introduction to the purpose and study of programming languages.

Someone once stated, "If a computer can do it, I can write it in FORTRAN." This statement, stated in different words, was actually discussed by mathematicians and logicians before computers were ever invented. The research done in the early 1930's produced some very interesting results. Several computational models were proposed and all were shown to be equivalent. These proposed models are interesting only from a theoretical point of view but they do show that very little hardware is actually required to build a computer and that the languages currently in use (i.e., FORTRAN, COBOL, LISP, etc.) are universal languages. What this means is that a program written in one language can also be written in any other language, including the computational models proposed in the 1930's! Hence, any program that can be written for a computer can be written in FORTRAN (or BASIC, or COBOL, etc.) but it may be painful or costly to do so. Painful programming will be illustrated below. One reason for the proliferation of programming languages and the development of new languages is an attempt to match specific languages to specific classes of problems. FORTRAN, for example, was designed to be a fast executing language for scientific applications where computations tend to be rather lengthy and very little I/O is performed. COBOL was designed for business applications with quite a bit of I/O but very little "number crunching." BASIC was designed to be easy to teach and learn.

Another reason for continued research is to improve existing languages by extending their features. BASIC is an excellent case in point. Other research has produced general purpose languages, such as PL/I, but usually at a cost of requiring more core and time for compilation. Sacrifices and tradeoffs in design goals are often necessary to achieve general purpose languages.

One of the early computational models proposed is the Turing machine. A Turing machine consists of an infinite length tape and a read/write head which moves over a stationary tape. In operation, the read/write head can perform only three operations: (1) read the character under the tape head and move the head right or left, (2) read the character under the head and write a new character, and (3) terminate. Keep in mind that the Turing machine is universal so that any program that can be written for a computer can be done in only these three steps! Input to the machine consists of a finite string of symbols (from some alphabet) written on the tape and output is a finite string written on the tape.

In moving the tape head right and left, if either end of the finite string is reached, a blank is automatically written at the end. Figure 1 is a program written in BASIC-PLUS on a PDP 11/45 to simulate a Turing machine, but for obvious implementation reasons, the tape is 81 characters wide. If the left end of the tape is ever reached, an error occurs.

Perhaps the best way to visualize the Turing machine concept is to visualize it as a set of rules for passing from one state to another. That is, a program is to be a set of quadruples of the form $Q_0XYQ_1$ where $Q_0$ is the state the machine is currently in (0 is the initial state), X represents the character under the tape head, Y represents the action to take, and $Q_1$ represents the next state for the machine. If Y is an R or an L, the tape is moved in the proper direction. If Y is anything except R or L, X is replaced by the character Y. Execution terminates when an undefined state is reached. Beginning with the initial state, the Turing machine takes the action specified by the quadruple corresponding to the current machine state and passes into the state specified until an undefined state is reached.

As an example, assume we have a string with no imbedded blanks which contain only lowercase *a*'s and *b*'s and we wish to translate them to uppercase. The approach to take is obvious. Starting from the left side of the input string, read a character, translate it, and move the read head to the right. Since we are assuming no imbedded blanks, only two possibilities exist for characters in the input string, namely *a* and *b*. Thus, only two instructions are necessary to translate the tape: 0aA0 and 0bB0. Next, we must move the read head by 0AR0 and 0BR0. Lastly, we can stop with 0␣␣1 since a blank will be at the end of the input string. Figure 1 shows the entire program and traces its execution for the string *ababba*. It is important to realize that each state must consider every possible character that could be under the tape head in that state or else an error occurs. The program (see Listing) does not concern itself with this error but may fall into the next defined state with unpredictable results. If example 2 could have any of the 26 letters of the alphabet, an additional 48 instructions would be required for the program!

Figures 2 and 3 are other examples of this "painful" programming. The program in Figure 2 takes an input string of 1's and 0's and replaces all 1's with #'s. Figure 3 illustrates a more practical problem in that it determines whether there are an even or and odd number of 1's in the input string. Furthermore, it illustrates the difficulty in producing text output. The point, again, is not that anyone would actually do useful work with a Turing machine but rather that it can be done. ■■

**(LISTING ON PAGE 50)**

# Turing Machine Simulation

## FIGURE 1.

```
                        TURING MACHINE

                          COMMANDS
                          --------

                         GO      STOP
                         LIST    NEW
                         DELETE  INSERT
                         TRACE   NOTRACE
                         WRITE   READ
                         ENTER   HELP

              'WRITE' AND 'READ' REFER TO DISK WHILE
              'ENTER' REFERS TO THE KEYBOARD.


COMMAND: ? ENTER


YOU MAY NOW ENTER UP TO 100 INSTRUCTIONS FOR THE TURING MACHINE.
PROGRAM BEGINS EXECUTION IN THE INITIAL STATE OF 0.

WHEN YOU ARE FINISHED ENTERING INSTRUCTIONS, TYPE A RETURN.  REMEMBER
THAT 'R' AND 'L' CANNOT BE PART OF YOUR ALPHABET SINCE THEY DESIGNATE DIRECTIONS
FOR THE TAPE HEAD TO MOVE.

# 1   ? 0aA0
# 2   ? 0AR0
# 3   ? 0bB0
# 4   ? 0BR0
# 5   ? 0  1
# 6   ?
 5  INSTRUCTIONS ENTERED


COMMAND: ? TRACE


COMMAND: ? GO
INPUT TAPE: ? ababba
INITIAL TAPE HEAD POSITION: ? 1

INITIAL TAPE
ababba
TAPE: Ababba   HEAD POSITION:  1   INSTRUCTION EXECUTED:  1
TAPE: Ababba   HEAD POSITION:  2   INSTRUCTION EXECUTED:  2
TAPE: ABabba   HEAD POSITION:  2   INSTRUCTION EXECUTED:  3
TAPE: ABabba   HEAD POSITION:  3   INSTRUCTION EXECUTED:  4
TAPE: ABAbba   HEAD POSITION:  3   INSTRUCTION EXECUTED:  1
TAPE: ABAbba   HEAD POSITION:  4   INSTRUCTION EXECUTED:  2
TAPE: ABABba   HEAD POSITION:  4   INSTRUCTION EXECUTED:  3
TAPE: ABABba   HEAD POSITION:  5   INSTRUCTION EXECUTED:  4
TAPE: ABABBa   HEAD POSITION:  5   INSTRUCTION EXECUTED:  3
TAPE: ABABBa   HEAD POSITION:  6   INSTRUCTION EXECUTED:  4
TAPE: ABABBA   HEAD POSITION:  6   INSTRUCTION EXECUTED:  1
TAPE: ABABBA   HEAD POSITION:  7   INSTRUCTION EXECUTED:  2
TAPE: ABABBA   HEAD POSITION:  7   INSTRUCTION EXECUTED:  5
FINAL TAPE: ABABBA
FINAL HEAD POSITION:  7
```

## Listing

```
10  ! ************************************************************************************
20  ! *  TURING -- SIMULATION OF A TURING MACHINE                                        *
30  ! *                                                                                  *
40  ! *  AUTHOR -- J. L. LAWRENCE       8/8/80                                          *
50  ! *                                                                                  *
60  ! *  MAIN VARIABLES: S--INSTRUCTION TO EXECUTE (PTR. INTO C$)                        *
70  ! *                  S$-STATE OF MACHINE                                             *
80  ! *                  I$-INPUT/OUTPUT TAPE                                            *
90  ! *                  C$-ARRAY OF USER INSTRUCTIONS (M X 4)                           *
100 ! *                  M--NUMBER OF INSTRUCTIONS                                       *
110 ! *                  P--TAPE HEAD POSITION                                           *
120 ! *                  L--LENGTH OF TAPE (I$)                                          *
130 ! *                  F--TRACE ON/OFF FLAG                                            *
140 ! ************************************************************************************
150 DIM I$(81),C$(100,4)
155 M=0
160 PRINT:PRINT:PRINT TAB(29);"TURING MACHINE"
170 PRINT:PRINT TAB(32);"COMMANDS":PRINT TAB(32);"--------":PRINT
180 PRINT TAB(30);"GO";TAB(37);"STOP"
190 PRINT TAB(30);"LIST";TAB(37);"NEW"
200 PRINT TAB(30);"DELETE";TAB(37);"INSERT"
210 PRINT TAB(30);"TRACE";TAB(37);"NOTRACE"
211 PRINT TAB(30);"WRITE";TAB(37);"READ"
215 PRINT TAB(30);"ENTER";TAB(37);"HELP"
216 PRINT:PRINT TAB(15);"'WRITE' AND 'READ' REFER TO DISK WHILE"
217 PRINT TAB(15);"'ENTER' REFERS TO THE KEYBOARD."
220 PRINT:PRINT:A$="":PRINT "COMMAND: ";:INPUT LINE A$
221 IF LEN(A$)<2 THEN A$=SPACE$(2)
222 A$=LEFT(A$,LEN(A$)-2)
230 IF A$="GO" THEN GOSUB 310
240 IF A$="STOP" THEN STOP
250 IF A$="LIST" THEN GOSUB 600
260 IF A$="NEW" THEN 155
270 IF A$="DELETE" THEN GOSUB 700
280 IF A$="INSERT" THEN GOSUB 800
290 IF A$="TRACE" OR A$="NOTRACE" THEN GOSUB 910
295 IF A$="ENTER" THEN GOSUB 1000
296 IF A$="WRITE" THEN GOSUB 2000
297 IF A$="READ" THEN GOSUB 3000
298 IF A$="HELP" THEN 160
300 GOTO 220
310 ! ***** EXECUTE THE USER'S PROGRAM *****
320 IF M<1 THEN PRINT "ERROR--NO PROGRAM IN THE MACHINE":GOTO 580
330 L$="":PRINT "INPUT TAPE: ";:INPUT LINE L$:L$=LEFT(L$,LEN(L$)-2)+" "
340 IF L>80 THEN L=80:L$=LEFT(L$,80)+" "
345 L=LEN(L$):FOR I=1 TO L:I$(I)=MID(L$,I,1):NEXT I
350 INPUT "INITIAL TAPE HEAD POSITION: ";P
360 IF P<1 OR P>L THEN PRINT "ERROR--INVALID TAPE POSITION":GOTO 580
370 PRINT:PRINT "INITIAL TAPE":PRINT L$
380 S$="0"
390 ! FIND STATE (INSTRUCTION) OF MACHINE
400 FOR S=1 TO M
410 IF C$(S,1)=S$ THEN 440
420 NEXT S
430 GOTO 560 : !**** REACHED AN UNDEFINED STATE--END OF PROGRAM
440 IF C$(S,2)<>I$(P) THEN 540
500 IF C$(S,3)<>"R" AND C$(S,3)<>"L" THEN I$(P)=C$(S,3):GOTO 530
510 IF C$(S,3)="R" THEN P=P+1:IF P>L THEN L=L+1:I$(L)=" "
520 IF C$(S,3)="L" THEN P=P-1:IF P<1 THEN PRINT "ERROR--LEFT END OF TAPE REACHED":GOTO 560
530 S$=C$(S,4)
531 IF F=0 THEN 400
```

## FIGURE 2.

```
COMMAND: ? LIST



# 1       0    1    #    0
# 2       0    0    R    0
# 3       0    #    R    0
# 4       0         #    1


COMMAND: ? GO
INPUT TAPE: ? 00110011000111
INITIAL TAPE HEAD POSITION: ? 1

INITIAL TAPE
00110011000111
FINAL TAPE: 00##00##0000####
FINAL HEAD POSITION:  15


COMMAND: ? GO
INPUT TAPE: ? 1100110011110001
INITIAL TAPE HEAD POSITION: ? 1

INITIAL TAPE
1100110011110001
FINAL TAPE: ##00##00####000##
FINAL HEAD POSITION:  16


COMMAND: ?
```

## FIGURE 3.

```
COMMAND: ? LIST


# 1       0    0    R    0
# 2       0    1    R    1
# 3       0              -    3
# 4       1    0    R    1
# 5       1    1    R    0
# 6       1              -    2
# 7       2    -    R    2
# 8       2              0    2
# 9       2    0    R    4
# 10      3    -    R    3
# 11      3              E    3
# 12      3    E    R    6
# 13      4              D    4
# 14      4    D    R    5
# 15      5              D    5
# 16      5    D    R    9
# 17      6              V    6
# 18      6    V    R    7
# 19      7              E    7
# 20      7    E    R    8
# 21      8              N    8
# 22      8    N    R    9


COMMAND: ? GO
INPUT TAPE: ? 110011001
INITIAL TAPE HEAD POSITION: ? 1

INITIAL TAPE
110011001
FINAL TAPE: 110011001-ODD
FINAL HEAD POSITION:  14


COMMAND: ? GO
INPUT TAPE: ? 00001111011
INITIAL TAPE HEAD POSITION: ? 1

INITIAL TAPE
00001111011
FINAL TAPE: 00001111011-EVEN
FINAL HEAD POSITION:  17


COMMAND: ?
```

```
532 PRINT "TAPE: "#:FOR I=1 TO L:PRINT I$(I)#:NEXT I:PRINT " ",
533 PRINT "HEAD POSITION: "#P#"  INSTRUCTION EXECUTED: "#S
535 GOTO 400
540 S=S+1:IF S<=M THEN 440
550 PRINT "SEVERE ERROR--END OF USER PROGRAM REACHED WITH A DEFINED STATE"
560 PRINT "FINAL TAPE: "#:FOR I=1 TO L:PRINT I$(I)#:NEXT I
570 PRINT:PRINT "FINAL HEAD POSITION: "#P
580 RETURN
600 ! ***** LIST USER INSTRUCTIONS ******
610 PRINT:PRINT:IF M<1 THEN PRINT "NO USER PROGRAM":PRINT:GOTO 650
620 FOR I=1 TO M:PRINT "0"#I#TAB(10)#C$(I,1)#TAB(15)#C$(I,2)#
630 PRINT TAB(20)#C$(I,3)#TAB(25)#C$(I,4)
640 NEXT I
650 RETURN
700 ! ***** DELETE AN INSTRUCTION *****
710 PRINT:INPUT "WHICH INSTRUCTION SHALL I DELETE: "#P
720 IF P<1 OR P>M THEN PRINT "ERROR--GIVEN INSTRUCTION DOES NOT EXIST":GOTO 780
730 M=M-1:IF M<1 THEN 780
740 FOR I=P TO M
750 FOR J=1 TO 4:C$(I,J)=C$(I+1,J):NEXT J
760 NEXT I
770 PRINT "INSTRUCTION "#P#" WAS DELETED"
780 RETURN
800 ! ***** INSERT A USER INSTRUCTION ******
810 L$="":PRINT "ENTER INSTRUCTION TO INSERT: "#:INPUT LINE L$:PRINT
815 L$=L$+SPACE$(4)
816 L$=LEFT(L$,4)
820 IF LEN(L$)<>4 THEN 810
830 INPUT "WHERE SHALL I INSERT THE INSTRUCTION: "#I:PRINT
840 IF I<1 OR I>M+1 THEN PRINT "ERROR--CANNOT INSERT INSTRUCTION IN THE PLACE INDICATED"
845 IF I=M+1 THEN 880                                                          :GOTO 900
850 FOR J=M+1 TO I+1 STEP -1
860 FOR K=1 TO 4:C$(J,K)=C$(J-1,K):NEXT K
870 NEXT J
880 FOR K=1 TO 4:C$(I,K)=MID(L$,K,1):NEXT K:M=M+1
890 PRINT "INSTRUCTION WAS ADDED":PRINT
900 RETURN
910 ! ****** TRACE/NOTRACE ROUTE *****
920 IF A$="TRACE" THEN F=1 ELSE F=0
930 RETURN
1000 ! ***** ROUTINE TO ENTER A USER'S PROGRAM *****
1010 PRINT:PRINT:PRINT "YOU MAY NOW ENTER UP TO 100 INSTRUCTIONS FOR THE TURING MACHINE."
1020 PRINT "PROGRAM BEGINS EXECUTION IN THE INITIAL STATE OF 0.":PRINT
1030 PRINT "WHEN YOU ARE FINISHED ENTERING INSTRUCTIONS, TYPE A RETURN.  REMEMBER"
1035 PRINT "THAT 'R' AND 'L' CANNOT BE PART OF YOUR ALPHABET SINCE THEY DESIGNATE
1036 PRINT "FOR THE TAPE HEAD TO MOVE.":PRINT                                  DIRECTIONS"
1040 M=0
1050 L$="":PRINT "0"#M+1#" "#:INPUT L$
1055 IF LEN(L$)=0 THEN 1090
1060 IF LEN(L$)<>4 THEN 1050
1070 M=M+1:FOR I=1 TO 4:C$(M,I)=MID(L$,I,1):NEXT I
1080 GOTO 1050
1090 PRINT M#" INSTRUCTIONS ENTERED":PRINT
1100 RETURN
2000 ! ***** ROUTINE TO WRITE USER PROGRAM TO DISK *****
2010 INPUT "ENTER OUTPUT FILE NAME: "#I$
2020 OPEN I$ FOR OUTPUT AS FILE #1
2030 PRINT #1,M
2040 FOR I=1 TO M:FOR J=1 TO 4:PRINT #1,C$(I,J):NEXT J:NEXT I
2050 CLOSE #1
2060 RETURN
3000 ! ***** ROUTINE TO READ USER PROGRAM FROM DISK *****
3010 INPUT "ENTER INPUT FILE NAME: "#I$
3020 OPEN I$ FOR INPUT AS FILE #1
3030 INPUT #1,M
3040 FOR I=1 TO M:FOR J=1 TO 4:INPUT #1,C$(I,J):NEXT J:NEXT I
3050 CLOSE #1
3060 RETURN
9000 END
```

**end**

$2.50 ($3.00 In Canada)

# DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

## FOR USERS OF SMALL COMPUTER SYSTEMS

Number 52 February 1981 Volume 6, Issue 2

**PARFOR: A Theoretical, Parametrized FORTH-like HLL**
*H.T. Gordon*

**Sinclair's ZX80**
**Review of a Very, Very Small Computer**
*John B. Palmer*

**Small-C**
**An Implementor's Notes and a Bug Corrected**
*P.L. Woods*

**Z-80 Memory Test**
*Ray Duncan*

**PAN: One Activity of the PCNET Project**
*Dave Caulkins and David C. Harris*

**and more, even more...**

P.O. Box E
Menlo Park
CA 94025

DR. DOBB'S JOURNAL of
COMPUTER Calisthenics & Orthodontia

Volume 6, Issue 2
Number 52
February 1981

# Contents

## Articles

**PARFOR: A Theoretical, Parametrized, FORTH-like HLL** ..... 8
*by H. T. Gordon*

**Sinclair's ZX80: Review of a Very, Very Small Computer** ..... 14
*by John B. Palmer*

**POKE and PEEK to Your Heart's Content** ..... 16
*by T. J. Van Iderstine*

**PAN: One Activity of the PCNET Project** ..... 17
*by Dave Caulkins and David C. Harris*

**DECONCAT: A De-Bugging Tool for Concatenated Programs** ..... 18
*by Greg Smith*

**Small-C: An Implementor's Notes and a Bug Corrected** ..... 20
*by P. L. Woods*

**Z-80 Memory Test** ..... 22
*by Ray Duncan*

**The Electric Phone Book** ..... 42

**The Radio Shack Pocket Computer** ..... 43
*by Dick Greenlaw*

## Departments

Letters ..... 6
Software Reviews ..... 34
*Dobb's Ex Machina* ..... 36
Of Interest ..... 38
Book Reviews ..... 41

© 1981 by People's Computer Company. All rights reserved.

*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, California 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025.

Subscription Rates & Writer's Guidelines on page 42.

**Publisher**
Ann Merchberger
**Editor**
Marlin Ouverson
**Contributing Editors**
Dave Caulkins
Dave Cortesi
**Art Director**
Clifford West
**Artists**
Barbara Ruzgerian
Greg Gordon
Jeannie Barroga
**Typesetter**
Renny Wiggins
**Marketing Director**
Gavin W. Cullen
**Advertising Director**
Michael Madaj
**Circulation Manager**
Peter Clark
**Circulation Assistant**
Leah Dansby

# Editorial

Software piracy as an issue has inspired a predictable shower of editorial comments and disclaimers from every magazine and bulletin on the market. The tendency, unfortunately, appears to be defensive rather than showing genuine interest in the problem. It is as if, after publishing a standard note of abhorrence on the subject, editors consider the matter closed.

Disregarding any inherent emotional content, it remains that our legal system does permit commercial concerns to copyright software. It is also true, however, that innovation takes place more easily in an unrestrained environment. Another concern are those non-profit, non-business users who find the official and financial requirements for obtaining licenses and updates too cumbersome to encourage personal microprocessing.

Creators of software obviously deserve compensation, it is simply that the proper mechanism for doing so is not presently understood. *DDJ* and People's Computer Company do not condone theft of copyrighted materials. Developers and distributors have always shown greater motivation, and have provided better services, when their larders are not empty. We do encourage software libraries, low-cost copying services and non-proprietary programs. It is most important that a large pool of information exist in the public domain. This body of material carries a germ of vitality which thrives on active interchange, usage, commentary, revision and free implementation by many different kinds of users.

The spirit of open participation pervades People's Computer Company and its publications. We will continue to publish general software, utilities, algorithms and theories for the benefit of the entire microprocessing community.



Marlin Ouverson
Editor

# Letters

## Strings in FORTH

Dear *DDJ*:

I would like to comment on the subject of strings in FORTH, as discussed in Mr. Deane's recent article (*DDJ* #50). I have worked on implementation of strings in CONVERS, a FORTH-like language, and I have found that it is quite convenient to have a string comparison routine which does more than determine if two strings are equal. My $COMPARE takes two string addresses from the stack and leaves a 1, 2, or 3 on the stack, and where 2 indicates the strings are equal, 1 indicates that the top string is "greater than" the bottom string, and 3 indicates the top string is "less than" the bottom string. This information makes it very easy to sort or search string arrays, and of course can be used to provide the string equal function if desired.

```
: S= $COMPARE 2 = ;
```

It is important to make the comparison give "greater than" and "less than" results in the sense of alphabetical order. The almost useless string comparisons in CBASIC serve as an example of how not to do it.

I also think that it is better to store a string count with the variable rather than using a null to signify the last character. In this way it is possible to use strings for general purpose buffers. In my implementation, when a string variable is executed it places the starting address of the data on the stack. The first two bytes of data are the maximum string size, the next two are the current count, followed by the string contents. This enables strings to be as large as available memory, but in actual practice this has not turned out to be very useful, so 255 byte strings are probably adequate.

Yours truly,
William B. Brogden
Island Cybernetics
P.O. Box 208
Port Aransas, TX 78373

Dear *DDJ*:

At MMS, we very much enjoyed reading Ralph Deane's fine article defining some string routines for the FORTH language.

We agree with his premise that floating point numbers and string-handling words are major missing features of most common FORTH versions available today. That is why MMSFORTH, our own version of FORTH for the Radio Shack TRS-80 Model I and Model III, provides string operators – the ones described and more – as a standard load option, and provides floating point number handling – a more complete set than that provided by the TRS-80 BASIC ROM – as an inexpensive utility routine. MMSFORTH is a full, professional version of FORTH which also provides double-precision integer arithmetic, various types of arrays, sorting algorithms, graphics capabilities, and more. Our standard versions for disk or cassette sell for under $100, and run in as little as 16K RAM.

Because MMSFORTH has become the standard professional FORTH on this very popular microcomputer, with its own users newsletter and more than 1,000 registered users, we received considerable feedback on their own likes and dislikes. Your FORTH fans may be interested in our reading of that feedback:

1. Floating-point math is sought by many FORTH neophytes but needed by few advanced users. Although better techniques often exist, beginning FORTH programmers may require FP to get their program up and running and, except for the theoretician, this is the bottom line.
2. String-handling is a necessity for nearly all types of serious applications programming. And once FORTH is applied thusly, it shines. MMSFORTH string words follow those in Level II BASIC (a Microsoft extended BASIC), which flattens the learning curve for its users. Adhering to some standard FORTH string nomenclature could create unnecessary confusion unless it is grounded in a similar common existing word set.
3. As with floating-point, beginners often want a full Z80 Assembler. MMS also provides this option, but we find that our own programming loses nothing by building on our full 8080 Assembler, instead. It's nearly 2K-bytes smaller and at least as fast – including its multi-step routine for block moves. Typically, users doubt this until they get good enough to compare. *Chacon a son gout!*
4. FORTH users want practical applications programs, ranging from real-time graphics games to sophisticated business programs. In properly commented FORTH all these become cost-effective, easy to maintain and enjoyable to program. MMS provides a powerful and speedy database management system, a games package, a significant set of additional utilities, and is developing a complete business system, a flexible communications package and an advanced word-processing system. Custom programming is a major activity at MMS, and FORTH makes it affordable.

FORTH is a programmer's dream come true. Thanks for sharing it with your readers! Any *DDJ* reader wishing a flier on MMSFORTH for the TRS-80 is invited to write or call.

Sincerely,
A. Richard Miller
Miller Microcomputer Services
61 Lake Shore Road
Natick, MA 01760

Dear *DDJ*:

In my article "A Proposal on Strings for FORTH" I supplied a demonstration program which computed the Soundex code for an entered name. I have since discovered that while it runs okay on a 6502 FORTH system, it will not work on a FIG FORTH-based system. This is due to a difference in the way that DO LOOP parameters are stored. A fix for this is as follows.

*DDJ* #50, page 43, Figure VII:
Change line 5 to:

```
: ITH DUP N$ MID$ DROP C@ 64 - ;
```

Change line 10 to:

```
: COMP N$ LEN 1 + 2 DO I ITH IST
    IF KTH TEST IF BLS THEN
```

These changes make the Soundex program compatible with both 6502 FORTH and FIG FORTH.

Ralph Deane
Box 27
Little Fort, BC V0E 2C0
Canada

## Softcost Software

Dear *Dr. Dobb's*:

I've implemented RATFOR, Mssrs. Kernighan and Plaugher's rational alternative to FORTRAN, on the Honeywell 6000 series machine. I'd be pleased to make a Honeywell tape of this version for anyone interested for the price of reproduction and return postage. Interested readers may establish communications with me by letter.

Sincerely,
Walter J. Kirsch III
4316 Starr Jordan Drive
Annandale, VA 22003

Since 1976, the most innovative magazine for the technically-sophisticated microcomputer user.

# Dr. Dobb's Journal

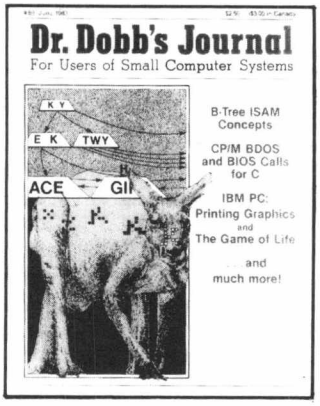
## For Users of Small Computer Systems




1976



1983




Have you been looking for a magazine that will provide you with serious articles on programming? *Dr. Dobb's Journal* can give you the technical depth you need to understand the latest software advances. We cover every microprocessor with articles on C, Forth, Unix, CP/M and much more. Our reputation for editorial quality is well known. Subscribe now, and discover what industry insiders have enjoyed since 1976.

Coming articles will cover:

- new language development
- 8- and 16-bit technology
- operating systems
- compilers, editors
- assemblers/cross-assemblers
- clever algorithms

**$25**

one year subscription
12 issues
saves $5 off the newsstand price

**YES!** Enter my subscription to *Dr. Dobb's Journal* and bill me later.

name

address

city state zip

**Mail to: DDJ, P.O. Box E., Menlo Park, CA 94025**

## N-Logs

Dear Editor,

Here are some comments on the article "N-Logs" in *DDJ* #50. The article contains a number of inaccuracies.

On page 5, it says that Napier used the e=2.71... as his base because it is easy to compute. I will quote loosely from the reference 1:

*Napier had no notion of the base . . . Briggs, after Napier's death, published tables of decadic logarithms . . . The tables of natural logarithms appeared almost simultaneously, but their importance was not recognized until the infinitesimal calculus was better understood.*

The natural logarithm $\ln(x)=\log_e(x)$ has many unique properties. For example:

$$\ln(-1)=i*\pi$$

However, "the ease of the evaluation" is not one of them. In the equation above, the "i" is imaginary unit ($i*i=-1$) and $\pi=3.14\ldots$ is Pythagoras' number.

Also, the "mistakes of the Bell labs" discussed on page 6 are open to argument. Decibel (one tenth of a Bel) is used in acoustics to measure both the intensity (i.e., power per unit area) and pressure level of sound ( = SPL). As a result of that, when you double your distance from the source of the sound, the intensity of sound is decreased by about 6dB; the SPL, which is more common when describing sound, would decrease by about 3dB if definition would not be modified. This confusing situation would also exist when discussing gains: the same amplifier would have gain 6dB or 3dB according to quantity used in calculation. That is the reason why textbooks stress the use of the logarithmic scale for the power only.

Whether we respect this convention, and also whether we treat dB as a unit, is certainly a matter of taste. The merit of the proposal is not affected by that. There are, however, more serious sins committed in that article, sins of inconsistency.

If we indeed do extend the use of the logarithmic scale to all quantities, then what the author calls 6-log is simply the quantity X expressed in microbels. It is unconventional, but acceptable to say that the speed of light is 5477 mB (millibels), provided that the value one meter per second was agreed on as the reference value.

Finally, the convention explained at the end is an unfortunate choice. If we want to avoid the superscripts, as some computer printers must, let's stick with the usual:

$$5.10^3=5.E3=5000$$
$$e^3=\exp 3\approx 20$$

The EXP on some calculators is short for Exponent, not for the exponential function!

Annotated references:

[1] D. J. Struik: *A Concise History of Mathematics*, pp 95-97. (This book is one of the better ways of spending $3; to order, ask Dover books for their science catalog, 180 Varick Street, New York, NY 10014.)

[2] G. Porges: *Applied Acoustics*, pp 16-20. (This is one of the few books on this subject which combine a good theoretical foundation with useful facts. Published by Edward Arnold in 1971.)

Peter Mikes
16641 Cowell Street
San Leandro, CA 94578

Dear *DDJ*:

I had immediate use for N-Logs (*DDJ* #50). I needed to store a large data file of the Dow Jones Industrial closings. By using the N-Log logic, I stored the data as an integer array. The following program should illustrate the method used. It is also a useful program to get a "feel" for what is happening.

```
10 INPUT "UPPER LIMIT";U
20 INPUT "LOWER LIMIT";L
30 IF L>=U THEN STOP REM U
   MUST BE GREATER THAN L
40 IF L<1 THEN STOP REM L MUST
   BE 1 OR GREATER
50 B=EXP((LOG(U)-LOG(L))/(2+
   LOG(L))
60 C=32767/LOG(U/B)
70 INPUT "DATA";A
80 D=LOG(A/B)*C+.5
90 IF D>32767 THEN D=32767
100 IF D<-32767 THEN D=-32767
110 D=INT(D)
120 E=EXP(D/C)*B
130 PRINT D,E
140 GOTO 70
```

Line 50 is a centering procedure to produce an evenly distributed set of positive and negative values based on the upper and lower limits. It produces a log which lies at the midpoint of a linear scale of logs from the log of "U" to the log of "L."

Line 60 produces a constant used to spread out the resulting data in a range of + or -32767. D = the integer value to be stored. E = the returned value after processing from integer form.

I anticipate (right or wrong) the Dow Jones will remain between 3,000 and 300 for the next five years. Using these as my upper and lower limits, I get the following:

B=948.684
C=24861.1

| Dow Input | Integer | Returned |
|---|---|---|
| 956.19 | 224 | 956.18 |
| 956.20 | 225 | 956.213 |
| 956.23 | 225 | 956.213 |
| 956.24 | 226 | 956.247 |
| 2785.58 | 30656 | 2785.54 |
| 2785.59 | 30657 | 2785.64 |
| 2785.68 | 30657 | 2785.64 |
| 2785.69 | 30658 | 2785.73 |

Stock volume, which can be dynamic at times, is stored in the same manner since accuracy is then spread evenly, based on the size of the number entered.

Regards,
Mark E. Brown
5 Maynard La.
East Islip, NY 11730

Dear *DDJ*:

This letter is to announce the availability of my programs which previously appeared in *Dr. Dobb's* in a new format. They are now also available on Micropolis disks as well as the 8-inch IBM compatible disks on which they have been available, both for CP/M. These are an improved version of the LISP which appeared in the December 1978 issue, and the table driven assembler which appeared in February 1980. The 8-inch disks continue to be available for $10 each. The Micropolis disks are now available for $15 each. Persons outside the United States and Canada should include an extra $2.

Sincerely,
Darrel J. Van Buer
1522 Brockton Avenue, Apt. 5
Los Angeles, CA 90025

## Small-C Compiler

Dear *DDJ*:

While implementing a CP/M version of the Small-C compiler (*DDJ* #45 and #48), I discovered an error in the runtime library routines for the left and right shift operators. The C language defines any shift from zero to the length of the object as legal. The routines ccasl and

*(continued on page 37)*

# PARFOR

## *A Theoretical Parametrized, FORTH-like HLL*

**by H.T. Gordon**

My first step in the chaotic HLL area was an attempt to understand the operation of FORTH (*DDJ* #51). My tentative conclusion was that FORTH now is a high-level assembly language for a virtual machine that ought to have a much more powerful core structure than it has. It is my belief that development of a better virtual machine is not a one-man task. The most important quality of an HLL is not speed or efficiency, but acceptability to a quite large number of users. I doubt that any HLL will ever be truly acceptable to everyone, even if its creator is a genius. I vaguely see that distinct types of HLL's are possible. Within each type, optimizations are possible such that several different excellent designs could be made to work equally well. If so, only one design is needed, so that the number of exchangeable, complex programs will be maximized. Reduction in the number of widely-used HLL's would also allow CPU designers to optimize "real" machines for one (perhaps even more than one) virtual machine. Minimizing variations at all levels makes possible economies of scale. CPU's, systems, and complex programs would all cost less.

The cost of development of any innovation is minimized by extensive *theoretical exploration*. I am convinced that ultimate quality can be maximized by the participation of many human minds, including those of amateurs like me. We cannot see as far as experts do, but we may sometimes see things that their biases cause them to overlook. Most of our ideas will indeed be of no value, long ago evaluated and rejected by the *cognoscenti*. Any expert who bothers to read what we write will be utterly bored. What interests them is the work of other experts, and the needs of the rich and powerful minority within our society. That's the IBM complex which dominates computer science. Big is beautiful.

The purpose of this note is to offer a purely theoretical extension of the "parametrized command" concept that arose from my earlier, casual study of FORTH. My hope is that it will be criticized (perhaps enhanced, perhaps rejected) for sound, logical reasons by more able minds than my own.

**Simplifications of parametrized command notation.** One of the human problems with all languages is the tedious keying-in of the source code. FORTH has a fair number of single-character commands, trading off mnemonic value for less keying effort, e.g., the period (.) character that commands pulling the number on TOS (top of variable stack) and printing it. My

initial proposal for parametrized commands was ?CXY, where ? is the identifier for this type, C is some one ASCII character that specifies the actual operation (e.g., S for a stack-manipulation), and X and Y are decimal-digit parameters. Since the number (*n*) of characters becomes the first byte of the dictionary "entry," the generalized form CXY would be entered as 3C, the 3 acting as the identifier and the C as the operation-code *expecting* (for actual keying-in) variable X and Y digital parameters. Thus, stack-manipulation types would exist in the dictionary as the eccentric entry 3S, and keying-in S10 would execute the equivalent of the present DUP command code, S20 the present OVER, etc. I shall not reiterate the many variants outlined for stack-manipulation in my earlier note. All would become three-character commands, the minimal possible length. Reserved "words" exist in many HLL's. This merely extends the concept to reserved word lengths. Parametrization adds some mnemonic value, but this is not the *forte* of FORTH! PARFOR, if it ever is born, will be more laconic than STOIC.

**Generalized loop-parameters access.** The existing command I causes the value of the current Index integer to be copied from the return-stack to what I term the $S_0$ location of the variables stack. In the Bruce Airy version of FORTH that triggered my own thinking, there are analogs (J, K, L, . . .) that allow similar access to the current Indices of the successive outer loops within which the presently-operational loop may be nested. The type IXY code would allow access to anything desired. X=0 would mean the operational loop within which code is now being written or used. Its parameters were pushed on the return-stack in the order: Index, Endex, and return-address, identifiable by Y=0, 1, and 2. Thus, I00 would be equivalent to the present I command, I01 would copy the current (constant) Endex to $S_0$, and I02 would copy the current return-address to $S_0$. Since the parameters of the next outer loop are similarly stacked just above the current ones, I10 would work like Airy's J, I20 like his K, etc., and all other parameters would be equally accessible. You lose the delightful brevity of I, but you save the limited supply of single-characters for possible powerful, parametrized commands. You gain access to things you may not want, but as I have said before, there is never an obligation to use all available power.

**The data-structures problem (for now just numerics).** This is the supreme problem in computer science, as many important books attest, as well as the HLL's designed to handle certain types optimally. I have read, but not mastered, only M. C. Harrison's *Data Structures and Programming* (Scott, Foresman & Co., 1973). My impression is that FORTHians skip lightly over this snakepit, relying on user-extensibility for do-it-yourselfing whatever may be required. Still, an HLL, especially in the only-a-dream stage of PARFOR, ought to grasp this nettle and consider whether its core structure can be designed to facilitate variety and complexity of data-structures. Nothing so fundamental should be an afterthought. CPU designers are keenly aware of this, and what is an HLL if not a softwarish CPU?

Multidimensional, complexly interlinked structures are far beyond me, as is the user-extensible BALM language described in Harrison's book. Numerics, being "linear data-objects," seem more intelligible. My impression is that most versions of FORTH, even if executed by an 8-bit CPU, operate on 16-bit numbers, though FIG FORTH has a few commands that handle 8-bit bytes and some "double-precision" ones that work on 32-bit numbers. (Maybe more bytes, since an entire byte may be in use to store the sign of the number?) Generalization presents a problem not unlike the one I briefly noted for microprocessors (*DDJ* #49, page 22): should you specify precision by using a specialized opcode implying it, as in FIG FORTH, or by a one-time prefix opcode to modify only the immediately-following action, or by a permanent (until negated) opcode modifying all subsequent actions? This last type is used in the number-base-setting of FORTH numerics, the base (binary, decimal, or hex) remaining in control until reset to a different base. Of course, a flexible base-type is possible using a parametrized primitive such as BXY. X would be 1 for binary, 9 for decimal, F for hexadecimal (hex coding being always implied). Y would be 0 for indefinite duration, 1 for one-time operation (potentially extensible all the way up to F for up to 15 re-uses before lapsing back to the saved permanent-base).

This is trivial, but the concept (as I suggested in *DDJ* #49) is applicable to bit-precision also. Although handling of 4-bit nibbles might be of interest, perhaps even of single bits, the precision-setting primitive LXY might use X to specify the number of bytes to be handled as a unit. L10 would be a permanent-set to single-byte precision, L20 to 16-bit, LF0 to 120-bit (15 bytes). This last would modify a stack-manipulation like $S_{10}$ to copy the top 15 bytes of the variables stack into the next 15 available locations! That would be quite a large integer (of the order of 10 to the 36th power!). Of course, trying to multiply $S_1$ by $S_2$ would blow up the stack, unless safeguards were built-in or an automatic shift to exponential notation (as in pocket calculators) were caused.

Who might need such big numbers? Perhaps no one, but if it's feasible why not allow it? This is a good time to point out the trade-off of endowing each command-code character with control over execution: one slip can spell disaster! There has to be a price for the combination of power, generality, and conciseness. Even those who would never use the power must run a greater risk of error.

Bruce Airy's version includes floating-point operations, but I see no virtue in decimal points. A signed integer paired with a signed integer exponent can approximate a wide range of real numbers. How wide? Assuming that a 15-byte upper limit for integers is workable, at least one of them is needed for the number, leaving 14 for the exponent, a value of colossal size but infinitesimal precision. The other extreme would be a 14-byte numeric with a 1-byte exponent, sacrificing range for precision, but still pretty big. This would be specified by altering the interpretation of LXY, making all possibilities permanent-set (until reset). Y=0 would mean an integer, and thus L20 would set the

> **"I am convinced that ultimate quality can be maximized by the participation of many human minds ..."**

numeric format to 16-bit integers. L22 would mean pairing of a 16-bit integer with a 16-bit exponent, etc. In principle one could have LFF, but a length of 30 bytes might be hard on the stack! The L logic could be written to reject values of X+Y greater than 15. Greater expertise than mine is needed for decisions of this kind. Nor do I feel qualified to go into logarithms, though I know their value, and I feel that they belong in a math core. As usual, the problem is what ought to be in the core structures and what ought to be left for special-purpose programs. Generality must dominate the core, combined (as far as possible) with speed and efficiency.

**Print command – output devices and formatting.** FIG FORTH has many typically specialized commands targeted on what seems to be a single display terminal. Proliferation was probably forced by the rudimentary character of the decimal-point primitive, that just pulls $S_1$ from the variables stack and prints it, assuming a length of 16 bits. My proposed LXY precision-setting command ought to alter the "." operation to print whatever length it specifies. But what might be done with the command-type .XY? A program may well need to address other output devices than the user terminal, and X=0 could be used to specify the terminal. Other values (up to 15 if hex-coded) involve switching and interfacing problems that "." logic must deal with in a generalized way, perhaps by jumps to segments of a system-monitor program tailored for operating whatever devices are in the system. What of the information in Y? There are many possibilities. The value 0 could retain the present effect (pull $S_1$ and output it), while 1 would output $S_1$ without pulling it, 2 would similarly output $S_2$, etc. More knowledge of device-control than I now have is needed to optimize this. The FIG FORTH specifics seem to provide a few simple output formats, more sophisticated ones demanding programmed construction.

**Read operation.** There is now no single-character input-read command code, though a ? might be made the equivalent of the "." output command. Again, FIG FORTH has many specific codes, presumably created in response to frequent needs. Once more I am at a loss in defining an ?XY code, though X could logically specify the input device. What is required is a clear general concept of the I/O in a system. This is an area where only user-experience can suggest optimal general modes, a problem that has often puzzled me at the assembly-language level.

**Arithmetic operations.** Like all other FORTH commands, the operators for add (+), subtract (-), multiply(*), and divide (/) are very specifically defined. Stack locations $S_1$ and $S_2$ are pulled, $S_1$ is used as the first operand, and the result is pushed on the stack (becoming $S_1$). Stack-manipulation is required to make use of, or to prevent loss of, the existing data. Pulling is clearly necessary to avoid blowing up the stack, especially in loop operations where even a small imbalance would accumulate. Still, if you pull two units and replace them by only one, or if (to save them) you duplicate them and then operate, you have either shortened or lengthened the stack. The all-wise programmer, of course, takes care of that, and the operator logic may detect overflow and underflow problems as the better pocket calculators do. I do not know whether shifts to exponential notation are automatically done, calculator-style, though it would be possible to encode an alteration of the (kernel-stored) notation-control directly set by LXY. This raises the important question of a PARFOR status register, which I shall deal with later on. For now, the question is whether parametrizing the arithmetic operators might be useful.

By analogy with the SXY stack-manipulation, +XY could specify the stack locations to be used, X being the first operand. Thus, +23 would denote ($S_2 + S_3, \rightarrow S_0$), nothing being wiped out. The programmer could "drop" operands by using S33 or S44 or a sequence of two S33's (the numeric change is required because the add lengthened the stack by one). Some comparative programming would be needed to see if there is a gain or loss of code or timing, in general, sufficient to decide. My guess is probably not. But there are other possibilities for XY interpretation. X=0 could be a programmer-command to forbid any further execution if either underflow or overflow occurred; X=1 would allow only underflow; X=2 only overflow (with retention of the most significant digits); X=3 overflow (retaining least significant digits); and X=4 an automatic shift to exponential notation limiting the integer precision to 4 bytes (and analogously for 5, 6, etc.). I cannot, offhand, think of a use for Y when X is less than 4, but Y can be used to set the limit for the exponent, which, if exceeded, will stop operation. Perhaps Y=0 could signify permission to continue building up the integer length to the limit value present in X. The advantage is that the programmer acquires some control over the execution-logic of the operator.

**The case for an HLL status-register.** So far I have only argued for a more information-rich command-coding that does not seriously alter the architecture of FORTH. The concept of creating the high-level analog of a CPU status-register first arose when I understood Bruce Airy's view of FORTH as the equivalent of a CPU. It had a "state" variable hidden in its kernel, automatically modified by operations, vaguely like the interrupt bits of a CPU. Program-set status, however, with its associated branch commands, was a rather crude zero or non-zero temporarily pushed on the variables stack. The whole thing struck me as extremely primitive (of course I had only Airy's description), reflecting the state of CPU's in the pre-1970 era and also the gradual accretionary evolution of FORTH in the mind of its maker (like the seemingly inconsistent branch constructs of LOOP and IF). When you've built something up over a long time, you're reluctant to tear down the ancient structures and

"... there is never an obligation
to use all available power."

rebuild from the ground up, preferring just to add on new and better things. That's why it is good that we oldsters die out – the young feel free to raze our rats' nests and start from scratch, possibly to make something better!

There's a plethora of statuses in any complex operation. Numeric or non-numeric. Positive, negative, equal. Zero or non-zero. Integer or exponential notation. Unit length 1-bit, 4-bit, 8-bit, etc. Some may be important enough to base decisions on. Others may have information value for analysis and debugging. In a microprocessor, circuitry sets a limit. In a virtual CPU, the limits are more subtle: timing losses, memory consumption, the capacity of the human mind. These are nonetheless *wider* limits, and an HLL status register can have the dimensions of a decision-table.

The status registers of microprocessors have been often described (briefly by me, in *Microcomputing*, April 1980, page 176). They store information bits, some in a continuous automatic mode, at a minor cost in circuitry with no timing penalty. Usually four of them control branch or jump instructions. The register is valuable in program debugging. In a FORTH-like HLL, the basic unit is the 8-bit byte, allowing 256 distinct states, and the register can be of indefinite length and user-extensible. However, there is a price both in memory and timing. The decision as to which primitives should update which status bytes automatically, therefore, requires thought.

The existing kernel-stored, dictionary-status addresses (HEAD, the first, and LAST, the last byte of the most recent entry) clearly belong in the register. FIG FORTH seems to have other entry-addresses, whose operation is not clear to me. The STATE variable, determining execution or compilation, is a status byte. There may be merit in a zero/non-zero status-register byte, to remember a prior status and allow its re-use later. Also useful might be a negative-status, a positive (non-zero) status, and a set of status bytes arising from a generalized COMPARE command: equal/unequal, less than/not less than, greater than/not greater than. Possibly the current stack-pointer addresses ought to be in the register sequence. A print-the-status-register primitive would allow the programmer to view all this at once, whenever needed. Perhaps the count of the number of loops created and terminated (even of their BEGINs and ENDs, and DOs and LOOPs) would also be useful. As always, user experience will reveal whether other statuses are desirable. The numeric-operation status could include the number of bytes in the integer and also the number in the exponent. A zero in the second would mean a pure integer. A zero in the first might denote non-numeric status. The programmer-specified controls that I suggested for the +XY primitive (also feasible for the other operators) could be saved in the register, and also things like overflow and underflow statuses. The list is already quite long, though not exhaustive.

**A parametrized conditional-jump primitive.** Assuming that a sequence of status-register bytes exists and is updated in the kernel, the existing IF could be replaced by FXY. This would pick up the correct jump address as now, but X would specify which status byte is to control the jump, and Y would be either 0 or 1. If X=0 specifies the zero/non-zero status byte, F00 would interpret as jump-if-zero, F01 as jump-if-non-zero. There would be analogous jump-if-negative, jump-if-equal, jump-if-overflow, etc. possibilities, including user-defined flags.

The CASE construct, allowing more than a bifurcation, is available in principle as a logical extension of the FXY, if the programmer sets the value of Y to a hex integer from 2 to F (F being reserved as a terminator). The preceding decision-logic is expected to set the "case-byte" in the status register to one of the possible values. The code FX2 will compile the address of the F primitive logic, leave the jump address empty, and be followed by the case-2 execution logic, terminated by an FX3. This will compile the address of F again, but fill the empty FX2 address of the FX2 with *its own address* (so that there will be a jump to FX3 if the case-byte was not 2), and leave an empty space of the preceding FXY with its own address. What of the X value? This can be used to warn the compiler of the number of cases to be encountered. The case status-byte can be used during compilation to store this number. An initial F42 would store the number 4. The compiler then expects a sequence: F33, F24, F15, and a terminator F0F, and compel the programmer to keep things in order! This might reduce the error-proneness of the CASE construct.

The above still leaves FXF uninterpreted for values of X from 1 to F. Readers may have noted that the F0F definition is equivalent to the THEN of existing FORTH, with all else only a playing on the possible intricacies of the IF. Also, moving the status into a kernel-updated register means that whatever is on the stack no longer matters when the F logic is executed. The ELSE can therefore be the alternate IF, e.g., . . .(decision-logic) . . . F00 . . . (non-zero status logic) . . . F01 . . . (zero-status logic) . . . F0F . . . Note that if no zero-status logic is needed, but something ought to be done if there was an overflow, the F01 can be replaced by an FX1 (where X is the status byte for overflow).

It may be useful to reiterate the six distinct operations possible in F compilations: (1) Push onto the variables stack the current address in the compiler command-counter, and increment the (normally-zero) jump-status byte in the status-register. If the command does nothing else, it is a "pure initiator" like the FORTH BEGIN. This could be the F1F. (2) Pull address previously stored on the stack, and decrement the jump-status byte. This would make no sense if the jump-status byte is at zero, so this would "inhibit" step two. (3) Compile into the program sequence the execution-address of the F command logic, and then the XY numerics that logic will need, as with all normal parametrized commands. Since F logic executes a jump, however, the next space in the program must get filled by the correct jump-address, now or later. (4) If the jump-status byte is zero, there is no address left on the stack. Therefore, the jump-address space is left vacant and step one must be done, so that a subsequent command will be able to locate and fill the vacancy with its own current address, creating a forward jump. (5) If jump-status is non-zero, a command may do step two and use the stored address to locate the vacancy and fill it with its own address. If nothing else is done, this is a terminator like the FORTH THEN, for which I have already proposed the F0F coding. (6) If jump-status is non-zero, and step three is done, a command may do step two and use the stack-stored address to fill its own jump-address, creating a backward jump. An unconditional backward jump, like the FORTH END, could be encoded as FFF, but this creates an indeterminate-loop problem.

"... given a task at a certain level of complexity, one ought to do it in the simplest possible way."

Types like FX0 or FX1 might allow a conditional exit, but we must first see how these IF-like operations must normally work. An initial F00 is inhibited from doing step two since the jump-status is zero. It does step four (a sequence of steps three and one). However, a subsequent F01 (working like an ELSE), finds a jump-status of 1, enabling step five (that includes step two, and then step four, so that the address of the F00 is replaced on the stack by that of the F01, with jump-status reset to 1).

Such chains have no logical limit, unless terminated by the F0F. It is clear that the sequence F1F (BEGIN) . . . F00 . . . cannot work if the F1F does only step one! The F00 "sees" a jump-status of 1 and will try to fill a non-existent jump command with its own address! This can be inhibited if the F1F pushes a zero onto the stack following its own address. When the F00 executes step two and finds that it has pulled a zero value, it shifts its execution to step three and step six, leaving the jump-status at zero to signal termination and creating a conditional backward jump, allowing exit if the zero-status byte eventually gets set to 1.

This type of loop still does not allow the equivalent of the BEGIN . . . IF . . . END construct of FORTH, where the IF allows conditional escape from the indeterminate loop created by the unconditional jump from END back to BEGIN. The IF must now *ignore* the BEGIN address and work as if it were an initial IF, doing only step four, and the FFF (END) must fill its vacant address with the first address of the post-END escape logic. What is needed is a different type of BEGIN (perhaps FEF) that would push first its own return address, then FFFF on the stack. A subsequent F00, on executing step two and finding this odd address, would shift execution to step four, leaving a jump-status of 2 in the register. The terminating FFF would then do both step five and step six.

**A pause for reflection on the rationale of PARFOR.** Readers who have persisted to this point in my description of the structure of an imaginary HLL must be wondering why I am maximizing its complexity. A few may recall my earlier paeans of praise to the goddess of simplicity! I still worship her in the sense that, given a task at a certain level of complexity, one ought to do it in the simplest possible way. That way may have to be quite complex. Reality is complex. Complexity is unavoidable. The only question is *Where should it be?*

The aim of PARFOR is to put it in the machine code of the control programs of the kernel and the primitives. It is there that the tiny elite of assembly-language experts can optimize operations for the specific CPU that does the ultimate execution. Complexity doesn't phase *them*! The problem has long been recognized at the machine-code level, in deciding whether to encode a small block of instructions in-line or as a subroutine. If it's going to be used often, subroutining it will trade-off speed of execution to save memory (often done in ROM design where memory is at a premium). Another way is to create complex, multi-purpose subroutines that handle many parameters. That becomes even more appealing as one's expertise in machine-language grows, since main programs can become simple, just controlling subprograms.

The FORTHian view is well-presented in a recent article by Bill Ragsdale of the FIG (*BYTE*, October 1980, page 274). Their design criteria for FORTH are: 1) correctness – most important to a user, 2) portability, 3) *simplicity* of built-in commands – *most important to FORTH*, 4) clarity, 5) generality, 6) execution speed, 7) memory compactness, 8) compilation speed, 9) historical continuity (I'm not sure what *that* means), 10) pronounceability, and 11) teachability.

It's at point three that PARFOR diverges from FORTH. That decision trades-off linguistic simplicity at the price of program complexity. FORTHians know that "a greater burden is placed on the programmer, who must create *application-specific* functions. This characteristic is tolerated, even cherished, by FORTH programmers: but who better understands the demands of the application, the application programmer or a distant compiler writer?"

To me, the real question is Who has the more brilliant mind and the broadest and deepest knowledge and experience? In my introduction I argued that amateurs can play a minor role, and that professionals err when they ignore us. However, star roles and center-stage belong to the experts. As an amateur, I don't cherish writing complex programs. I would relish having expertly-designed, bug-proofed tools to use. That's the rationale of operating systems like CP/M and other expert creations, including all HLL's.

**Some concluding thoughts, and a hazy PARFOR architecture.** My first article on FORTH was not easy to write, and was intended to do some ice-breaking. This one has tried to initiate some building, an even more fatiguing task (no doubt for readers as well)! The possibilities of even parametrization are immense, and *it is only one of many possible approaches* to fuller utilization of the information-carrying capacity of the characters in command-codes.

As a kind of last gasp, I shall try to complete the interpretation of the RXY stack-rotation command casually mentioned in my earlier article. The existing ROT command of FORTH, that executes ($S_3 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3$), was defined by R31, and the corresponding counter-rotation by R13. The operation of a legion of analogous rotations is commanded by using other non-zero parameters. Interpretation is also possible for R0Y and RX0. Though the $S_0$ location is undefined, it can be momentarily set to zero. R20 will cause ($S_2 \rightarrow S_0 \rightarrow S_1 \rightarrow S_2$), a kind of "left-shift" in which the previous $S_2$ is lost and replaced by the previous $S_1$, while $S_1$ is now zero. R02 will cause an analogous "right-shift," with $S_1$ lost. It would be possible to use a kernel (status-register?) storage area as the effective $S_0$, so that the shifted-out data would not be lost, and in fact could be shifted-in like the carry bit in CPU rotate-thru-carry instructions.

How useful might this be? More generally, how far can one usefully push the analogy of an HLL to a CPU? FORTH now has one Protean working-register – the variables stack – where nearly everything is done. It implements named dictionary entries to store 16-bit units, and the ! and @ commands to move these to and from the stack. If my "unit-length-set" LXY is implemented, a storage capacity as large as 16 bytes might be needed. One might have a parametrized variable VXY in the dictionary with this capacity. Values of X from 0 to F would speci-

**"The possibilities of even parametrization are immense ..."**

fy one byte in the sequence. Y=0 could be interpreted as a command to push that byte on the stack (the $S_0$ location), and other Y values as a command to store the $S_Y$ stack value into the Y storage location.

I do not wish to stress any specific operation (needs thought!), only the fact that this kind of dictionary variable is beginning to look like an *auxiliary working register* and perhaps belongs in the kernel together with the status-register. Readers are aware that the only parametrized command in existing FORTH is the backward-jumping DO/LOOP, defined for 16-bit numerics, with one of them an auto-incrementing Index. One wonders if a DXY analog, whose Index could also index some parametrized commands within the loop, would be worth having? Limited range, but greater power.

This is no longer an alteration of detail but of the very architecture of FORTH, bound to dismay FORTHians much as the growing-up of a child can be psychically disturbing to its parents. Not yet too much, since it's hardly even an embryo! Without a lot of expert design-decisions and intense effort, it will go nowhere. Bill Ragsdale's quote from English expert C. A. R. Hoare, that the most difficult problem of language design is deciding what to leave out, is not yet relevant. PARFOR has barely begun to collect all the things that might be put in. If that ever gets done, it will encounter the Noah Problem: how will all these beasts interact inside the Ark? The real criterion will be *elegance*, superior to sheer power and much harder to achieve. Simplicity is only one of the subcriteria of elegance.

The hypothetical architecture of PARFOR would consist of a kernel and many primitives, using the two stacks and a complex status-register. Primitives would be mostly (all?) parametrized, with very complex, optimized machine code, and some might function as auxiliary working registers. The low end of the status register would consist of a sequence of status bytes, some set or reset automatically by certain primitives and perhaps by special set/reset command primitives. The latter kind of intentional programmer-control would be implemented for other status parameters, such as those defining the configuration of the variables stack (e.g., the LXY set of integer-length and exponent-length of a numeric, where the "meaningless" X=0 might allow an alphanumeric of length Y) and the numeric base (decimal, hex or whatever). There would also be error-checking bytes to guard against imbalance of DO's and LOOP's, or imbalances of stack pulls and pushes. Also, a set of addresses, like stack pointers and dictionary entry-points. Some unused space should be left to allow for expansion.

The options available to the adventurous are the eternal ones: whether to add improvements to an antiquated but time-tested structure (which may be what FORTHians mean by "historical continuity"), or to try a costlier major reconstruction, or to take the great risk of creating something possibly more grandiose that no one will ever use. Successive FORTRAN's (I, II, etc.) have stressed compatibility with the ancestral form, making it the most conservative of all HLL's and compelling the creation of new ones. The FORTHian "standardizers" fear the Tower-of-Babel effect of user-extensibility. If they steer FORTH too firmly, to avoid Scylla, they may be heading for the Charybdis of an independent FORTH-like HLL, probably far more formidable than PARFOR! Who can foresee the limits of human creativity? ■■



H.T. Gordon is a long-time contributor and friend of *DDJ*. He acquired a KIM-1 in 1976, and his first micro output was a note in *DDJ* #12. His work since then has led many to consider him expert, an honorific he disclaims, because to his way of thinking the trade-off for expertise is loss of breadth.

Now sixty-two years of age and a familiar denizen in the academic towers of Harvard and Berkeley, Hal has dabbled in several of the sciences (especially biology, though he resists being called a specialist) and in numerous non-scientific fields, including history and philosophy.

His Zen-like inclinations tell him that too much knowledge is a dangerous thing, and he applauds the Delphic "nothing too much." On the computer scene, he sees overemphasis on differences among CPU's, hardware systems and languages as an undermining of the underlying unity which exists in the world of operations and their algorithms. Similarities, even among human beings, are more vital than are differences. That is one of the motivations for this current move up to the HLL level.

Typifying Hal's outlook is a sentence from his 1947 Harvard doctoral thesis in neuro-physiology. *"The mightiest poem, symphony, theory, machine – these are less admirable than the mechanism that designed them: the human brain."* ■■

*DDJ's* responsive readership
can give you invaluable feedback
on your ideas and innovations.
Find out how you can reach them
by contacting us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# Sinclair's ZX80

## *Review of a Very, Very Small Computer*

by John B. Palmer

A few days ago, I received a Sinclair ZX80 in the mail. I took it out of the box and got it to work after finding a portable TV that would tolerate the Channel 2 output of the built-in VHF video modulator. I was pleased with the ZX80's overall performance, but I was disappointed with the quality of the video display.

Before I give a rundown of all my likes and dislikes, let me first review what the Sinclair ZX80 is and what it is not. I hope this will aid others who may be thinking about the purchase of a personal computer in the $200 price range.

The ZX80, as you might guess, is built around a Z-80 microprocessor. It has a built-in 4K integer BASIC and 1K of RAM for programs you write. A very tiny keyboard is part of the 7 inch by 8-1/2 inch (about 21 mm by 19 mm) plastic wafer that encloses everything except the power supply. Connections for recorder, TV and power supply are on the backside, where the wafer thickens to 1-1/2 inches (3-1/2 mm).

As I mentioned earlier, the ZX80 comes with a VHF video modulator built-in and preset to Channel 2 for use in North or South America. (The ZX80 is made in England.) You also get a nice, 128 page operating manual that explains how to install the unit and how to program in Sinclair's version of BASIC.

The 9-volt, external power supply has a long cord and looks very much like the units you often see with calculators and transistor radios. Of course, the unit I received was for use on 120 volt, 60 Hz., and it has a "UL" label on it.

Now here are some things you do not get for $200. You do not get a TV set nor any kind of display that would allow you to use the ZX80 without a TV set. Neither do you get a cassette recorder. I tried it on a well-used Hitachi portable TV and a worn out Sony cassette. Both seem to work satisfactorily.

Sinclair does not include any cassette programs with the unit. They say they will provide such in the future. For now, the user must do his own programming. There are no built-in games or pictures, only primitive, low-resolution graphics.

An expansion interface is provided by a nearly standard edge connector. A 50-pin, flat cable card edge connector, like the ones that fit a Shugart 8-inch drive, can be made to fit the Sinclair ZX80 after some minor surgery with a sharp knife. Sinclair says they will sell additional memory in the future. There is no provision for the user to add memory himself. There is not even a wire diagram for the unit.

Now let's get to the details. Overall, I like the ZX80, but it has some real weak spots. I will give my harsh comments first and save the praise for last. Here is how I rate Sinclair's ZX80, using the following scale:

| | |
|---|---|
| Video display | Give it a 1, Very poor |
| Keyboard | Give it a 2, Weak |
| Expandability | Give it a 2, Weak |
| Price | Give it a 3, Good |
| Recorder interface | Give it a 3, Good |
| Graphics | Give it a 3, Good |
| CPU | Give it a 4, Very good |
| BASIC | Give it a 4, Very good |
| Editor | Give it a 5, Excellent |

Video Display: why did I give it such a low score? Because I really had a hard time getting a picture I could read. Sinclair warns you on page eleven of the manual that you have to try tuning the TV set and adjusting the contrast and brightness controls. When I first got the unit I was in Silicon Valley, where the airways are full of all kinds of VHF activity. I just could not keep out interference.

Either the ZX80 does not have enough output or else it can not reject common mode signals that come via the power supply. I found that the display would improve when I used a long extension cord for the power supply. There was no provision for using an alternate UHF or VHF channel, so I dug into the little tin box inside of the ZX80. I found the air-wound coil and spread it open until I could tune in the display on Channel 3. That did help, but not enough. We have a cable system here and every channel in the low VHF is used.

Even if Sinclair would provide a VHF/UHF option, I suspect there would still be trouble with the display. The bandwidth of the modulator seems to be too wide for the low-cost, black-and-white TV set that I wish to use. If I try to tune to the very edge of the channel to increase the high frequency detail, I get a lot of trash that appears to come from the ZX80. I can not see buying a better TV set just to correct this problem. Perhaps a bandwidth compromise could help. (Take a look at Lancaster's TV Cookbook.)

Now about the keyboard: it is too small. I can not get four fingers of my left hand over the A, S, D, and F keys. The forty keys are not keys at all. A flat surface with the keypads marked in blue, it is impossible to develop a touch for the keys. You have to revert to typing with one finger. (One good thing, the BASIC allows single-key entries.) And the internal code is not ASCII (it is a 6-bit alphanumeric code that may be converted to a subset of ASCII). This means you can not use another keyboard, or a printer, without quite a bit of work. Most of the raw information about the internal code is documented in the user's manual. What is needed now are some good, detailed examples of how to enjoy the ZX80 without tearing it apart (which is fun!).

The ZX80 has an attractive price. To my knowledge it is the lowest cost, assembled and tested unit that provides a TV display and BASIC programming. Here in the United States, there are few kits under two hundred dollars that promise as much as the Sinclair. Personally, I strongly recommend that any novice not try to build such kits unless the kit-building is the objective. Will the price of the ZX80 come down? I believe that Sinclair will be able to lower the price of the present ZX80 to about $150 and then introduce a better model for about $250 sometime in 1981.

The cassette interface seems to work quite well. The operating manual says that your recorder must provide 4 volts, peak to peak, to fully drive the input during playback. Recording is not much of a problem, just read the instructions. During playback of your program, you must adjust the volume control until you find the best setting. This may take a few tries. Once you find the right setting, leave it.

My old Sony worked alright on the second try. I believe that is very remarkable. It took me a week to get my APPLE cassette going, and I have heard of some TRS-80 owners who are still trying to adjust the playback volume! With the ZX80 it is quite simple. Turn it up as loud as it will go and then back off just a bit. The video display gives some idea of what is going on. If the display remains blank after playing the tape beyond the area where you recorded data, just rewind

and try again with more volume.

If you get a ZX80, do not expect it to work with tapes made by a friend who has a different personal computer. In fact, you may find that your ZX80 can not read tapes made by another ZX80. This is true of most personal computers and is not a problem unique to the ZX80. Quite a while ago, many users got together in Kansas City and came up with a reliable tape system. But the most popular brands of small computers ignore the standard. This makes me mad! Why can I not trade tapes with my friends just because they have brands X and Y, and I have Z?

The graphics are good. In fact, they are better than some more costly systems. Any letter or symbol on the keyboard can be displayed in reverse video by means of the CHR$ function. And there are twenty additional shapes that are to be used in making bar graphs, squares or highlights. The low-resolution graphics are satisfactory. Too bad the letters were not better.

The Z-80 CPU is possibly the most popular among hobbyists. Sinclair's BASIC allows you to write machine code using POKE. The USR function allows BASIC to call the machine code. A parameter is sent and returned in the HL register pair. The PEEK function allows access to special system variables or any part of memory. The 1K RAM is from 16384 thru 17407. All notation must be in decimal in the BASIC text. (The ZX80 could be used to learn about Z-80 code, but it does not have a system monitor other than the BASIC interpreter. You could write your own and save it on cassette.)

The CPU does the whole show. There is no video display chip. While the CPU is waiting for the user to enter something on the keyboard, it is updating the television screen 60 times a second (or 50 times a second in Europe). This is the secret of how Sinclair can offer so much in a small package. Perhaps you may sense that I am partial to the Z-80. I think that Sinclair has done a good job of using the advantages of the Z-80 CPU.

Sinclair BASIC is very good. It reminds me of Tiny BASIC, but has many more features that make the ZX80 an enjoyable personal computer. All of the standard GOTO, GOSUB, IF THEN, FOR NEXT, PRINT, and other BASIC commands are present. To save space in the 1K RAM area and to reduce the programmer's work, all principle BASIC keywords are entered with a single keystroke. Many of the single-stroke keywords appear on the same keypads as do the letters and numerals. How does BASIC know if you want to type in a keyword or something else? It is automatic. Some other BASIC words are known as "tokens" rather than"keywords." These are entered with a shift. There are eight lesser-used functions that must be given letter by letter.

This BASIC is not for any operation that requires decimal-point arithmetic. Sinclair's BASIC will seldom give the right answer to a division problem. There is a good explanation for this in the operating manual, and a program listing that shows how to correct the problem. It would have been nice if the remainder from division had been stored into a special variable.

Strings may be tested and altered with some novel functions. On pages 76 and 77 is a table showing all the codes for ZX80-BASIC. Page 77 explains the use of CHR$( ), CODE$( ) and TL$( ). With this information, you can have many enjoyable hours making up nonsensical programs that display cryptic messages. This is far more important than decimal point arithmetic. After all, the real reasons anyone would buy a low-cost personal computer should not include anything very serious. I work all day doing serious things. With my new pastime, I can hide away and write programs that serve the noble task of bringing me pleasure and relaxation.

What makes this BASIC outstanding is the operating manual. Not too big. Not too little. Just right to go with a computer you can carry in your lunch pail with room left for a sandwich and a banana! There is enough information that a novice could learn to program using the operating manual and his Sinclair. For the more advanced, there is enough detail about how the system works to allow you to try some fancy tricks. There is a program example of how to use the frame refresh to measure reaction time in milliseconds! (No, they do not give ROM source code – sigh.)

Now the best part: the text editor. I did not expect a low-cost personal computer, with the features already described, to have a text editor. Without bothering to read the operating manual all the way through (which is perfectly all right, if you at least browse through it), I noticed four little arrows at the top of the keyboard. I tried one and found, to my surprise, that what happened was perfectly natural. The back arrow allows me to go back and insert more text. The forward arrow allows me to move forward in the text. The rubout will rubout the character to the right of the cursor.

Compare the above with some of the very popular word processor hardware and software. I used a very expensive, professional Data Entry Terminal to write this review. Over half of the special keys do not work in a logical manner. My big terminal also has four cursor controls. Only one works almost right. You have to learn a bunch of control codes just to do something logical. At work, I spend several hours working on another system that is alien to the one I use at home. And I have to spend some time each day explaining the text editors to newer employees.

A simple and natural cursor movement text editor in the Sinclair was far beyond expectations. Now, I do not mean to say that is is a full text editor. I mean that it is beyond what is often seen in personal computers up to the one thousand dollar price range.

The line currently being written or edited is at the bottom of the display. When you get to the end of the line, you just hit the key called "newline." It is not called "return." It is not called "enter." It is called "newline," Hit it and your line finds its place in the BASIC list, in the upper part of the video display. The cursor goes to the left of the now clear bottom line and displays an inverted "K" to let you know that the BASIC parser now expects a single-stroke keyword. Hit a single-letter keyword, it appears instantly, and the cursor changes to an inverted "L" to let you know that the parser now expects an argument to the keyword. If you enter an invalid argument, the cursor changes to an inverted "S" to tell you that your syntax is not correct at this point. Move the cursor back, and it becomes an inverted "L" as soon as you pass to the right of the character that caused the syntax error. In some cases, the "S" will appear until you have fully formed an expression. There is no way you will forget the final, left parenthesis!

What about lines you have already entered? Any line you have already entered does not have any syntax errors. The editor will not allow you to hit the

*(continued on page 43)*

# POKE and PEEK to Your Heart's Content

**An enhancement to Microsoft BASIC Version 3.2 (8K version) to enable PEEK and POKE to any location in memory.**

For four years now I have been running MITS ALTAIR BASIC Version 3.2 (Microsoft 8K version) with no major complaints. However, I recently expanded my ALTAIR 8800's memory beyond the 32K point which created a problem with the PEEK and POKE functions.

In this early version of Microsoft BASIC, which many of us are still using, these two functions work only with addresses of less than 32768 (32K). Calling these functions with arguments greater than 32767 or with negative arguments (the convention for larger versions of BASIC) results in a FC (Function Call) error. I found myself unable to relocate my Polymorphics VTI screen memory address at the new Top of Memory so as to give BASIC access to the expanded memory and still be able to update points on the screen directly from BASIC using PEEK and POKE.

Two long days of running BASIC under a simulator allowed me to locate and analyze the subroutines which control the behavior of PEEK and POKE. Another day with the simulator and I had the patches described here developed, tested and installed in BASIC allowing PEEK and POKE to all memory.

To implement this enhancement, an eleven-byte subroutine must be added to BASIC and two Calls must be altered within it. I added the required subroutine immediately following the trig functions. Using this location will necessitate the alteration of BASIC's initialization routine to set the start of program memory after the subroutine and to eliminate the start-up option to delete the trig functions. An alternative would be to locate the subroutine elsewhere, near the Top of Memory.

Table I presents the new subroutine and the patched subroutine Calls as well as the reentry point and the bypassed subroutines. The addresses given are from my Paper Tape version of BASIC. Other versions may have these subroutines at different addresses, but my experience has shown that the variation will likely be less than 1K. The full subroutines are given to allow you to find them easily in your version.

With these changes installed, you will be able to POKE or PEEK anywhere within the direct addressing area, using either positive or negative addresses up to 65535. That is to say, you may use either convention to address any location, -5 will point to the same location as 65531 and +5 will point to the same location as -65531.

So patch it in and POKE and PEEK to your heart's content!

■■

**TABLE I**

```
0FF7  PEEK    CALL   SR01+3     ;Changed from L0638
0FFA          LDAX   D
0FFB          JMP    L0CB9
0FFE  POKE    CALL   SR01       ;Changed from L0635
1001          PUSH   D
1002          RST    1
1003          DB     2CH        ;','
1004          CALL   L0FCD
1007          POP    D
1008          STAX   D
1009          RET

0634  L0634   RST    2
0635  L0635   CALL   L0960     ;Bypassed in POKE
0638  L0638   RST    5          ;Bypassed in PEEK & POKE
0639          JM     L064F      ;
063C  L063C   LDA    L0242      ;
063F          CPI    90H        ;
0641          JC     L1301      ;Reentry from new subroutine
0644          LXI    B,9080H
0647          LXI    D,0000H
064A          CALL   L12D6
064D          MOV    D,C
064E          RZ
064F  L064F   MVI    E,08H
0651          JMP    L02C6

16C8  SR01    CALL   L0960     ;New Subroutine to enable
16CB          LDA    L0242      ;POKE and PEEK to all
16CE          CPI    91H        ;directly addressable memory.
16D0          JMP    L9641

                     END
```


## by T. J. Van Iderstine

*T. J. Van Iderstine, Box 5407, Kreole Station, Moss Point, MS 39563.*

# PAN

## *One Activity of the PCNET Project*

**The Personal Computer NETwork had its inception at the First West Coast Computer Faire. It marked the beginning of today's international movement toward computer networks. PCNET is the frontrunner in a field still largely dominated by Computer Bulletin Boards and other relatively simple systems.**

The PCNET Project is conducting three principal activities. The most significant is the Personal Computer NETwork (PCNET) "protocol," a method by which any personal computer can telephone a text message to any other personal computer automatically, detecting and correcting any errors that might occur because of the noisiness of the telephone system. We are designing and programming one or more "model" implementations, and we will conduct a large workshop in which users of various personal computers will recode a model version into a useful language for their own particular hardware. The model and all versions from the workshop will be distributed without copyright, so that anyone will be able to use the PCNET protocol directly or as the basis for other programs which need reliable communication. This activity is now being coordinated by David Harris.

The second activity, coordinated by Pat Cleland, is the "Electric Phone Book," a list of Bulletin Board Systems believed to be in operation. There systems are computers programmed to receive phone calls and manage the reading and leaving of text messages for other people. Like a public bulletin board, anyone can read most of the messages posted, and anyone can post them. To encourage use by as many people as possible, the PCNET Committee is arranging to have these systems' phone numbers published in *Recreational Computing*. Incidentally, the PCNET Committee itself operates a Bulletin Board System, on an Apple computer.

**by Dave Caulkins and David C. Harris**

The third activity of the PCNET Project is the PAN electronic mail system, which we started distributing in early 1980. PAN lacks the power of the full PCNET protocol in terms of error control and data transparency, but is quite suitable for short, natural language messages. A more detailed description of PAN can be found in the June/July '80 issue of *DDJ*.

We and the TNW Corporation (our associate in developing PAN) have decided that licenses for the PET version of PAN will no longer be necessary. We plan to put a complete listing of PET PAN into an upcoming issue of *DDJ*.

We're making some changes in PAN for a number of reasons as described below. If you have a PAN system and wish to be able to conform to the PAN standard for communication with present and future versions of PAN, please make the indicated changes.

1) To make PAN conform to the PCNET standard serial character format (1 start bit, 8 data bits, 1 stop bit and no parity):

   change CHR$(145) to CHR$(154) in line 1030

2) If you have purchased a TNW488 modem recently, it may be set up for device numbers 16 and 17. If this is the case, change line 1028 to read as follows:

```
1028 OPENR,16,0:OPENS,17,0:
     OPENT,16,1:OPENU,17,1
```

3) Some computers for which PAN is being implemented dedicate the ← character to special purposes. For this reason the PAN end-of-message character is being changed from ← to ↑. Make changes as follows:

   In line 3001 change ← to ↑.

   Change line 3116 to read:

```
3116 IFA$="↑"THENX1=-1
```

   Change 95 to 94 in line 3247 (1 place), in line 3248 (2 places), and in line 3250 (1 place).

4) Some computers for which PAN is being implemented require longer delays for some of their operations. To accommodate these machines some PAN timeouts have been lengthened. Change lines 7740 and 8000 as follows:

```
7704 IFTI-V<1800GOTO7702
8000 V=TI+1800:I=1
```

5) PAN version numbers are of the form ANN where the A is a single alphabetic character; present assignments are P for PET, A for Apple and X for experimental version. Change line 9467 to the new version number as follows:

```
9467 L$="@@@P10"
```

6) The previous version of PAN (P09) did not handle the cursor properly during message entry on PETs with new ROMs. Change line 13000 to read as follows:

```
13000 A=224+PEEK(50003)*-28
```

   Add new line 13001:

```
13001 CP=PEEK(A)+PEEK(A+1)*
      256+PEEK(A+2)
```

The present version of PAN has one outstanding defect: the receiving PAN does not acknowledge receipt of a complete message. We intend to change the PAN protocol to provide the sending PAN with a positive acknowledgement from the receiving PAN that the entire message has been accepted. This change will be documented in an upcoming issue of *DDJ*.

Robert Maas pointed out that the PAN protocol that appeared in the June/July issue of *DDJ* was not as detailed as it should have been. The following revised version should be better. Note that a PAN message can be sent manually with only a terminal and modem, because the protocol is so simple.

At the hardware level, PAN conforms to the Bell 103A standard operating at 300 baud. The serial character format is 1 start bit, 8 data bits and 1 stop bit. PAN does not use parity; it is desirable but not required for the transmitting PAN to set the parity bit to the mark state. PAN is designed for use with upper case characters. Some PAN versions will accept lower case characters, but some will not; unless the capabilities of the target PAN are known, it is safer to use upper case characters only. The UP ARROW character (↑) is reserved for use as the end-of-message character and should not be used as part of the message text.

*(continued on page 37)*

# DECONCAT

## *A De-Bugging Tool for Concatenated Programs*


## by Greg Smith

*GREG SMITH, Commonwealth Department of Transport, Ground Facilities Division, P.O. Box 367, Canberra 2601, Australia.*


In order to save precious bytes and running time, program statements in several BASIC's can be concatenated onto one line. In North Star BASIC, statements on the same line are separated by a backslash (\).

For example:

```
10 FILL 10466,3\IF Z= 3 THEN 20
   \END
```

There are programs on the market such as DOC (from Mini Business Systems) which concatenate BASIC programs. Figure 1 is the listing of a sample program after it has been concatenated by DOC.

One disadvantage of having concatenated programs is trying to debug or make changes. This is the reason that DECONCAT was written. DECONCAT is a program which will split concatenated program lines into their component statements, making each statement a separate line.

There are three simple steps to follow when using DECONCAT:

1) Renumber the original program.
2) Create a file into which the new (deconcatenated) program can be written.
3) Run the program, supplying the name of the program to be deconcatenated and the name of the new program file.

Figure 2 is the listing of the sample program after it has been run through DECONCAT. DECONCAT also checks to ensure there is a blank between the line number and the first character in the statement, which makes the program much easier to read (Figure 3).

**How it works.**

A North Star BASIC program line is written onto file in the form:

A,B,C,STATEMENT,<CR>

A is the byte containing the number of bytes in the line, while B and C are the bytes which contain the line number in the form:

LINE NUMBER=Cx256+B

When a line is concatenated, it is represented by:

A,B,C,STATEMENT\STATEMENT\
. . . etc. . . .\<CR>

DECONCAT operates by reading a program line and searching for a backslash. If no backslash is found then the line remains unchanged. When a backslash is found the line is split. The first part of the line retains the original line number (thus preserving GOTO's, etc.) and the remainder of the line is assigned a line number one greater than the original line (hence the need to renumber the original program before attempting to deconcatenate it). ■■

## Listing

```
LISTING

LOAD DECONCAT
READY
LIST 10,490

10 REM . . . . . .DECONCAT. . . . . .
20 REM . . . . . 7/10/79
30 REM . . . . . (C)  G. L. SMITH
40 REM . . . . . .    4 WILLCOTT STREET
50 REM . . . . . .    GEEBUNG,
60 REM . . . . . .    BRISBANE,
70 REM . . . . . .    QUEENSLAND, 4034
80 REM . . . . . .    AUSTRALIA
90 DIMA9 (300)
100 INPUT"SOURCE FILE NAME ? ",W$
110 INPUT"FILE TYPE ? ",W
120 INPUT"TARGET FILE NAME ? ",W1$
130 INPUT"FILE TYPE ? ",W1
140 !
150 !"OLD LINE  NEW LINE"
160 OPEN#0%W,W$
170 OPEN#1%W1,W1$
180 REM. . .READ LINE LENGTH (A) AND LINE NO. BYTES (B,C)
190 READ#0,&A,&B ,&C
200 IF A=1 THEN 300
210 REM. . .CALCULATE LINE NO. (E)
220 E=C*256+B
230 !E ,
240 REM. . .READ LINE
250 FOR I=1 TO A-3
260 READ#0,&A9(I)
270 NEXTI
280 GOTO 340
290 REM. . .END OF PROGRAM
300 WRITE#1,&A
310 CLOSE#1
320 CLOSE#0
330 END
340 M=0
350 D=1
360 Q=1
370 REM. . .CHECK FOR LINE NUMBERS IN STATEMENTS (154)
380 FOR L=D TO A-3
390 IF A9(L)=154 THEN L=L+2 ELSE 420
400 NEXTL
410 REM. . .CHECK FOR INVERTED COMMAS (34)
420 IF A9(L)<>34 THEN 450
430 IF S=1 THEN S=0 ELSE S=1
440 REM. . .CHECK FOR END OF INVERTED COMMAS
450 IF S=1 THEN 480
460 REM. . .CHECK FOR BACKSLASH (92)
470 IF A9(L)=92 THEN GOSUB 550
480 NEXTL
490 L=L-1
READY
```

```
LIST 500,

500 IF M=1 THEN A=L-Q+4
510 Z=1
520 GOSUB 600
530 Z=0
540 GOTO 190
550 M=1
560 A9(L)=13
570 REM. . .CALCULATE STATEMENT LENGTH (A)
580 A=L-Q+4
590 REM. . .INSERT SPACE (32) AFTER LINE NO.
600 IF A9(Q)=32 THEN 630 ELSE A=A+1
610 WRITE#1,&A,&B ,&C ,&32,NOENDMARK
620 GOTO 640
630 WRITE#1,&A,&B ,&C ,NOENDMARK
640 FOR F=Q TO L
650 WRITE#1,&A9(F),NOENDMARK
660 NEXTF
670 !TAB(10),E
680 IF Z=1 THEN RETURN
690 REM. . .CALCULATE NEW LINE NO. BYTES (B,C)
700 E=E+1
710 C=INT(E/256)
720 B=INT((E/256-C)*256+.1)
730 Q=L+1
740 RETURN
READY
```

**end**

## FIGURE 1

```
10 REM EXAMPLE PROGRAM\!"DECONCAT EXAMPLE PROGRAM"
50 !"DECONCAT WILL NOT MISTAKE A BACKSLASH (\), IN A PRINT"
90 !"STATEMENT SUCH AS THIS, FOR A NEW STATEMENT \\\\\\\\"
130 REM INPUT PHASE\OPEN#1,"DATA"\READ#1,J,K
170 FOR J=1 TO K\IF J=23 THEN !"FOUND" ELSE 250
210 IF J=45 !"LOST IT"\NEXTJ
250 REM END OF SEARCH\IF J<>23 THEN !"THE END" ELSE 290\
    GOTO 330
290 !"J= ",J
330 END
370 REM. . . . . .
410 REM DECONCAT WILL HANDLE ANY NUMBER OF STATEMENTS
    IN THE ONE LINE e.g.
450 REM\REM\REM\REM\REM\REM\REM\REM\REM\REM\REM\
    REM\REM\REM\REM\REM\REM\REM\REM\REM\REM\
    REM\REM\REM\REM\REM\REM\REM\REM\REM\REM
490 REM. . . . . .
530 REM DECONCAT IS NOT CONFUSED BY NUMBERS IN
    STATEMENTS
570 REM HAVING THE SAVE VALUE AS THE ASCII CHARACTER
    VALUES
610 REM FOR WHICH IT SEARCHES e.g.
650 !34/92*154\GOTO 32
690 ON D GOTO 92,154,34,92,32
```

## FIGURE 2

```
10 REM EXAMPLE PROGRAM
11 !"DECONCAT EXAMPLE PROGRAM"
50 !"DECONCAT WILL NOT MISTAKE A BACKSLASH (\), IN A PRINT"
90 !"STATEMENT SUCH AS THIS, FOR A NEW STATEMENT \\\\\\\\"
130 REM INPUT PHASE
131 OPEN#1,"DATA"
132 READ#1,J,K
170 FOR J=1 TO K
171 IF J=23 THEN !"FOUND" ELSE 250
210 IF J=45 !"LOST IT"
211 NEXT J
250 REM END OF SEARCH
251 IF J<>23 THEN !"THE END" ELSE 290
252 GOTO 330
290 !"J= ",J
330 END
370 REM. . . . . .
410 REM DECONCAT WILL HANDLE ANY NUMBER OF STATEMENTS
    IN THE ONE LINE e.g.
450 REM
451 REM
452 REM
453 REM
454 REM
455 REM
456 REM
457 REM
458 REM
459 REM
460 REM
461 REM
462 REM
463 REM
464 REM
465 REM
466 REM
467 REM
468 REM
469 REM
470 REM
471 REM
472 REM
473 REM
474 REM
475 REM
476 REM
477 REM
478 REM
479 REM
480 REM
481 REM
490 REM. . . . . .
530 REM DECONCAT IS NOT CONFUSED BY NUMBERS IN
    STATEMENTS
570 REM HAVING THE SAVE VALUE AS THE ASCII CHARACTER
    VALUES
610 REM FOR WHICH IT SEARCHES e.g.
650 !34/92*154
651 GOTO 32
690 ON D GOTO 92,154,34,92,32
```

# Small~C

## *An Implementor's Notes and a Bug Corrected*

by P. L. Woods

*P. L. WOODS, 78 Dukes Avenue, Muswell Hill, London N10 2QA, England.*

**Abstract**

The author describes his experience of implementing the small C compiler[2] in FORTRAN as a cross-compiler running on a large minicomputer, and describes a bug detected during the implementation. The bug permits an incorrect interaction between a function argument list and the type declarations for those arguments, resulting in incorrect code being generated for references to function arguments. Corrective code is presented.

**Introduction**

The May 1980 issue of *Dr. Dobb's Journal* could have been written just for me. I have built a home-brewed, Z-80 based computer system, and am now looking for reasonably priced software to run on it. After about a year of hand assembling programs, and then loading hexadecimal numbers directly into memory via a VDU, I came across a local supplier of a Z-80 assembler. This greatly eased the programming burden, but as I have tried Pascal and currently use FORTRAN professionally, I knew that something was lacking, namely a high level language on my micro.

Pascal seems widely available, and a compiler for a subset of this has been published[1], but my earlier experiences with this language had convinced me that while it achieves its design aims, as a practical language capable of supporting portable programs it leaves much to be desired. So I decided against Pascal.

FORTRAN was a possibility, but is not readily available on a home system. This language has many critics, but these are more than outweighed by the number of successful compilers for the language, and the wealth of available programs. Further, once written, a compiler for a subset of the language would be useful to those in situations much the same as mine, and those prepared to make do with the inadequacies of the language in exchange for something widely available. I thus started to consider the design of a FORTRAN compiler to generate relocatable machine code for a Z-80 processor, and coded some initial routines.

That was until I came across Ron Cain's article presenting a small C compiler in its entirety[2], to be followed a short time later by the run time system for his program[3]. I had read *The C Programming Language*[4], which is the definitive document for the full language, and also for tiny-c[5]. A comparison between the facilities proposed for my sub-FORTRAN compiler and those implemented by Mr. Cain showed that, if anything, his compiler was more extensive than mine would have been. This, together with the fact that his program was a finished product while mine was still at the design stage, led to only one conclusion: terminate work on the FORTRAN project and set about implementing the small C compiler.

**Compiling the Compiler**

The first step to implement the small C compiler would be to create a machine-readable copy of the source code. That is easily solved, and would merely require a lot of typing and careful checking on my part. But that would have left me with a small C compiler written in a language for which I did not yet have a compiler: namely small C! I do not have access to a C compiler, so I started to look for other ways of compiling the compiler.

There are two solutions to this class of problem. One is to hand-code the program into assembly language. This is impractical and defeats the object of the exercise. The other solution is to translate the troublesome program into a language for which one does have a compiler, and ensure that the translation is so easy that it may be performed without much mental effort (I am lazy). I inspected the small C compiler for possible snags, and found a further restriction. The chosen language must support re-entrant functions; that is, functions must be able to call themselves.

I did, however, have access to a computer which had a FORTRAN compiler and while FORTRAN is normally regarded as not permitting recursive subroutines, the author of this particular product had been especially liberal-minded, and had added this feature as an option to his compiler. My problem was solved. I would translate the small C compiler to FORTRAN and then compile, using the re-entrant code option.

This meant I would effectively end up with a cross-compiler for small C, a program which runs on one computer and produces code for a second. That would be better than nothing, for I could then feed small C (written in small C) into my cross compiler and get an 8080-executable version. I would still be faced with the problem of typing the assembler version of small C into my home computer, but at least I would not have the daunting prospect of hand coding the compiler. In the meantime, I could cross-compile short small C programs and take them home to test them.

**Implementation Difficulties**

The actual building of the cross-compiler went very smoothly, with few problems. These mainly concerned C and FORTRAN incompatibilities. In C, the first element of an array is number [0], while in FORTRAN it is [1]. This was solved by simply adding 1 to each array reference where it occurred.

Also, as FORTRAN does not have a pointer type, all pointers had to be changed to either array indices or array elements, depending on the context. Deciding which, and then which array to subscript, was sometimes a little difficult.

Although FORTRAN is strong on its scientific variable types (DOUBLE PRECISION and COMPLEX), it is lacking in the area of a character variable type. Thus, all character types had to be converted to integers and this unfortunately meant that certain C constants such as '[' become meaningless numbers like 91. Because of difficulties with the character set, I also decided that the FORTRAN version of the compiler would accept upper-case reserved words only ("IF" instead of "if"), even though the preferred style is to use lower case.

Another problem was the FORTRAN programmers' usual nightmare of what to do with string (Hollerith) constants. One solution would have been to have changed all strings to integer arrays set by DATA statements filled in the same format the compiler maintains in its own internally generated strings. This does, however, lack clarity, so I chose to leave the strings alone and accept the overhead of unpacking and repacking strings into integer arrays with subroutine calls. In certain cases (e.g., "astreq"), this meant that I had to have two versions of the function, depending on the origin of the arguments – Hollerith string or symbol table entry.

The final incompatibility was that some of the function names in small C looked very much like FORTRAN key-

words. Although the function CALL should have been safe enough, the same could not have been said of MOD and AND, so these and other similar names were changed by appending a Z.

The last problem encountered was that the liberal-minded author of the compiler had not been as generous as it had first appeared. While his FORTRAN compiler would indeed generate re-entrant code, this was only true for subroutines, not for functions. Thus, all the small C functions that returned a value had to be rewritten as subroutines. This proved not too onerous a task, and merely involved changing function declaration statements to subroutine declarations, and modifying the statements that returned the function values to do this via a subroutine argument.

**Moral**

There is a moral to this part of the story. It is that if you see something in *Dr. Dobb's Journal* which you like, but it is written in a language that your computer will not accept, do not give up hope! Instead, ask around to find someone who has a processor for that language which you might use, or consider rewriting the program in a language which your computer will accept. That will be far less work than re-inventing the silicon chip.

**A Bug Discovered . . .**

My selected course of implementing small C led me to carefully check the sort of code being produced for various small C source statements. Due to my poor typing, this soon revealed a problem. At one point, I had written in a test routine:

```
testsetup (pone, ptwo)
    int ptwo, pone;
    (
    /* and so on . . .  */
```

The code generated did not work. My test source code looked all right, but to make sure, I inspected the source code in the original article to see if I could find any instance of what I was trying to do. (If it worked before, it should work for me.) Then I noticed that Mr. Cain had always been careful to declare the types of his function parameters in exactly the same order as they appeared on the function header statement. This is certainly a good style to adopt, but in my test program I had declared the types of the parameters in the reverse order. This, I thought, should make no difference, and is certainly the case in all other computer languages I have come across. Nevertheless, I started to check what the compiler was doing while it was reading function header lines and then setting the types of the parameters.

As an aid to this, I wrote and compiled a pair of test functions and obtained the output shown in Listing I. The only difference between these functions is the order in which the parameters are typed, but the code generated is clearly different as to the values loaded into the HL register pair, and in the second case is wrong. The reason for the use of upper case is due to the technical difficulties encountered with the FORTRAN version of the small C compiler mentioned above.

My attention was soon centered upon "newfunc" and "getarg" and the bug was revealed. The small C compiler as published[2], uses the function header line only to determine the number of arguments. Their addresses on the stack are set by their order of declaration in the following "int" or "char" statements, not by their order of appearance on the header line.

**. . . and a Bug Corrected**

This was a definite bug in the compiler, so I set forth to fix the error. The approach was clear. Entries for dummy arguments must be inserted into the local symbol table when the parameter list is being read, rather than during the processing of the type declarations. However, only part of the symbol table entry is known for the variable at that point (its name and relative address), so that the rest of the information ("int" or "char", pointer or not), must be patched in at a later date when the "int" or "char" statements are read.

This called for a complete rewrite of the "getarg" function, but only minor changes to "newfunc". A side effect of the changes in "newfunc" was that some code to clear the local variable list had to be moved around, as the original code cleared this array after reading the parameter list, rather than before as required by my modifications. Complete listings of both "newfunc" and "getarg" are presented in Listing II.

As a test of my correction, I reran my trial functions from Listing I. The results are shown in Listing III, and it is clear that the code is now correct. Finally, I retried "testsetup", and that worked too.

**Conclusion**

The result of this work is that I now have a cross-compiler available for my home-brewed Z-80 based home computer system, a prospect that seemed rather distant before I came across the recent editions of *Dr. Dobb's Journal*. I will shortly have the compiler up and running on my own machine, freeing myself from assembly language programming at home.

Corrective code for a nasty little bug is presented here. I hope I have saved someone from sleepless nights while trying to work out why his legal program does not work.

**References**


[1] *The BYTE Book of Pascal*. B. W. Liffick (ed.), Byte Publications, 1980.

[2] *A Small C Compiler for the 8080's*. Ron Cain, *Dr. Dobb's Journal*, #45 (May 1980), pp. 5-19.

[3] *A Run Time Library for the Small C Compiler*. Ron Cain, *Dr. Dobb's Journal*, #48 (September 1980), pp. 4-15.

[4] *The C Programming Language*. B. W. Kernighan & D. M. Ritchie, Prentice Hall, 1978.

[5] *The tiny-c Owner's Manual*. T. Gibson & S. Guthery. tiny-c associates, 1978.


■ ■

(LISTING ON PAGE 32)

## A Very Special THANK YOU

Readers who responded to *DDJ*'s recent survey have greatly assisted us to understand the growing number of ways we can serve you.

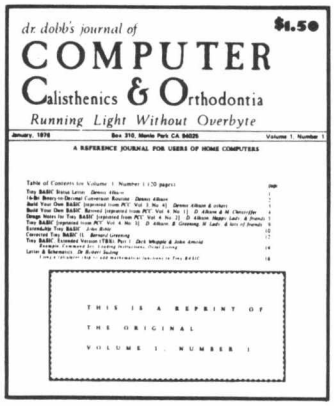
Five of those surveyed were selected at random to receive a small gift of thanks. The first three compiled volumes of *Dr. Dobb's Journal* have been sent to these individuals:

D. W. Daetwyler
Ken Data
Ned Freed
Steve Patchen
Fred W. Pluebell

Since 1976, the most innovative magazine for the technically-sophisticated microcomputer user.

# Dr. Dobb's Journal

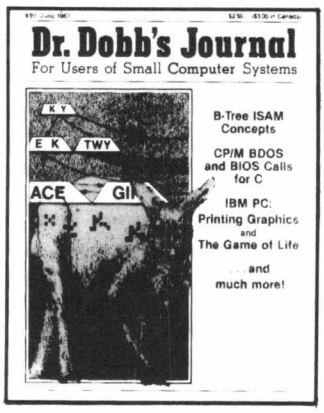
## For Users of Small Computer Systems




1976



1983




Have you been looking for a magazine that will provide you with serious articles on programming? *Dr. Dobb's Journal* can give you the technical depth you need to understand the latest software advances. We cover every microprocessor with articles on C, Forth, Unix, CP/M and much more. Our reputation for editorial quality is well known. Subscribe now, and discover what industry insiders have enjoyed since 1976.

Coming articles will cover:

- new language development
- 8- and 16-bit technology
- operating systems
- compilers, editors
- assemblers/cross-assemblers
- clever algorithms

**$25**

one year subscription
12 issues
saves $5 off the newsstand price

**YES!** Enter my subscription to *Dr. Dobb's Journal* and bill me later.

name

address

city state zip

**Mail to: DDJ, P.O. Box E., Menlo Park, CA 94025**

# Z-80 Memory Test

## by Ray Duncan

*Ray Duncan, Laboratory Microsystems, 4147 Beethoven Street, Los Angeles, CA 90066.*

**General Description**

This Z-80 Memory Test for CP/M-based systems is modeled after the exhaustive diagnostic programs used by minicomputer memory add-on manufacturers. Mode and area of memory tested are controlled from the keyboard.* Failing locations are isolated to the bit level and audited on the console. Errors in mapping the logical address space to the physical memory are trapped by calculating and writing the entire data pattern before reading any locations.

**Configuring test for your computer**

All console input and output is done by a set of subroutines which access the four configuration bytes at 0103H-0106H. You may easily customize the test for a different serial I/O board by reading in the HEX or COM file with DDT or DEBUG, modifying these four bytes, and SAVEing a new COM file (see Fig. 1).

It is recommended that you do NOT modify the actual input and output routines "Ci," "Ciw," and "Co."

**Using the test program**

After booting up your system, type:

A>Z80MT

The test program will load into memory and print the title and version number:

Laboratory Microsystems
Z-80 Memory Test version 2.3

The program will then request the test mode:

I=itemize errors, T=print error total only, E=exit test
>

You must select I, T, or E. Selections have the following consequences:

I = normal mode of memory test, all memory errors found are itemized. This mode should be used when soft or intermittent errors are suspected.

T = print total errors per pass only, do not itemize individual errors. This mode may be used to avoid long printouts when you have a known failing location.

E = exit test – if the test program determines that the operating system was not destroyed during the testing process, a "warm boot" is executed. Otherwise, the test prints a warning message and loops waiting for manual restart of the computer system.

The program will now request that you specify the area of memory to be tested:

Enter address of the first byte of memory to test:
>
Enter address of the last byte of memory to test:
>

The addresses you enter must be four hexadecimal digits. For example, legal entries would be 1000, 2FFF, 8000, CAAA. The test program protects itself from destruction; if you enter an address that falls within the boundary of the test program, it will reset that address to the end of the program and print a warning message.

After acceptance of the last byte address to test, the program automatically begins executing. You can interrupt execution of the test at any time by depressing any key, however response may take several seconds when testing large areas of memory.

For each error detected, an audit is printed on the console in the form (all numbers are in hexadecimal, see Fig. 2).

*(continued on page 35)*

**Listing begins on page 24**

**Figure 1.**

| Memory Address | In Listing As | Significance |
|---|---|---|
| 0103 | 00 | Address of terminal status port |
| 0104 | 01 | Address of terminal data port |
| 0105 | 40 | Data ready flag (RDA) |
| 0106 | 80 | Transmit buffer empty flag (TBE) |

**Figure 2.**

Address=xxxx Pattern=xx Contents=xx Xor=xx Type=xx

| | |
|---|---|
| Address: | Memory address where error detected |
| Pattern: | Data that was written to that location in memory |
| Contents: | Data that was read back from that location in memory |
| Xor: | Logical exclusive OR of pattern and contents, isolates the failing bits(s) |
| Type: | Test makes best guess of whether error detected is hard (write) or soft (read) |

**Figure 3.**

Pass=xxxx Errors=xxxx Cum. errors=xxxx And=xxxx Or=xxxx

| | |
|---|---|
| Pass: | Number of passes executed since test began |
| Errors: | Total errors this pass |
| Cum. errors: | Total errors since test began |
| And: | Logical AND of all failing memory addresses since test began (i.e., gives bits in common to the addresses of all failing locations) |
| Or: | Logical inclusive OR of addresses of all failing memory locations (i.e., gives the union of all bits in the addresses of failing locations) |

** Because the memory test may destroy the CP/M monitor, console I/O is done directly.*

# Z-80 Memory
## Listing

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15
'Z-80 Memory Test'

                        0002 ;
                        0003 ; Memory Diagnostic for Z-80
                        0004 ;
                        0005 ; Version 2.3  1 Sept 1980
                        0006 ;
                        0007 ; Copyright 1980 by
                        0008 ; Ray Duncan
                        0009 ; Laboratory Microsystems
                        0010 ; 4147 Beethoven Street
                        0011 ; Los Angeles, CA 90066
                        0012 ;
                        0013 ; The purchaser may freely create
                        0014 ; any number of copies of this program
                        0015 ; on machine readable media in order
                        0016 ; to support his computer system(s).
                        0017 ; Further resale of this program
                        0018 ; is prohibited.
                        0019 ;
                        0020 ;
0000                    0021          Org     100H
                        0022 ;
      (0100)            0023 Stack    equ     $
      (0000)            0024 CPM      equ     0
      (0000)            0025 Tstat    equ     0       ;terminal status port
      (0001)            0026 Tdata    equ     1       ;terminal data port
      (0040)            0027 Rda      equ     40H     ;input data ready flag
      (0080)            0028 Tbe      equ     80H     ;transmit buffer empty
      (001B)            0029 Esc      equ     1BH     ;ASCII escape code
      (000D)            0030 Cr       equ     0DH     ;ASCII carriage return
      (000A)            0031 Lf       equ     0AH     ;ASCII line feed
                        0032 ;
      (0003)            0033 Rcnt     equ     3       ;sequential reads
      (0003)            0034 Wcnt     equ     3       ;sequential writes
                        0035 ;
0100  C30701            0036          Jp      Mem     ;enter from CPM
                        0037 ;
0103  00                0038 $stat    Db      Tstat   ;status port
0104  01                0039 $data    Db      Tdata   ;data port
0105  40                0040 $rda     Db      Rda     ;data ready flag
0106  80                0041 $tbe     Db      Tbe     ;transmit buffer empty
                        0042 ;
```

```
0174  ED52              0101          Sbc     HL,DE   ;within program area
0176  E1                0102          Pop     HL
0177  F28301            0103          Jp      P,Mem05 ;FBA is ok, jump
017A  21A005            0104 Mem045   Ld      HL,Memt ;if FBA within program
017D  CD8003            0105          Call    Dsply   ;area, set it to end of
                        0106                          ;program and print
0180  219906            0107          Ld      HL,Tend ;a warning.
                        0108 Mem05                    ;Save First byte addr
0183  222005            0109          Ld      (Memi),HL
0186  216E04            0110          Ld      HL,Memc ;print enter last
0189  CD8003            0111          Call    Dsply   ;byte address
018C  CD8D03            0112          Call    Entr    ;...accept address
018F  E5                0113          Push    HL      ;Save LBA
0190  E5                0114          Push    HL
0191  B7                0115          Or      A       ;clear carry flag
0192  ED5B2005          0116          Ld      DE,(Memi)
0196  ED52              0117          Sbc     HL,DE   ;make sure FBA < LBA
0198  300A              0118          Jr      NC,Mem06;its OK, jump
019A  E1                0119          Pop     HL      ;restore stack
019B  E1                0120          Pop     HL
019C  21D405            0121          Ld      HL,Memu ;FBA is >= LBA, so
019F  CD8003            0122          Call    Dsply   ;print error message
01A2  18BE              0123          Jr      Mem04   ;and accept addresses
                        0124                          ;again.
                        0125 Mem06                    ;now addresses accepted,
01A4  C1                0126          Pop     BC      ;BC <- LBA
01A5  21D904            0127          Ld      HL,Memg+5 ;convert it for print
01A8  CD8003            0128          Call    Cha
01AB  ED4B2005          0129          Ld      BC,(Memi) ;convert FBA for print
01AF  21D404            0130          Ld      HL,Memg
01B2  CD8003            0131          Call    Cha
01B5  E1                0132          Pop     HL      ;HL <- LBA
01B6  E5                0133          Push    HL
01B7  ED5B9D05          0134          Ld      DE,(Memr) ;get first byte addr
                        0135                          ;of operating system
01BB  B7                0136          Or      A,A     ;clear carry flag
01BC  ED52              0137          Sbc     HL,DE   ;check if area to be
                        0138                          ;tested will overwrite
                        0139                          ;operating system code
01BE  380A              0140          Jr      C,Mem08 ;jump if OS spared
01C0  212706            0141          Ld      HL,Memw ;operating system will
01C3  CD8003            0142          Call    Dsply   ;be destroyed, print
                        0143                          ;warning message
01C6  AF                0144          Xor     A       ;and set flag to block
                        0145                          ;warm boot of OS
01C7  329F05            0146          Ld      (Mems),A
                        0147 Mem08                    ;print abort instruction
01CA  210B06            0148          Ld      HL,Memv
01CD  CD8003            0149          Call    Dsply
01D0  D1                0150          Pop     DE      ;DE <- LBA
01D1  13                0151          Inc     DE      ;LBA <- LBA+1
                        0152 ;
                        0153 ; Main loop of memory test:
                        0154 ; Begin a pass.
                        0155 ;
01D2  0E01              0156 Mem1     Ld      C,1     ;initialize pattern no.
01D4  210000            0157          Ld      HL,0    ;initialize error count
```

```
                        0044 ;
0107  F3                0045 Mem     Di              ;disable interrupts
0108  310001            0046         Ld      SP,Stack ;initialize stack
                        0047                         ;pick up first byte
                        0048                         ;of operating system
                        0049                         ;and save it, so that
                        0050                         ;we can later determine
                        0051                         ;whether test has
                        0052                         ;destroyed resident code
010B  2A0600            0053         Ld      HL,(CPM+6)
010E  229D05            0054         Ld      (Memr),HL
0111  7E                0055         Ld      A,(HL)
0112  329F05            0056         Ld      (Mems),A
0115  019906            0057         Ld      BC,Tend ;format address of
0118  21CF05            0058         Ld      HL,Memt1;end of test
011B  CD8003            0059         Call    Cha
                        0060 Mem01                   ;test starts here
011E  210000            0061         Ld      HL,0
                        0062                         ;initialize pass count,
                        0063                         ;cumulative error count,
                        0064                         ;and address OR product
0121  22D204            0065         Ld      (Memf),HL
0124  225406            0066         Ld      (Memx),HL
0127  222505            0067         Ld      (Meml),HL
012A  21FFFF            0068         Ld      HL,-1    ;init address AND
012D  222305            0069         Ld      (Memk),HL
0130  210704            0070         Ld      HL,Mema ;print program title
0133  CD8003            0071         Call    Dsply
                        0072 Mem03                   ;get test mode
0136  3E01              0073         Ld      A,1      ;set default=itemize
0138  326805            0074         Ld      (Memp),A
013B  213005            0075         Ld      HL,Memn
013E  CD8003            0076         Call    Dsply    ;print select I,T,or E
0141  CDFC03            0077         Call    Crlf
0144  3E3E              0078         Ld      A,'>'    ;provide a cue mark
0146  CD7F06            0079         Call    Co
0149  CD5606            0080         Call    Ciw      ;wait for input
014C  CD7F06            0081         Call    Co       ;echo it
014F  F620              0082         Or      A,20H    ;fold to lower case
0151  FE65              0083         Cp      'e'      ;if E exit
0153  CA3F02            0084         Jp      Z,Mem55
0156  FE69              0085         Cp      'i'      ;if I itemize errors
0158  2808              0086         Jr      Z,Mem04
015A  FE74              0087         Cp      't'      ;if T print total
                        0088                         ;errors only
015C  20D8              0089         Jr      NZ,Mem03;none,try again
015E  AF                0090         Xor     A        ;set total only flag
015F  326805            0091         Ld      (Memp),A
                        0092 Mem04                   ;now get memory limits
0162  214204            0093         Ld      HL,Memb ;print enter FBA
0165  CD8003            0094         Call    Dsply
0168  CDBD03            0095         Call    Entr     ;get 16 bit address
016B  7C                0096         Ld      A,H      ;if upper byte of FBA
016C  B7                0097         Or      A,A      ;is negative, OK to use
016D  FA8301            0098         Jp      M,Mem05 ;so jump
0170  119906            0099         Ld      DE,Tend ;otherwise,
0173  E5                0100         Push    HL       ;make sure FBA is not
```

*(continued on the top of page 24)*

```
01D7  22D004            0158         Ld      (Meme),HL
                        0159 ;
                        0160 ; Test all of designated memory for
                        0161 ; current pattern.
                        0162 ;
                        0163 ; Write pattern into memory.
                        0164 ;
01DA  0603              0165 Mem15   Ld      B,Wcnt  ;initialize writes counter
01DC  2A2005            0166 Mem2    Ld      HL,(Memi) ;get first byte address
                        0167                         ;to test
01DF  CD5D06            0168         Call    Ci      ;check keyboard
01E2  B7                0169         Or      A,A
01E3  2053              0170         Jr      NZ,Mem5 ;character waiting,
                        0171                         ;interrupt test
01E5  C5                0172         Push    BC      ;save pattern and
                        0173                         ;writes counter
                        0174 Mem21                   ;now write this pattern
01E6  CD2903            0175         Call    Pattn   ;compute pattern for
                        0176                         ;this memory address
01E9  77                0177         Ld      (HL),A  ;...write it
01EA  23                0178         Inc     HL      ;advance memory address
01EB  7D                0179         Ld      A,L     ;check if end of area
01EC  BB                0180         Cp      E       ;to be tested
01ED  20F7              0181         Jr      NZ,Mem21;loop,not yet
01EF  7C                0182         Ld      A,H
01F0  BA                0183         Cp      D
01F1  20F3              0184         Jr      NZ,Mem21;loop,not done yet
01F3  C1                0185         Pop     BC      ;get writes counter
01F4  10E6              0186         Djnz    Mem2    ;write pattern over
                        0187                         ;and over
                        0188                         ;
01F6  0603              0189         Ld      B,Rcnt  ;initialize reads
                        0190                         ;counter
                        0191 ;
                        0192 ; Now read pattern back from memory and
                        0193 ; compare to computed pattern.  If
                        0194 ; difference is found on first read,
                        0195 ; assume a possible write error.  If
                        0196 ; first read matches, compare 16 more
                        0197 ; times looking for soft read errors.
                        0198 ;
01F8  2A2005            0199 Mem3    Ld      HL,(Memi)       ;get FBA
                        0200                         ;of memory to test
01FB  CD5D06            0201         Call    Ci      ;check keyboard
01FE  B7                0202         Or      A,A     ;if character waiting
01FF  2037              0203         Jr      NZ,Mem5 ;interrupt test
0201  C5                0204         Push    BC      ;save pattern and
                        0205                         ;reads counter
0202  CD2903            0206 Mem31   Call    Pattn   ;Compute pattern for
                        0207                         ;this memory address
0205  47                0208         Ld      B,A     ;...save it
0206  7E                0209         Ld      A,(HL)  ;read memory
0207  B8                0210         Cp      B       ;is data correct?
0208  2806              0211         Jr      Z,Mem32 ;yes, jump
020A  70                0212         Ld      (HL),B  ;write the correct data
020B  CDBB02            0213         Call    Err1    ;data doesn't match,
                        0214                         ;print possible write
```

*(continued on the next page)*

# Z-80 Memory

```
                        0215                            ;error audit
020E  1814              0216            Jr      Mem35   ;test next address
                        0217 Mem32                      ;data matched ok on
                        0218                            ;first try, try for a
                        0219                            ;soft read error
0210  96                0220            Sub     (HL)    ;by hitting this address
0211  86                0221            Add     (HL)    ;solid 16 times
0212  96                0222            Sub     (HL)
0213  86                0223            Add     (HL)
0214  96                0224            Sub     (HL)
0215  86                0225            Add     (HL)
0216  96                0226            Sub     (HL)
0217  86                0227            Add     (HL)
0218  96                0228            Sub     (HL)
0219  86                0229            Add     (HL)
021A  96                0230            Sub     (HL)
021B  86                0231            Add     (HL)
021C  96                0232            Sub     (HL)
021D  86                0233            Add     (HL)
021E  96                0234            Sub     (HL)
021F  86                0235            Add     (HL)
0220  B8                0236            Cp      B       ;does data still match?
0221  C4C902             0237            Call    NZ,Err2 ;no,print possible
                        0238                            ;read error audit
0224  23                0239 Mem35      Inc     HL      ;advance memory address
0225  7D                0240            Ld      A,L     ;check if reached end of
0226  BB                0241            Cp      E       ;memory area to be tested
0227  20D9              0242            Jr      NZ,Mem31;not done yet,loop
0229  7C                0243            Ld      A,H
022A  BA                0244            Cp      D
022B  20D5              0245            Jr      NZ,Mem31;not done yet,loop
022D  C1                0246            Pop     BC      ;restore pattern and
                        0247                            ;read counter
022E  10C8              0248            Djnz    Mem3    ;read pattern over
                        0249                            ;and over
                        0250 ;
                        0251 ;
                        0252 ; Done with one pattern, advance
                        0253 ; to next and check for end of pass.
                        0254 ;
0230  0C                0255            Inc     C       ;increment pattern
0231  79                0256            Ld      A,C
0232  FE0B              0257            Cp      11      ;done yet?
0234  20A4              0258            Jr      Nz,Mem15;no,loop
0236  1820              0259            Jr      Mem6    ;audit this pass
                        0260 ;
                        0261 ;
                        0262 ; Character waiting on keyboard,
                        0263 ; interrupt test and check for
                        0264 ; exit request.
                        0265 ;
0238  F620              0266 Mem5       Or      A,20H   ;fold to lower case
023A  FE65              0267            Cp      'e'
023C  C21E01            0268            Jp      NZ,Mem01;Restart test if not E
                        0269 Mem55                      ;Exit from test
023F  212705            0270            Ld      HL,Memm ;print Goodbye
0242  CDB003            0271            Call    Dsply
```

```
029D  CDB003            0329            Call    Cha     ;of failing addresses
02A0  ED4B2505          0330            Ld      BC,(Mem1)
02A4  211B05            0331            Ld      HL,Memg4;convert logical OR
02A7  CDB003            0332            Call    Cha     ;of failing addresses
                        0333 Mem67                      ;now print pass audit
02AA  21D404            0334            Ld      HL,Memg
02AD  CDB003            0335            Call    Dsply
02B0  3A2205            0336            Ld      A,(Memj);rotate bit crosstalk
02B3  07                0337            Rlca            ;pattern so that over
02B4  322205            0338            Ld      (Memj),A;eight passes all bit
                        0339                            ;patterns will be used.
02B7  D1                0340            Pop     DE      ;restore LBA+1
02B8  C3D201            0341            Jp      Mem1    ;start another pass.
                        0342 ;
                        0343 ; Error auditing routine.
                        0344 ; Console output of the form:
                        0345 ;
                        0346 ; A=xxxx P=xx C=xx Xor=xx Error-type
                        0347 ;
                        0348 ; A = failing address
                        0349 ; P = calculated pattern
                        0350 ; C = actual contents of address
                        0351 ; Xor = exclusive OR of pattern and contents
                        0352 ;       (isolates failing bit)
                        0353 ; Error-type = Rd presumed read (soft) error
                        0354 ;              Wt presumed write (hard) error
                        0355 ;
02BB  F5                0356 Err1       Push    AF      ;possible write error
02BC  3E57              0357            Ld      A,'W'
02BE  32CD04            0358            Ld      (Memd5),A
02C1  3E74              0359            Ld      A,'t'
02C3  32CE04            0360            Ld      (Memd5+1),A
02C6  F1                0361            Pop     AF
02C7  180C              0362            Jr      Error
                        0363 ;
02C9  F5                0364 Err2       Push    AF      ;possible read error
02CA  3E52              0365            Ld      A,'R'
02CC  32CD04            0366            Ld      (Memd5),A
02CF  3E64              0367            Ld      A,'d'
02D1  32CE04            0368            Ld      (Memd5+1),A
02D4  F1                0369            Pop     AF
                        0370 ;
                        0371 Error                      ;error auditing routine
02D5  C5                0372            Push    BC      ;save registers
02D6  D5                0373            Push    DE
02D7  E5                0374            Push    HL
02D8  F5                0375            Push    AF
02D9  A8                0376            Xor     B       ;Logical exclusive OR
                        0377                            ;of calculated pattern
                        0378                            ;and actual memory
                        0379                            ;contents
02DA  4F                0380            Ld      C,A
02DB  21C404            0381            Ld      HL,Memd4;convert OR for output
02DE  CDA803            0382            Call    Chab
02E1  F1                0383            Pop     AF      ;get memory contents
                        0384                            ;and convert it for
                        0385                            ;output
```

```
0245 2A9D05          0272          Ld      HL,(Memr)
0248 3A9F05          0273          Ld      A,(Mems);check if operating
                     0274                          ;system was destroyed
024B BE              0275          Cp      (HL)
024C CA0000          0276          Jp      Z,CPM   ;no,return control to
                     0277                          ;operating system
024F 216905          0278          Ld      HL,Memq ;operating system is
0252 CD8003          0279          Call    Dsply   ;destroyed, print a
                     0280                          ;warning and loop for
                     0281                          ;manual restart
0255 C35502          0282          Jp      $
                     0283 ;
                     0284 ;
                     0285 ; Done with pass through memory.
                     0286 ;
                     0287 ; Print console audit in the form:
                     0288 ;
                     0289 ; Pass: xxxx  Errors: xxxx  Cum. errors: xxxx
                     0290 ;  (if cumulative errors > zero then also print)
                     0291 ;    And: xxxx  Or: xxxx
                     0292 ;
                     0293 Mem6                     ;end of pass
0258 D5              0294          Push    DE      ;save LBA+1
0259 ED4BD204        0295          Ld      BC,(Memf)
025D 03              0296          Inc     BC      ;count passes
025E ED43D204        0297          Ld      (Memf),BC
0262 21E504          0298          Ld      HL,Memg1;convert pass count
0265 CD8003          0299          Call    Cha
0268 ED4BD004        0300          Ld      BC,(Meme)
026C 21F304          0301          Ld      HL,Memg2;convert error count
026F CD8003          0302          Call    Cha     ;this pass
0272 ED4B5406        0303          Ld      BC,(Memx)
0276 2AD004          0304          Ld      HL,(Meme)
0279 09              0305          Add     HL,BC   ;accumulate errors
                     0306                          ;for all passes
027A 225406          0307          Ld      (Memx),HL
027D E5              0308          Push    HL      ;format cumulative
027E C1              0309          Pop     BC      ;errors
027F 210605          0310          Ld      HL,Memg23
0282 CD8003          0311          Call    Cha
0285 3E0D            0312          Ld      A,Cr    ;set up output to
                     0313                          ;skip AND & OR of
                     0314                          ;failing memory
                     0315                          ;addresses if no
                     0316                          ;errors have been
                     0317                          ;found.
0287 320B05          0318          Ld      (Memg25),A
028A 2A5406          0319          Ld      HL,(Memx)
028D 7C              0320          Ld      A,H     ;make sure no errors
028E B5              0321          Or      A,L
028F 2819            0322          Jr      Z,Mem67 ;none yet, jump
0291 3E20            0323          Ld      A,' '   ;remove the carriage
                     0324                          ;return from output
                     0325                          ;string...
0293 320B05          0326          Ld      (Memg25),A
0296 ED4B2305        0327          Ld      BC,(Memk)
029A 211105          0328          Ld      HL,Memg3;convert logical AND
```

*(continued on the top of page 26)*

```
02E2 4F              0386          Ld      C,A
02E3 21BC04          0387          Ld      HL,Memd3
02E6 CDA803          0388          Call    Chab
02E9 48              0389          Ld      C,B     ;convert pattern
02EA 21AF04          0390          Ld      HL,Memd2
02ED CDA803          0391          Call    Chab
02F0 C1              0392          Pop     BC      ;convert current
02F1 C5              0393          Push    BC      ;memory address
02F2 21A104          0394          Ld      HL,Memd1
02F5 CD8003          0395          Call    Cha
02F8 2AD004          0396          Ld      HL,(Meme)
02FB 23              0397          Inc     HL      ;count errors this pass
02FC 22D004          0398          Ld      (Meme),HL
02FF D1              0399          Pop     DE      ;get current memory
0300 D5              0400          Push    DE      ;address
0301 2A2305          0401          Ld      HL,(Memk)
0304 7A              0402          Ld      A,D     ;save logical AND of
0305 A4              0403          And     H       ;failing addresses
0306 67              0404          Ld      H,A
0307 7B              0405          Ld      A,E
0308 A5              0406          And     L
0309 6F              0407          Ld      L,A
030A 222305          0408          Ld      (Memk),HL
030D 2A2505          0409          Ld      HL,(Meml)
0310 7A              0410          Ld      A,D     ;Save logical OR of
0311 B4              0411          Or      H       ;failing addresses
0312 67              0412          Ld      H,A
0313 7B              0413          Ld      A,E
0314 B5              0414          Or      L
0315 6F              0415          Ld      L,A
0316 222505          0416          Ld      (Meml),HL
0319 3A6805          0417          Ld      A,(Memp);check itemize errors
                     0418                          ;flag
031C B7              0419          Or      A
031D 2806            0420          Jr      Z,Err9  ;skip print if flag=0
031F 219904          0421          Ld      HL,Memd ;print error audit
0322 CD8003          0422          Call    Dsply
0325 E1              0423 Err9     Pop     HL      ;restore registers
0326 D1              0424          Pop     DE      ;and return to main
0327 C1              0425          Pop     BC      ;test
0328 C9              0426          Ret
                     0427 ;
                     0428 ;
                     0429 ; Compute test data pattern for given
                     0430 ; memory address.
                     0431 ;
                     0432 ; Call with  HL = memory address
                     0433 ;                C = pattern counter
                     0434 ;
                     0435 ; Return      A = data pattern
                     0436 ;
                     0437 Pattn                    ;pattern computation
0329 E5              0438          Push    HL
032A 0600            0439          Ld      B,0     ;branch on pattern
032C 213103          0440          Ld      HL,Patt0-2
032F 09              0441          Add     HL,BC
0330 09              0442          Add     HL,BC
```

*(continued on the next page)*

# Z-80 Memory

```
0331  E3          0443            Ex      (SP),HL ;(restore mem addr)
0332  C9          0444            Ret             ;(branch)
0333  1812        0445 Patt0      Jr      Pat1    ;1 Cambridge pattern
0335  181E        0446            Jr      Pat2    ;2 address
0337  181E        0447            Jr      Pat3    ;3 alternate 1s and 0s
0339  181F        0448            Jr      Pat4    ;4 address inverse
033B  1820        0449            Jr      Pat5    ;5 alternate 0s and 1s
033D  1813        0450            Jr      Pat6    ;6 all ones
033F  181F        0451            Jr      Pat7    ;7 Cambridge inverse
0341  180D        0452            Jr      Pat8    ;8 all zeros
0343  1826        0453            Jr      Pat9    ;9 bit crosstalk
0345  1831        0454            Jr      Pat10   ;10 bit crosstalk inverse
                  0455 ;
0347  7D          0456 Pat1       Ld      A,L     ;Cambridge pattern
0348  0F          0457            Rrca
0349  0F          0458            Rrca
034A  0F          0459            RRca
034B  AC          0460            Xor     H
034C  E601        0461            And     1
034E  2802        0462            Jr      Z,Ones
0350  AF          0463 Zeros      Xor     A
0351  C9          0464            Ret
0352  3EFF        0465 Ones       Ld      A,0FFH
0354  C9          0466            Ret
                  0467 ;
0355  7D          0468 Pat2       Ld      A,L     ;address
0356  C9          0469            Ret
                  0470 ;
0357  3EAA        0471 Pat3       Ld      A,0AAH  ;alternate 1s and 0s
0359  C9          0472            Ret
                  0473 ;
035A  7D          0474 Pat4       Ld      A,L     ;address inverse
035B  2F          0475            Cpl
035C  C9          0476            Ret
                  0477 ;
035D  3E55        0478 Pat5       Ld      A,55H   ;alternate 0s and 1s
035F  C9          0479            Ret
                  0480 ;
      (0352)      0481 Pat6       Equ     Ones    ;all bits = one
                  0482 ;
0360  7D          0483 Pat7       Ld      A,L     ;Cambridge inverse
0361  0F          0484            Rrca
0362  0F          0485            Rrca
0363  0F          0486            Rrca
0364  AC          0487            Xor     H
0365  E601        0488            And     1
0367  28E7        0489            Jr      Z,Zeros
0369  18E7        0490            Jr      Ones
                  0491 ;
      (0350)      0492 Pat8       Equ     Zeros   ;all bits = zero
                  0493 ;
036B  7D          0494 Pat9       Ld      A,L     ;bit crosstalk
036C  1F          0495            Rra
036D  3804        0496            Jr      C,Pat91
036F  3A2205      0497            Ld      A,(Memj)
0372  C9          0498            Ret
0373  3A2205      0499 Pat91      Ld      A,(Memj)
```

```
                  0557 ;
                  0558 ; Call         HL = address for 2 character
                  0559 ;                   ASCII output string
                  0560 ;               C = 8 bit binary data
                  0561 ;
                  0562 ; Return       HL,DE,BC unchanged, A destroyed
                  0563 ;
03A8  E5          0564 Chab       Push    HL      ;save registers
03A9  D5          0565            Push    DE
03AA  C5          0566            Push    BC
03AB  23          0567            Inc     HL
03AC  1602        0568            Ld      D,2
03AE  18D8        0569            Jr      Cha1
                  0570 ;
                  0571 ;
                  0572 ; Print character string
                  0573 ;
                  0574 ; Call     HL = first byte address of
                  0575 ;               output string (must end
                  0576 ;               with ASCII carriage return)
                  0577 ;
03B0  CDFC03      0578 Dsply      Call    Crlf
03B3  7E          0579 Dsply1     Ld      A,(HL)
03B4  CD7F06      0580            Call    Co      ;send this char.
03B7  FE0D        0581            Cp      Cr      ;end of string?
03B9  C8          0582            Ret     Z       ;yes,exit
03BA  23          0583            Inc     HL      ;no,bump string
                  0584                            ;pointer
03BB  18F6        0585            Jr      Dsply1
                  0586 ;
                  0587 ;
                  0588 ; Get keyboard entry of hex integer
                  0589 ;
                  0590 ; Return HL = 16 bit binary data
                  0591 ;
03BD  210000      0592 Entr       Ld      HL,0    ;initialize data
03C0  CDFC03      0593            Call    Crlf    ;send carriage return
                  0594                            ;and line feed
03C3  3E3E        0595            Ld      A,'>'   ;send a cue mark
03C5  CD7F06      0596            Call    Co
03C8  0E04        0597            Ld      C,4     ;char counter
03CA  CD5606      0598 Entr1      Call    Ciw     ;get 1 character
03CD  CD7F06      0599            Call    Co      ;echo it
03D0  FE0D        0600            Cp      Cr      ;carriage return?
03D2  C8          0601            Ret     Z       ;yes,exit
03D3  FE41        0602            Cp      'A'     ;is it 0-9?
03D5  3802        0603            Jr      C,Entr15;yes
03D7  E6DF        0604            And     A,0DFH  ;no,force lower case
                  0605 Entr15                     ;
                  0606                            ;shift previous data
                  0607                            ;left 4 bits
03D9  29          0608            Add     HL,HL
03DA  29          0609            Add     HL,HL
03DB  29          0610            Add     HL,HL
03DC  29          0611            Add     HL,HL
03DD  3816        0612            Jr      C,Entr3 ;if overflow, print ?
03DF  FE30        0613            Cp      '0'     ;is it 0-F?
```

```
0376  2F          0500            Cpl
0377  C9          0501            Ret
                  0502 ;
0378  7D          0503 Pat10      Ld      A,L       ;bit crosstalk inverse
0379  1F          0504            Rra
037A  30F7        0505            Jr      NC,Pat91
037C  3A2205      0506            Ld      A,(Memj)
037F  C9          0507            Ret
                  0508 ;
                  0509 ;
                  0510 ;
                  0511 ; Binary to hex ASCII conversion
                  0512 ;
                  0513 ; Call         HL = address for 4 char
                  0514 ;                   ASCII output string
                  0515 ;              BC = 16 bit binary data
                  0516 ;
                  0517 ; Returns      HL,DE,BC unchanged
                  0518 ;              A  = garbage
                  0519 ;
0380  E5          0520 Cha        Push    HL        ;Save registers
0381  D5          0521            Push    DE
0382  C5          0522            Push    BC
0383  23          0523            Inc     HL
0384  23          0524            Inc     HL
0385  23          0525            Inc     HL
0386  1604        0526            Ld      D,4       ;char counter
0388  79          0527 Cha1       Ld      A,C       ;next 4 bits
0389  E60F        0528            And     0FH
038B  FE0A        0529            Cp      0AH       ;is it A-F?
038D  3802        0530            Jr      C,Cha15   ;no
038F  C607        0531            Add     7         ;yes
0391  C630        0532 Cha15      Add     '0'       ;form ASCII
0393  77          0533            Ld      (HL),A    ;store this char.
0394  2B          0534            Dec     HL        ;back up through
                  0535                              ;output area
0395  1E04        0536            Ld      E,4       ;double right
0397  B7          0537            Or      A         ;shift 4 bits
0398  78          0538 Cha2       Ld      A,B
0399  1F          0539            Rra
039A  47          0540            Ld      B,A
039B  79          0541            Ld      A,C
039C  1F          0542            Rra
039D  4F          0543            Ld      C,A
039E  1D          0544            Dec     E         ;decrement shift
                  0545                              ;counter
039F  20F7        0546            Jr      NZ,Cha2   ;still shifting...
03A1  15          0547            Dec     D         ;decrement
                  0548                              ;character counter
03A2  20E4        0549            Jr      NZ,Cha1   ;still converting...
03A4  C1          0550            Pop     BC        ;restore registers
03A5  D1          0551            Pop     DE        ;and exit
03A6  E1          0552            Pop     HL
03A7  C9          0553            Ret
                  0554 ;
                  0555 ;
                  0556 ; Binary to Hex ASCII conversion, 8 bits
```

*(continued on the top of page 28)*

```
03E1  3812        0614            Jr      C,Entr3 ;illegal character
03E3  FE47        0615            Cp      'F'+1
03E5  300E        0616            Jr      NC,Entr3;illegal character
03E7  FE41        0617            Cp      'A'     ;is it A-F?
03E9  3802        0618            Jr      C,Entr2 ;no,its 0-9
03EB  C609        0619            Add     9       ;add fudge factor
03ED  E60F        0620 Entr2      And     0FH     ;isolate 4 bits
03EF  B5          0621            Or      L       ;Merge with previous
03F0  6F          0622            Ld      L,A     ;data
03F1  0D          0623            Dec     C       ;count characters
03F2  C8          0624            Ret     Z       ;exit if 4 received
03F3  18D5        0625            Jr      Entr1   ;get another char
03F5  3E3F        0626 Entr3      Ld      A,'?'   ;print question mark
03F7  CD7F06      0627            Call    Co      ;and restart entry
03FA  18C1        0628            Jr      Entr
                  0629 ;
                  0630 ;
                  0631 ; Print carriage return and line feed
                  0632 ;
03FC  3E0D        0633 Crlf       Ld      A,Cr
03FE  CD7F06      0634            Call    Co
0401  3E0A        0635            Ld      A,Lf
0403  CD7F06      0636            Call    Co
0406  C9          0637            Ret
                  0638 ;
                  0639 ;
                  0640 ; Miscellaneous messages and data area
                  0641 ;
0407  0A4C6162    0642 Mema       Db      Lf,'Laboratory '
      6F726174
      6F727920
0413  4D696372    0643            Db      'Microsystems   Z-80'
      6F737973
      74656D73
      2020205A
      2D3830
0426  204D656D    0644            Db      ' Memory Test  '
      6F727920
      54657374
      2020
0434  20766572    0645            Db      ' version 2.3',Lf,Cr
      73696F6E
      20322E33
      0A0D
0442  456E7465    0646 Memb       Db      'Enter address of first '
      72206164
      64726573
      73206F66
      20666972
      737420
0459  6D656D6F    0647            Db      'memory byte to test: ',Cr
      72792062
      79746520
      746F2074
      6573743A
      0D
046E  456E7465    0648 Memc       Db      'Enter address of last '
```

*(continued on the next page)*

# Z-80 Memory

```
                72206164
                64726573
                73206F66
                206C6173
                7420
0484  6D656D6F    0649          Db      'memory byte to test: ',Cr
      72792062
      79746520
      746F2074
      6573743A
      0D
0499  41646472    0650 Memd     Db      'Address='
      6573733D
04A1  24242424    0651 Memd1    Db      '$$$$  Pattern='
      20205061
      74746572
      6E3D
04AF  24242020    0652 Memd2    Db      '$$  Contents='
      436F6E74
      656E7473
      3D
04BC  24242020    0653 Memd3    Db      '$$  Xor='
      586F723D
04C4  24242020    0654 Memd4    Db      '$$  Type='
      54797065
      3D
04CD  20200D      0655 Memd5    Db      '  ',Cr
04D0  0000        0656 Meme     Dw      0       ;errors this pass
04D2  0000        0657 Memf     Dw      0       ;pass count
04D4  24242424    0658 Memg     Db      '$$$$-$$$$  Pass:  '
      2D242424
      24202050
      6173733A
      20
04E5  24242424    0659 Memg1    Db      '$$$$  Errors:  '
      20204572
      726F7273
      3A20
04F3  24242424    0660 Memg2    Db      '$$$$  Cum. errors:  '
      20204375
      6D2E2065
      72726F72
      733A20
0506  24242424    0661 Memg23   Db      '$$$$ '
      20
050B  0D416E64    0662 Memg25   Db      Cr,'And:  '
      3A20
0511  24242424    0663 Memg3    Db      '$$$$  Or:  '
      20204F72
      3A20
051B  24242424    0664 Memg4    Db      '$$$$',Cr
      0D
0520  0000        0665 Memi     Dw      0       ;first byte address
                  0666                          ;to test
0522  FE          0667 Memj     Db      0FEH    ;bit crosstalk pattern
0523  FFFF        0668 Memk     Dw      -1      ;logical AND of
                  0669                          ;failing addresses
                61737420
                62797465
                20616464
                72657373
                206C6573
                7320
05F2  7468616E    0688          Db      'than first byte address. ',Cr
      20666972
      73742062
      79746520
      61646472
      6573732E
      0D
060B  0A          0689 Memv     Db      Lf
060C  546F2061    0690          Db      'To abort test push any key'
      626F7274
      20746573
      74207075
      73682061
      6E79206B
      6579
0626  0D          0691          Db      Cr
0627  5761726E    0692 Memw     Db      'Warning: operating system '
      696E673A
      206F7065
      72617469
      6E672073
      79737465
      6D20
0641  77696C6C    0693          Db      'will be destroyed. ',Cr
      20626520
      64657374
      726F7965
      642E0D
0654  0000        0694 Memx     Dw      0       ;cumulative error count
                  0695 ;
                  0696 ;
                  0697 ; Return (in Acc) one character from console.
                  0698 ;
0656  CD5D06      0699 Ciw      Call    Ci
0659  B7          0700          Or      A       ;anything there?
065A  28FA        0701          Jr      Z,Ciw   ;no, loop
065C  C9          0702          Ret             ;got char, return
                  0703 ;
                  0704 ; Read console status and input
                  0705 ; a character (in Acc) if waiting,
                  0706 ; else return Acc = zero.
                  0707 ;
065D  C5          0708 Ci       Push    BC      ;save register
065E  3A0301      0709          Ld      A,($stat)
0661  4F          0710          Ld      C,A
0662  3A0501      0711          Ld      A,($rda)
0665  47          0712          Ld      B,A
0666  ED78        0713 Ci1      In      A,(C)   ;read status port
0668  A0          0714          And     B       ;check if character
                  0715                          ;waiting now
0669  2002        0716          Jr      NZ,Ci2  ;yes, jump
```

```
0525  0000          0670 Meml   Dw    0       ;logical OR of
                    0671                      ;failing addresses
0527  0A476F6F      0672 Memm   Db    Lf,'Goodbye',Cr
      64627965
      0D
0530  493D6974      0673 Memn   Db    'I=itemize errors, '
      656D697A
      65206572
      726F7273
      2C20
0542  543D7072      0674        Db    'T=print error total only, '
      696E7420
      6572726F
      7220746F
      74616C20
      6F6E6C79
      2C20
055C  453D6578      0675        Db    'E=exit test',Cr
      69742074
      6573740D
0568  00            0676 Memp   Db    0       ;flag 1=itemize,0=total
0569  4F706572      0677 Memq   Db    'Operating system '
      6174696E
      67207379
      7374656D
      20
057A  64657374      0678        Db    'destroyed, manual '
      726F7965
      642C206D
      616E7561
      6C20
058C  72657365      0679        Db    'reset is needed.',Cr
      74206973
      206E6565
      6465642E
      0D
059D  0000          0680 Memr   Dw    0       ;address of first byte
                    0681                      ;of operating system
059F  00            0682 Mems   Db    0       ;actual first byte of OS
                    0683                      ;before testing begins
05A0  456E6420      0684 Memt   Db    'End of program used as first '
      6F662070
      726F6772
      616D2075
      73656420
      61732066
      69727374
      20
05BD  61646472      0685        Db    'address to test = '
      65737320
      746F2074
      65737420
      3D20
05CF  24242424      0686 Memt1  Db    '$$$$',Cr
      0D
05D4  4572726F      0687 Memu   Db    'Error: last byte address less '
      723A206C
```

*(continued on the top of page 30)*

```
066B  C1            0717        Pop   BC      ;no,restore
066C  C9            0718        Ret           ;register and exit
066D  3A0401        0719 Ci2    Ld    A,($data)     ;read data port
0670  4F            0720        Ld    C,A
0671  ED78          0721        In    A,(C)
0673  C1            0722        Pop   BC      ;restore register
0674  E67F          0723        And   7FH     ;strip parity bit
0676  FE1B          0724        Cp    Esc     ;if not escape code,
0678  C0            0725        Ret   NZ      ;return to caller
                    0726                      ;if escape code,
                    0727                      ;restart test
0679  310001        0728        Ld    SP,Stack
067C  C31E01        0729        Jp    Mem01
                    0730 ;
                    0731 ; Write one character (in Acc) to console.
                    0732 ;
067F  C5            0733 Co     Push  BC      ;save registers
0680  F5            0734        Push  AF
0681  3A0301        0735        Ld    A,($stat)
0684  4F            0736        Ld    C,A
0685  3A0601        0737        Ld    A,($tbe)
0688  47            0738        Ld    B,A
                    0739 Co1                  ;read status port
0689  ED78          0740        In    A,(C)
068B  A0            0741        And   A,B     ;check if transmit
                    0742                      ;busy
068C  28FB          0743        Jr    Z,Co1   ;yes,loop
068E  3A0401        0744        Ld    A,($data)     ;no,send character
0691  4F            0745        Ld    C,A
0692  F1            0746        Pop   AF      ;(restore data)
0693  ED79          0747        Out   (C),A   ;(send it)
0695  C1            0748        Pop   BC      ;(restore other reg.)
0696  C9            0749        Ret           ;return to caller
                    0750 ;
      (0699)        0751 Tend   Equ   $+2
0697  (0100)        0752        End   100H

Errors              0
```

**end**

# Symbol Table

```
$DATA   0104    $RDA    0105    $STAT   0103    $TBE    0106    CHA     0380
CHA1    0388    CHA15   0391    CHA2    0398    CHAB    03A8    CI      065D
CI1     0666    CI2     066D    CIW     0656    CO      067F    CO1     0689
CPM     0000    CR      000D    CRLF    03FC    DSPLY   03B0    DSPLY1  03B3
ENTR    03BD    ENTR1   03CA    ENTR15  03D9    ENTR2   03ED    ENTR3   03F5
ERR1    02BB    ERR2    02C9    ERR9    0325    ERROR   02D5    ESC     001B
LF      000A    MEM     0107    MEM01   011E    MEM03   0136    MEM04   0162
MEM045  017A    MEM05   0183    MEM06   01A4    MEM08   01CA    MEM1    01D2
MEM15   01DA    MEM2    01DC    MEM21   01E6    MEM3    01F8    MEM31   0202
MEM32   0210    MEM35   0224    MEM5    0238    MEM55   023F    MEM6    0258
MEM67   02AA    MEMA    0407    MEMB    0442    MEMC    046E    MEMD    0499
MEMD1   04A1    MEMD2   04AF    MEMD3   04BC    MEMD4   04C4    MEMD5   04CD
MEME    04D0    MEMF    04D2    MEMG    04D4    MEMG1   04E5    MEMG2   04F3
MEMG23  0506    MEMG25  050B    MEMG3   0511    MEMG4   051B    MEMI    0520
MEMJ    0522    MEMK    0523    MEML    0525    MEMM    0527    MEMN    0530
MEMP    0568    MEMQ    0569    MEMR    059D    MEMS    059F    MEMT    05A0
MEMT1   05CF    MEMU    05D4    MEMV    060B    MEMW    0627    MEMX    0654
ONES    0352    PAT1    0347    PAT10   0378    PAT2    0355    PAT3    0357
PAT4    035A    PAT5    035D    PAT6    0352    PAT7    0360    PAT8    0350
PAT9    036B    PAT91   0373    PATTO   0333    PATTN   0329    RCNT    0003
RDA     0040    STACK   0100    TBE     0080    TDATA   0001    TEND    0699
TSTAT   0000    WCNT    0003    ZEROS   0350
```

# Small-C
## Listing I

### Incorrect Compilation of Test Routines

```
;  Small-C Compiler Rev 1.1P
; /*                                                              */
; /*     EXAMPLE TO INVESTIGATE A POTENTIAL BUG.                  */
; /*     IE WHAT DETERMINES THE ADDRESSES OF DUMMY ARGUMENTS.     */
; /*           THE FUNCTION HEADING OR THE TYPE DECLARATION.      */
; /*                                                              */
; TEST1 (ONE, TWO)
TEST1:
;       INT ONE, TWO;
;       {ONE / TWO;
           LXI   H,4
           DAD   SP
           CALL  CCGINT
           PUSH  H
           LXI   H,4
           DAD   SP
           CALL  CCGINT
           POP   D
           CALL  CCDIV
;       }
           RET
; /*                                                              */
; TEST2 (ONE, TWO)
TEST2:
;       INT TWO, ONE;
;       {ONE / TWO;
           LXI   H,2
           DAD   SP
           CALL  CCGINT
           PUSH  H
           LXI   H,6
           DAD   SP
           CALL  CCGINT
           POP   D
           CALL  CCDIV
;       }
           RET
; 0 ERRORS IN COMPILATION.
           END
```

**end**

```
            /* NOW LET USER DECLARE WHAT TYPES OF THINGS THOSE          */
            /*      THOSE ARGUMENTS WERE.                              */
            {IF (AMATCH ("CHAR", 4)) {GETARG (CCHAR); NS ();}
            ELSE IF (AMATCH ("INT", 3)) {GETARG (CINT); NS ();}
            ELSE {ERROR ("WRONG NUMBER ARGS"); BREAK;}
            }
      IF (STATEMENT () != STRETURN)      /* DO A STATEMENT, BUT IF IT'S  */
            /* A RETURN, SKIP, CLEANING UP STACK.                      */
            {MODSTK (0);
            RET ();
            }
      SP = 0;                 /* RESET STACK POINTER AGAIN.            */
      LOCPTR = STARTLOC;      /* DEALLOCATE ALL LOCALS.                */
      }
/*                                                                      */
/*          DECLARE ARGUMENT TYPES.                                     */
/*                                                                      */
/*    CALLED FROM "NEWFUNC" THIS ROUTINE ADDS AN ENTRY IN THE LOCAL     */
/*          SYMBOL TABLE FOR EACH NAMED ARGUMENT.                       */
/*                                                                      */
/*    FUNCTION COMPLETELY REWRITTEN.   PL WOODS.                        */
/*                                                                      */
GETARG (T)
      INT T;
      {INT J, ARGTOP, LEGALNAME, ADDRESS;
      CHAR N[NAMESIZE], C, *ARGPTR;
      ARGTOP = ARGSTK;
      WHILE (1)
            {IF (ARGSTK == 0) RETURN;    /* NO MORE ARGUMENTS.         */
            IF (MATCH ("*")) J = POINTER;
                  ELSE J = VARIABLE;
            IF ((LEGALNAME = SYMNAME (N)) == 0) ILLNAME ();
            IF (MATCH ("["))             /* IS IT A POINTER?           */
                  /* IT IS A POINTER SO SKIP ALL STUFF BETWEEN "[..]"  */
                  {WHILE (INBYTE () != ']')
                        IF (ENDST ()) BREAK;
                  J = POINTER;           /* ADD ENTRY AS POINTER.      */
                  }
            IF (LEGALNAME)
                  {IF (ARGPTR = FINDLOC (N))
                        /* ADD IN DETAILS OF THE TYPE AND ADDRESS OF   */
                        /*      THE NAME.                              */
                        {ARGPTR[IDENT] = J;
                        ARGPTR[TYPE] = T;
                        ADDRESS = ARGTOP - ((ARGPTR[OFFSET]&255) +
                              ((ARGPTR[OFFSET+1]&255) << 8));
                        ARGPTR[OFFSET] = ADDRESS;
                        ARGPTR[OFFSET+1] = ADDRESS >> 8 ;
                        }
                  ELSE
                        ERROR ("EXPECTING ARGUMENT NAME");
                  }
            ARGSTK = ARGSTK - 2;
            IF (ENDST ()) RETURN;
            IF (MATCH (",") == 0) ERROR ("EXPECTED COMMA");
            }
      }
```

**end**

# Listing II

## Corrective Code for Small-C Compiler

```
/*                                                              */
/*    BEGIN A FUNCTION.                                         */
/*                                                              */
/*    CALLED FROM "PARSE" THIS ROUTINE TRIES TO MAKE A FUNCTION OUT   */
/*          OUT OF WHAT FOLLOWS.   MODIFIED VERSION.   PL WOODS.      */
/*                                                              */
NEWFUNC ()
      {
      CHAR N[NAMESIZE], *PTR;
      IF (SYMNAME (N) == 0)
            {ERROR ("ILLEGAL FUNCTION OR DECLARATION");
            KILL ();      /* INVALIDATE LINE.                  */
            RETURN;
            }
      IF (PTR = FINDGLB (N))      /* ALREADY IN SYMBOL TABLE?    */
            {IF (PTR[IDENT] != FUNCTION) MULTIDEF (N);
                  /* ALREADY A VARIABLE BY THAT NAME.          */
            ELSE IF (PTR[OFFSET] == FUNCTION) MULTIDEF (N);
                  /* ALREADY A FUNCTION BY THAT NAME.          */
            ELSE PTR[OFFSET] = FUNCTION;
                  /* OTHERWISE WE HAVE WHAT WAS EARLIER ASSUMED TO BE */
                        /* A FUNCTION.                         */
            }
      /* IF NOT IN TABLE, DEFINE AS A FUNCTION NOW.            */
      ELSE ADDGLB (N, FUNCTION, CINT, FUNCTION);
      /* WE HAD BETTER SEE OPEN PAREN FOR ARGS...              */
      IF (MATCH ("(") == 0) ERROR ("MISSING OPEN PAREN");
      /* PRINT FUNCTION NAME.                                  */
      OUTSTR (N);
      COL ();
      NL ();
      /* CLEAR LOCAL STACK POINTER.   TWO LINES INSERTED.   PL WOODS.  */
      LOCPTR = STARTLOC;        /* PL WOODS                    */
      ARGSTK = 0;   /* INIT ARG COUNT.                         */
      WHILE (MATCH (")") == 0)    /* THEN COUNT ARGS.          */
            /* ANY LEGAL NAME BUMPS ARG COUNT.                 */
            {IF (SYNAME (N))
                  /* TEXT MODIFIED.   PL WOODS.                */
                  {IF (FINDLOC (N)) MULTIDEF (N);
                  ELSE
                        {ADDLOC (N, 0, 0, ARGSTK);
                        ARGSTK = ARGSTK + 2;
                        }
                  }
            ELSE {ERROR ("ILLEGAL ARGUMENT NAME"); JUNK ();}
            /* END OF MODIFICATION.   PL WOODS.                */
            BLANKS ();

            /* IF NOT CLOSING PAREN, SHOULD BE A COMMA.        */
            IF (STREQ (LINE+LPTR, ")") == 0)
                  {IF (MATCH (",") == 0)
                        ERROR ("EXPECTED COMMA");
                  }
            IF (ENDST ()) BREAK;
            }
      /* CODE TO RESET LOCPTR TO STARTLOC DELETED.   PL WOODS.   */
      SP = 0;               /* PRESET STACK POINTER.           */
      WHILE (ARGSTK)
```

*(continued on the top of page 32)*

# Listing III

## Correct Compilation

```
;  Small-C Compiler Rev 1.1P
; /*                                                            */
; /*    EXAMPLE TO INVESTIGATE A POTENTIAL BUG.                 */
; /*    IE WHAT DETERMINES THE ADDRESSES OF DUMMY ARGUMENTS.    */
; /*          THE FUNCTION HEADING OR THE TYPE DECLARATION.     */
; /*                                                            */
; TEST1 (ONE, TWO)
TEST1:
;       INT ONE, TWO;
;       {ONE / TWO;
            LXI   H,4
            DAD   SP
            CALL  CCGINT
            PUSH  H
            LXI   H,4
            DAD   SP
            CALL  CCGINT
            POP   D
            CALL  CCDIV
;       }
            RET
; /*                                                            */
; TEST2 (ONE, TWO)
TEST2:
;       INT TWO, ONE;
;       {ONE / TWO;
            LXI   H,4
            DAD   SP
            CALL  CCGINT
            PUSH  H
            LXI   H,4
            DAD   SP
            CALL  CCGINT
            POP   D
            CALL  CCDIV
;       }
            RET
; 0 ERRORS IN COMPILATION.
            END
```

**end**

# Software Reviews

**DB.BAS – General File Maintenance (Revised Version)**
Published by **Holiday Software**
**4807 Arlene Street**
**San Diego, CA 92117**
Price: **$88.95**
Reviewed by **Michael Favitta**

DB.BAS Ver. 2 is Holiday Software's new release of their general file maintenance package. Those of you who read my review of Version 1 in *Dr. Dobb's* #47 know I thought the idea was good, but the package was not what I considered complete. It also contained some serious errors. Ver. 2 of DB.BAS has corrected those errors along with adding capabilities which were lacking in the earlier version.

DB.BAS is a set of CBASIC-2 programs which allow for the easy creation and maintenance of indexed sequential files. These files can be used to hold the data required for any small application where random access to data records is desired. The package also provides the means to write out selected data records in various predetermined or user-specified formats, thus giving the user a limited "query" capability.

The package requires a system containing at least 40K of RAM, one or more disk drives, the CP/M operating system, and CBASIC-2 Ver. 2.06. The amount of available memory determines the maximum size of the index array, which in turn limits the number of records a file may contain. A 40K system will allow up to 250 records per file.

DB.BAS commands allow you to CREATE files, ADD records to existing files, CHANGE, DELETE, and FIND records by number or key, to create secondary key INDEX files, APPEND data files, perform a USER written function, and to PRINT all or select groups of records at the console or printer. The APPEND, USER, and expanded PRINT commands are new additions in Ver. 2.

The APPEND command merges some or all the records in one file into a second file. The records to be merged may be selected by matching data or field names in a record with those present in the output file. The matching routines recognize special characters that will match any digit, any alpha, and any character. If no selection criteria is chosen, you can blindly move records into the output file. This feature can be used to restructure your data files, as the output file record description need not match that of the input file. When using selection with matching data and an empty output file, you may create files containing only records with a common attribute.

The new PRINT command allows groups of records which are selected by a mechanism similar to that of the APPEND command to be output in four different formats. A horizontal table, a vertical form like that of an index card, a mailing label, and any form the user wants to provide himself. Column headings are generated automatically from record field names. When printing mailing labels, this feature is suppressed and data from fields defined as non-label at CREATE time will not be printed. These new print features enhance the flexibility of the package as Ver. 1 could only dump records in the index card format.

Another improvement present in Ver. 2 is the inclusion of a memo on how to get help if you encounter a program bug. Since Ver. 2 was not quite perfect, I took Holiday Software up on their offer and sent them documentation on some minor bugs. Their response was quick (about a week) and complete, providing both an explanation and a fix for each problem. One should note that even though the service was excellent, fixing a problem would be quite a typing effort, since only source listings are available.

DB.BAS has been designed to be easy to use. When the program is loaded, a menu listing all the commands is displayed. Once a command has been selected from the main menu, the menu for that command is displayed. Whenever a menu isn't present, you are guided by system prompts. The interactive nature of this product is what I consider to be its greatest strength. A comprehensive user's manual is also supplied. It is well-organized and contains examples of how each command may be used. All data files used in the examples are provided so that the examples in the manual may be used as a hands-on tutorial.

Holiday Software's new release of DB.BAS is definitely a step up from Ver. 1. New and enhanced commands round out this package into a system capable of handling data files from creation to being sources of information. If the size of your application is within the limits set by DB.BAS, then this product could be a cost-effective approach to solving your data manipulation problems.

**ZDM FOR CP/M USERS**
Published by **RD Software**
**1290 Monument Street**
**Pacific Palisades, CA 90272**
**(213) 454-8270**
Price: **$30.00 for program and manual**
**$ 3.00 for manual alone**
Formats: **8-inch IBM – Micropolis disk – Northstar Disk – Tapelib/Tarbell**
Reviewed by **Gene Head**

ZDM is a Z-80 debugger and monitor program very similar to the debugger distributed with the CP/M operating system, DDT. The only feature found in DDT and missing in ZDM is the in-line assembly command "A." Bob Doolittle of RD Software says future versions of ZDM will have this feature. He also noted that the actual release of future versions will depend on the response given to this edition. In-line assembly is actually a poor description for the "A" command in DDT. In reality, it is in-line patching using mnemonics. Assembly, to me, implies just that, ASSEMBLY. When I change or insert a line of source code and re-assemble a source file, everything gets moved and shifted appropriately. With the "A" command in DDT, I am merely patching using mnemonics. No actual assembly is taking place. My conclusion is the missing "A" command in ZDM is of little serious consequence.

The best feature of ZDM is its Z-80 capability. Additionally, ZDM surpasses DDT in the following areas:

1. Upon initialization, the user customizes the program to match the terminal's specifications.
2. Unlike CP/M's DDT, a long file that would overwrite (and crash) DDT is sensed by ZDM and an error message is issued while preserving ZDM. This is especially useful in smaller-sized systems.
3. More efficient Z-80 code reduces the program size and allows one more full page for the TPA.
4. ZDM recognizes both upper- and lower-case keyboard input.
5. ZDM displays extended 8080 mnemonics similar to TDL's.
6. Additional commands include:
   Display and alter both sets of Z-80 registers and flags.
   Enable or disable interrupts when entering the target program.
   Read from an input port and write to an output port.

HEX-MATH for computing relative off-sets, etc.

I utilized the ZDM on a North Star-based Z-80 system. It worked the first time. Initially, I didn't even bother to read the manual. I like software where the user is prompted intelligently, and ZDM passes this test. After PIPing a copy to a work disk (the original will never have the write protect tab removed, NEVER), I typed ZDM. It signed on with the request to enter the width of my terminal in hex. After that, the number of lines was requested. After these parameters were entered, ZDM politely asks, "Are you sure?" A positive response to this and the final message says "save 19 ZDM.COM'. Even I could do it! This is a one-time-only procedure. Now when I execute ZDM it is set to work with my terminal.

Exercising all the features, I found the program to function as advertised. The most noticeable difference between ZDM and DDT is in the List command. Watching a "disassembled" listing without the familiar "20 ??" is refreshing. I find the TDL-like mnemonics a bit unusual but I suppose that comes down to whatever you have become accustomed to. The HEX-MATH feature is nice when computing relative off-sets for patching. Being able to examine and alter the prime register set has uncovered some ideas concerning their usefulness that I have not experienced before.

There is a very small bug that has been discovered, and an equally small patch to correct the problem. There seems to be a problem when tracing through an EXAF instruction. The solution is to substitute a value of 9H at memory location 11C0H. ZDM can be used to modify itself as follows:

1. In CP/M command mode and the ZDM disk in the logged-in drive, type:
   ZDM ZDM.COM (CR)
2. After the ZDM prompt (-) appears, type:
   S11C0 (CR)
   9 (CR)
   . (CR)
   G0 (CR) G(zero)
3. Back in the command level, save the patched version by typing:
   SAVE 19 ZDM.COM (CR)

This will cure the EXAF tracing problem.

I am unable to compare ZDM with ZSID because I don't have a copy of ZSID. I have a copy of SID but no manual. I know ZDM has no symbolic capabilities, so a comparison here would be like comparing a skateboard to a motorcycle. They both get you where you want to go; one is just a bit slower and requires a bit more energy. On the other hand, the expense is $100 for ZSID and $30 for ZDM. So before you buy, consider how far you travel and how often. This should help you decide. Being cheap, I'd go with ZDM. It more than meets my requirements for a usable program development tool.

**Command Summary**

D – Display contents of memory in Hex and ASCII
DI – Disable Interrupt before executing target program
EI – Enable Interrupt before executing target program
F – Fill memory with byte value
G – Go execute target program, breakpoint optional
H – Display sum and difference of two hex values
I – Set FCB (File Control Block) with filename.ext
L – List Z-80 mnemonics by disassembling object code
M – Move block(s) of memory
QI – Get a value from an input port
QO – Put a value on an output port
R – Read the file as defined in the current FCB
S – Successively examine and optionally alter memory bytes
T – Trace a program through optional number of steps
U – Execute an optional number of program steps without display
X – Examine and optionally alter primary registers of Z-80 CPU
& – Same as X except operates on prime (alternate) registers; can also unconditionally exchange primary and prime registers

■ ■

*(Continued from page 22)*

At the completion of each pass, a summary is printed on the console in the form (all numbers are in hexadecimal, see Fig. 3).

**Structure of memory test**

Each pass through memory tests the ability to write and read ten different patterns. In order to trap errors in mapping to the physical address space, the pattern for the entire designated memory area is calculated and written before any location is read back.

#1 Cambridge pattern. A byte is either 00 or FF depending on its address.
#2 Address pattern. The contents of each byte is equal to the least significant eight bits of its address.
#3 Alternate ones and zeroes. Each byte is filled with the bit pattern 10101010.
#4 Address inverse pattern. Logical NOT of pattern #2.
#5 Alternate zeroes and ones. Each byte is filled with the bit pattern 01010101.
#6 All ones. Each byte has all bits set.
#7 Cambridge pattern inverse. Logical NOT of pattern #1.
#8 All zeroes. Each byte has all bits cleared.
#9 Bit crosstalk pattern. All bits in a given byte are set or reset except one. The immediately adjacent bytes contain the logical NOT of the given byte. For example, location 100=FE, 101=01, 102=FE. On each pass, the bit crosstalk pattern is rotated left one bit, so that in eight passes all possible patterns are used.
#10 Bit crosstalk inverse pattern. Logical NOT of pattern #9.

**Test availability on machine readable media**

For readers not wishing to retype the program source text, we will make the test available on an eight-inch, soft-sectored, single-density diskette including source, object, and documentation files. Check or money order for $25.00 should be sent to Laboratory Microsystems, 4147 Beethoven Street, Los Angeles, CA 90066. ■ ■

# DOBB'S Ex Machina

**Occasionally, a reader of *Dr. Dobb's Journal* will experience poor product performance or what he considers to be a company's failure to provide the quality of service they advertise. If his own honest attempts to resolve the situation fail, he may then take recourse to the impartial (if not heavenly) mediation of *Dobb's Ex Machina*.**

**When writing a complaint to us, please send backup data (e.g., carbons or photocopies of letters you have sent to the company in question, replies received, copies of registered receipts). It is not possible to cover all complaints received, but we will publish a representative sample. *DDJ* has no particular ax to grind with any seller or manufacturer, but we will pay close attention to those companies about which we receive numerous and consistent complaints.**

**Dear Sir,**

We have come to know that your magazine is providing assistance for readers who experience difficulties with software or hardware suppliers.

Therefore, as we did not receive any answer to our previous letters or Telex messages, we would like to request you to find out why Electrolabs is not supplying the software package for which they have received payment on or before March 26th, 1980. For your reference we are enclosing a copy of our last letter written to Electrolabs, which provides all details relating to our order.

Thanking you,
Sincerely yours,
Ulrich Blass
Aurelec Data Processing Systems
Aurelec, Prayogashala
Auroville, Kottakuppam 605104
India
9/4/80

P.S. The fact of so far having not received the software package, puts us in a difficult position, as we are answerable to the Indian Foreign Exchange Control authorities.

*Letter written by Mr. Blass to Electrolabs:*

**Gentlemen:**

We are very surprised to find out that so far you neither did answer our letter dated June 9th nor our Telex dated July 17th where we reminded you to ship the software package OS-1 without further delay.

Please note that as early as March 7th we have sent to you a cashier's cheque for the full amount of US$238 against the cost of your OS-1 operating system.

On April 10th we received your order acknowledgement together with a software license which was returned to you on April 19th duly filled in.

As subsequently we did not receive the shipment we sent a reminder on June 9th and a Telex on July 17th.

As by today we still did not receive any communication from you, we have to assume that either the OS-1 is not yet developed or Electrolabs went out of business.

Therefore we have chosen to contact a leading American microcomputer magazine to assist us to receive a software package or a refund of US$238.

Anticipating an immediate reply,
Yours sincerely,
Ulrich Blass
Aurelec Data Processing Systems
9/1/80

DDJ *sent a letter and a copy of the above correspondence to Electrolabs on 9/22/80. When no reply was received, we mailed a follow-up letter on 10/24/80. About one week later, we called their offices and learned that orders for OS-1 had been turned over to Software Labs, who informed us that Mr. Blass' order had been shipped. His letter to us, following, notes a few of his observations.*

**Dear Marlin,**

Thank you very much for your letter of the 24th October. As already communicated . . . we have now finally received the software package ordered from Electrolabs. I thank you very much for your very efficient help in this matter.

Apart from the software we did not receive any letter from Electrolabs and the long delay in delivery was not explained.

I give below my appreciation of the OS-1 Operating System, which Electrolabs claims to be a UNIX-like DOS with the added feature of CP/M compatibility. We cannot state that this claim is false, but still the OS-1 is much less useful than anticipated.

First of all, OS-1 is only CP/M 1.4 compatible.

Secondly, to run a CP/M program a sort of utilitiy – the CP/M adapter – is automatically loaded, which together with OS-1 leaves less than 12K to the user in a 48K system, obviously not sufficient to run any larger CP/M program (ZSID and Wordmaster, for instance, do not even load).

Further, the documentation is still incomplete and, for example, all information regarding the adaptation of OS-1 to a different mass storage device than an IBM 3740 Diskette (double density, hard disk, etc.) is not provided but promised for a later date.

So for the time being, OS-1 might be an interesting tool to experiment with, but in its present state of development, it certainly cannot claim to replace either CP/M 2.0 nor CP/M 1.4. However, the general architecture seems to be rather promising, but we would really appreciate to receive a complete and finished product rather than a prototype.

I hope that this information might be of some use to your readers. Should you receive any further communication from Electrolabs, we would appreciate very much to receive a copy.

Thanking you again,
Sincerely,
Ulrich Blass
Aurelec
11/11/80

*Editor's note: As one of Electrolabs' creditors, People's Computer Company recently received notice that MKW, Inc., dba Electrolabs, has filed an order for relief with the United States Bankruptcy Court under 11 U.S.C. Chapter 11.*

*John White, President of Software Labs informs us that his is a separate business which purchased and is fully supporting OS-1, with updates due by early January 1981. Co-owner of the company is Cachun Lee, the original designer of OS-1.*

■ ■

## PAN

*(continued from page 17)*

**The PAN Protocol:**

1) The originating PAN dials the number of the answering PAN.
2) When the answering PAN detects ring, it goes off hook and sends carrier.
3) When the originating PAN detects carrier it sends the string "@@@ANN" where ANN consists of a single alpha character followed by two numerics. This string is the version number of the answering PAN; for the CBM PET version of PAN it is P10. For experimental versions, A is X.
5) When the originating PAN receives 3 or more "@" characters it sends five CTRL-S characters.
6) When the answering PAN receives one or more CRTL-S characters it prepares to accept subsequent received characters as a PAN message. If no CTRL-S characters are received the answering PAN times out and goes on hook.
7) The originating PAN sends the message text; the last character of the message text should be UP ARROW (↑) to serve as a human readable end-to-message mark. The first line of the message text (between the first character and the first carriage return) contains the phone number of the originating PAN in the form (AAA) NNN-MMMM where AAA is the area code and NNN the prefix. It is desirable but not required that this first line also contain the sender's initials and the originating PAN's local time of transmission in twenty-four hour format (HHMMSS where HH ranges from 00 to 23).
8) The originating PAN sends five CTRL-R characters to signal end-of-message; it then goes on hook.
9) When the answering PAN receives one or more CTRL-R characters it goes on hook and places the received character string in the message buffer. If the answering PAN loses carrier at any time during message reception, it goes on hook.

*Correspondence and inquiries about the Personal Computer NETwork should be accompanied by a business-size, self-addressed stamped envelope and mailed to PCNET, P.O. Box E, 1263 El Camino Real, Menlo Park, CA 94025.*

■ ■

## Letters

*(continued from page 7)*

ccasr, however, shift before testing for completion, so a shift of zero is done wrong. I corrected this by moving the test to the top of the loop as follows:

```
; Shift DE arithmetically right by HL
and return in HL
ccasr:  XCHG
        DCR     E
        RM
        MOV     A,H
        RAL
        MOV     A,H
        RAR
        MOV     H,A
        MOV     A,L
        RAR
        MOV      L,A
        JMP     ccasr+1
; Shift DE arithmetically left by HL
and return in HL
ccasl:  XCHG
        DCR     E
        RM
        DAD     H
        JMP     ccasl+1
```

I would like to applaud Ron Cain's impressive effort in writing the Small-C compiler and thank *DDJ* for publishing it.

Yours truly,
James L. Colvin Jr.
3394 E. Styles Ave.
Camarillo, CA 93010

### Byte-Count Routine

Dear Marlin,

A letter from Ralph Kelley has called to my attention the 6502 byte-count routine used in his article published in the Jan/Feb 1980 issue of *COMPUTE*. This is similar to, but more code-efficient than, Jim Butterfield's CNTBYT routine that I listed in my own article in *DDJ* #22 (February 1978). The first 17 bytes are functionally identical, "weeding out" 11 opcodes with unusual bit-patterns. At that point, Ralph noted that the residual opcode bit-patterns had become $10 for all one-byte codes, less than $10 for all two-byte opcodes, and greater than $10 for all three-byte opcodes. This makes it possible to replace the four instructions used by Butterfield with the following three: CMP $10, BEQ ONE, BCC TWO. If coded as a subroutine, it needs 26 program bytes instead of 28 and also runs about two cycles faster than CNTBYT. However, my 28-byte non-linear BYTCNT routine, also listed in *DDJ* #22, runs significantly faster, especially in analyzing the two-byte codes that tend to be predominant in 6502 programs.

*DDJ* and I are responsible for making the byte-count problem a 6502 machine-coding-skill game. The usual fate of algorithms is to be anonymously interred within a tumulus of code, and this must be true of byte-count routines for other microprocessors. The circuitry of the 6502 is so intricate that a software emulation becomes an intriguing puzzle, which is also very useful. Although there are only 3 possible counts, each one is generated by 2 distinct opcode bit-patterns, so that every solution must isolate 6 classes, requiring 5 branch instructions. Each branch must be preceded by at least one bit-analysis instruction, and this minimum of 5 was achieved in my BYTNUM routine (*DDJ* #17), but code-efficiency was not optimal since 4 of them were 3-byte BIT instructions. Butterfield's masterly solution used 7 bit-analysis instructions, which Kelley has ingeniously reduced to 6. My BYTCNT routine also uses 6, but since one of them is a BIT it cannot be as code-efficient. If the 6502 had a BIT-immediate instruction, BYTNUM logic could optimize both code- and time-efficiency. As things are, a code vs. time trade-off is inevitable.

What I stressed in *DDJ* #22 may be worth noting, that although all these routines exit with the correct byte-count in the X-register, their bit-analysis instructions also alter the status flags in different ways, such that even more than 6 opcode types may be identifiable. So far, no one seems to have used this auxiliary information, but it ought to be kept in mind.

Sincerely,
H. T. Gordon
College of Natural Resources
University of California
Berkeley, CA 95720

Dr. Dobb's Journal *welcomes letters to the editor as a forum for ideas, innovations, irascibility and even idiosyncrasies. Some letters may be edited for clarity and brevity. The Doctor likes hearing from you – keep on writing!*

■ ■

# Of Interest

## Calls For Papers

Papers are now being accepted for the Fifth Western Educational Computing Conference to be held in San Francisco on November 19-20. Topics are computers and computer applications in any area of interest to instructors and administrative personnel who deal with computers at the college or university level. Deadline is March 1. Write to Professor Elizabeth Alpert, Computer Science Dept., Hartnell College, 156 Homestead Ave., Salinas, CA 93901.

"Reliability in Electrical and Electronic Components and Systems" is the theme for Eurocon '82, an interdisciplinary conference on reliability and risk assessment, to be held in Copenhagen. Preliminary titles and one- to two-page summaries of papers are being accepted for the event to take place June 14-18, 1982. Contact the Conference Office, Danish Engineers' Postgraduate Institute, The Technical University of Denmark, Bldg. 208, DK-2800 Lyngby, Denmark.

## Computer Aid For Handicapped

John Hopkins University is conducting a nationwide search for ideas and inventions to aid the handicapped through computer technology. The search is funded by grants from the National Science Foundation and Radio Shack. A $10,000 grand prize and 100 other national regional awards have been established, including fifteen personal computer systems. Entries may be a device, system or a computer program. Major objectives of the contest are to focus computing technology on the needs of the handicapped and to harness individual innovation and creativity on a nationwide basis. Information is available from Personal Computing for the Handicapped, John Hopkins University, P.O. Box 670, Laurel, MD 20810.

## Miscellany

*Tiny-c two* runs ten times faster than the original tiny-c, and includes new features like 32-bit integers, new operators, redirectable and direct access I/O, a UNIX[tm]-style command interpreter and more. The Owner's Manual costs $50, the diskette is $200 from tiny-c associates, P.O. Box 269, Holmdel, NJ 07733. . . . *Hardware and software for Heathkit computers* is available from a number of suppliers not affiliated with the Heath Company, and who have specialized applications plus general interest products not yet offered by Heath. For a directory, send U.S. $7.50 to Buss, 325-D Pennsylvania Ave. S.E., Washington, DC 20003. . . . Hewlett-Packard, realizing it is software that will sell its machines, has announced a number of new utilities including a *dumb terminal emulator*. For specifics, write to the Inquiries Manager at HP, 1507 Page Mill Road, Palo Alto, CA 94301. . . . Dilithium Press' new catalog lists over *sixty-five books on microprocessing*. Also included is software for the TRS-80 Level II and the 8K PET. For a free copy, send to 30 N.W. 23rd Place, Portland, OR 97201. . . . Sorcim has introduced a version of their *Pascal/M for the Intel 8086 CPU*, and *TRANS 86*, a program that "translates existing Z-80 or 8080 code into 8086-8088 code." The press release neglected to state whether the translation is done in assembly language or at the level of a hex file. If you work with the 8086 and check this out, send a review to *DDJ*. . . . For the benefit of readers abroad, here are the addresses of two firms that now represent Digital Research in *the Far East and Europe*:

Microsoftware Associates
102 Plasada
3-16-14 Minami Aoyama
Mnato-ku
Tokyo 107, Japan
03-403-2120
Telex 2426875 MSA

Vector International
Research Park
B-3030 Leuven
Belgium
32 (16) 20-24-26
Telex 26202 VECTOR

For a Datapro report comparing 210 *alphanumeric display terminals*, and the results of a survey covering over 18,000 installed terminals, send $15 to Datapro Research Corporation, 1805 Underwood Blvd., Delran, NJ 08075.

## New Products

Hewlett-Packard has announced a plan for would-be *software entrepreneurs*. The company will market non-HP software for their Series 80 machines through a catalog distributed to all HP dealers, and expects to help promote applications software in a large variety of fields. Developers of quality software for micros would be natural participants. HP will work with software developers to expand the amount of software, and all products listed in the catalog will be available from HP dealers. . . . In a related announcement, HP has offered an *Assembler ROM for the Series 80* machine. The $295 ROM contains an assembler and an editor. One interesting feature is that the editor checks each line for correct assembly-language syntax as the line is typed. A programmer using a Series 80 assembly language will be able to redefine BASIC keywords, add new BASIC commands and functions. Assembly-language programs will have full access to all routines in the BASIC operating system, including those for math, CRT control and general I/O. The assembler ROM is documented by a 400-page manual. . . . The Microperipheral Corporation recently introduced a *new line of direct connect modems* for the TRS-80, ATARI, APPLE and virtually any terminal or computer with provision for the RS232 serial I/O at 300 baud. Prices start at $199.95, with the AUTODIAL/AUTOANSWER option at $79.95. Write them at 2643-151st Place N.E., Redmond, WA 98052. . . . Personal Software Inc. has given Commodore International Ltd. exclusive license to reproduce, sell and distribute *Visicalc outside the U.S.* in German, French, Spanish, Italian, Dutch and Danish. . . . *Direct input of temperatures* is possible for a variety of micros using TEMPSENS by Connecticut Microcomputer, Inc. Cost for a two-probe unit is $49.95. Address inquiries to 34 Del Mar Drive, Brookfield, CT 06804. . . . *The Cybervision personal computer* is the first major product of United Chem-Con's new Home Products Division. It is based on the RCA 1802 chip, accepts input from cassette or the two alphanumeric keypads which are provided (note that these are *not* the usual keyboard) and professes color graphics and human quality audio. The price tag is $400. For information write to United Chem-Con at 2950 Old Tree Drive, Lancaster, PA 17603. . . . *GROUP/380 is a microcomputer interest group* for micro systems equivalent to IBM's 360/370 mainframes. The group links potential users of these systems to the large amount of software available from organizations either for a copying charge or at micro prices. Contact Mokurai Cherlin, GROUP/380, P.O. Box 1131, Mt. Shasta, CA 96067. . . . DOS/65 is an operating system which provides features for 6502's similar to those available to 8080's and Z-80's from CP/M[tm]. Prices range from $100-$150.

For details mail a self-addressed, stamped envelope to Richard A. Leary, 1363 Nathan Hale Dr., Phoenixville, PA 19460. . . . *6800 software runs on Apple II's* with THE MILL processor board plugged into any peripheral slot. Its heart is the Motorola 6809E processor. Available from Stellation Two, P.O. Box 2342, Santa Barbara, CA 93120. Priced at $275. . . . An *8-inch floppy subsystem for North Star* requires only plug-in installation. The software-hardware package is priced at $1910, with the software alone costing $150. From John D. Owens Associates, Inc., 12 Schubert St., Staten Island, NY 10305. . . . Eventide's *memory expansion board for HP 9845* desktop computers claims to be identical in operation and installation to the board supplied by HP, but costs only $2250 (one-half of HP's price). Each WMAP-1 board adds 128 kilobytes of memory. Write 265 West 54th St., New York, NY 10019. . . . The M24SS static RAM card for SS-50/SS-50C bus computers is available in 8-, 16-, and 24-kilobyte configurations from Percom Data Company, 211 N. Kirby, Garland, TX 75042. . . . KIMSET1 is an *analog-to-digital conversion system for KIM microcomputers*, providing 16 channels of input, each with a resolution of 8 bits. Plugging this into the KIM applications port enables it to read and process temperature, velocity, pressure, db, pH, acceleration, humidity, light and fluid levels, etc. Price is $285 from Connecticut microComputer Inc., 150 Pocono Road, Brookfield, CT 06804. . . . *A diagnostic software package for Cromemco systems* exercises, tests and reports any errors in the operation of floppy or hard disk drives. Information available from Cromemco, 280 Bernardo Ave., Mountain View, CA 94043. . . . Nestar System, whose Cluster/One makes a local network out of a dozen Apples, has announced an office mail program they call The Messenger. Network users may send, receive, file, make notes on, copy or broadcast electronic analogs of office memos from, to, at and by each other. The Messenger costs $2250; call Nestar at (415) 327-0125.

■ ■

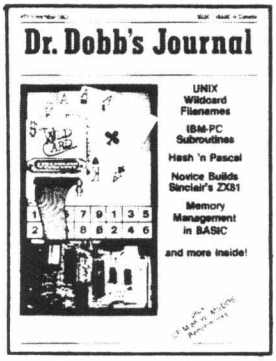

# Dr. Dobb's Journal

For Users of Small Computer Systems

Each issue includes:
- valuable software tools
- algorithms & problem solving
- industry news
- important product reviews

With in-depth coverage of:
- telecommunications
- systems programming
- language development
- machine independent programs

and much, much more!




Yes! Please enter my subscription for
☐ 1 yr. (12 issues) $25 ☐ 2 yrs. $47 (save $13 off newsstand)
☐ Please bill me ☐ I enclose check/money order

Name_____

Address _____

City_____ State_____ Zip_____

**Dr. Dobb's Journal, A Publication of People's Computer Company**
**1263 El Camino Real, P.O. Box E, Menlo Park, CA 94025**

## Are you ready?

*DDJ*, the world's foremost microcomputer publication, has been working for years to prepare its readers to be innovators, to lead the wave of breakthroughs in our changing technology.

Every issue of *Dr. Dobb's Journal* helps one to understand the nuts and bolts of small computer systems. We offer entire listings of valuable software: our pages have included compilers, cross-assemblers, editors, new languages, hardware interfaces and more – usually before anyone else thinks of them!

## Even more important!

Our subscribers share insights, correspond, and contribute to one another's work, more than any other group we know. They treat *Dr. Dobb's Journal* as a "hands-on" publication.

This warm cooperation has done more to refine software products, and generally to advance the state of microcomputer technology, than perhaps any other resource. And it is available to you through our pages!

## For the straight facts . . . . . .

If you are a serious computing professional or enthusiast, then you should take a very close look at what *DDJ* offers you. We've been on the cutting edge since 1976.

# Book Reviews

**TRS-80 INTERFACING, BOOK 2**
By **Johathan A. Titus, Christopher A. Titus, and David G. Larsen**
Published by **Blacksburg Continuing Education Series**
**254 pages, $9.95**
Reviewed by **Steven Wexler**

This book introduces interfacing techniques which will allow the reader to utilize the TRS-80 (Models I, II, or III) for control purposes. The authors do not attempt cradle-to-grave descriptions of hardware projects, preferring to concentrate on technique. The actual use of the circuits described is left up to the reader.

Some of the topics covered are: driving high-current and high-voltage loads, generating voltage and current signals, measuring unknown voltages and currents, remote control techniques, and new interface/controller chips.

If you do not have a hardware background, read *TRS-80 Interfacing, Book 1* before attempting to read this book. Those of you comfortable with the typical magazine hardware article should not have any trouble.

*TRS-80 Interfacing, Book 2* is a reasonably good textbook (or reference book) at a fair price. If it fits your needs, by all means buy it.

**Z8000 ASSEMBLY LANGUAGE PROGRAMMING**
By **Leventhal, Osborne and Collins**
Published by **Osborne/McGraw-Hill**
**604 pages, $19.99**
Reviewed by **John B. Palmer**

This is the first book on Z8000 programming to cover everything. There is enough good info to make two regular size books. I am glad the publishers released it as one thick volume.

This comprehensive Z8000 reference work sets new standards of quality for microprocessor documentation. At $19.95 it is very reasonable if compared to more expensive (but not better) software literature sold by some leading semi-conductor makers.

Seventeen chapters give a well-rounded coverage of Z8000 assembly language programming and closely related subjects. The most important points get enough attention. A new programmer should be able to start producing useful programs after reading chapters 1, 2, 4 and 5.

Chapter three dominates the book with 376 pages devoted to the massive Z8000 instruction set. (If you don't want to learn the entire instruction set, skip over to chapter four.) It is a joy to see at least a full page of text for each instruction type. Important instructions may have four or more pages of text. Lots of clear, eye-pleasing diagrams are used here and in the rest of the book.

I was pleased to see even a few pages on floating point (I am weak in this area). But I was sad to find very little about multitasking, which I have done on an 8080.

About 100 pages of the book deal with the modern philosophy of programming. It stresses program design, debugging and documentation. Flow charts and structure are endorsed. Compare this with ninety-six pages on I/O and half that many for interrupts. The text tends to favor the novice programmer.

Really, at times the tutorials seem overly helpful. But I think the authors made the right choice. More esoteric stuff can be found in the many excellent references to other literature. There are twenty-five such references just in the chapter on I/O. If you can not find the answer in this book, at least you will be told where you can find what the authors may have set aside.

This book does not ignore an important Z8000 weakness. Instead, it helps the user identify which early production chips failed to properly execute which instructions.

The last two chapters give practical, real world Z8000 application programs. Not just source codes, but all the design steps a software engineer would need to carry out his job.

This book has one unavoidable failure: it should have been printed one year earlier. But that is more Zilog's failure than the authors'. Because Zilog was late in getting the Z8000 to the market, few groups had running systems with which to test software designs.

Overall, the book is very practical and reflects a modern, conservative approach toward 16-bit microprocessor programming.

**COMPUTER PERIPHERALS FOR MINICOMPUTERS, MICROPROCESSORS & PERSONAL COMPUTERS**
By **C. Louis Hohenstein**
Published by **McGraw-Hill**
**312 pages, $19.50**
Reviewed by **Susan Bowers**

This book was a very pleasant surprise – a technical book written in a *readable* manner. A great many charts, pictures and illustrations are included, which help clarify the material. Exactly enough background information is given to enable the reader to understand the basic principle and best use of each type of peripheral.

*Computer Peripherals for Minicomputers, Microprocessors and Personal Computers* is divided into five major parts. The first is an introduction, with the selected fundamentals of computers necessary for an understanding of peripherals. The remaining sections deal with: Output Peripherals; Input Peripherals; Memory Peripherals; and Data Transmission, Remote Terminals and Interfaces. The several kinds of equipment in each section are thoroughly covered. Included are comparisons and discussions of all the types in each category, with drawings and good illustrations of how they work. The clearly stated explanations are sufficient to provide a basis for choosing which devices would be most useful in specific circumstances.

There are so many peripherals it would have been easy to lose the reader in a confusing maze of technical terms. The author avoids this nicely, charting a clear path to comprehension. He makes a complex subject seem simple. This is an excellent resource book for those who are learning about peripherals, and would be extremely useful to someone planning to purchase a small computer system.

**CRT CONTROLLER HANDBOOK**
By **Gerry Kane**
Published by **Osborne/McGraw-Hill**
**184 pages, $6.99**
Reviewed by **Anthony Skjellum**

*CRT Controller Handbook* describes five LSI cathode-ray tube (CRT) controller chips. It contains details of operations, and the numerous figures are supplemented by data sheets. An introductory chapter explains the concept of CRT control and the technical presentation is quite good throughout the book. The text concentrates on microprocessor-based CRT systems which operate in conjunction with an LSI controller device. Pertinent topics such as timing, memory interfacing and contention logic, character generation and video display generation are discussed.

CRT designers who plan to use an LSI CRT chip will benefit from this book since it provides a good comparison of

features and notes pitfalls. This book will also serve as a useful guide when choosing a device for your application. The chips discussed are the DP8350, 8375, 6845 and 5027/9927. The book deals almost exclusively with alphanumeric CRT's although graphic terminals are mentioned in passing. While the experienced designer will appreciate the information provided here, case studies and/or specific examples would have improved the book considerably. Finally, no detailed attention is given to software which would relate to microprocessor-based CRT systems.

This is not a tutorial book and is certainly not for beginners. Its total content will be mostly of interest to designers although certain sections are of general interest as well.

■ ■


DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

**Subscription Rates:** $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. dollars, drawn on a U.S. bank.

**Writer's Guidelines:** All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

**Donating Subscribers**

**Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives five one-year subscriptions).

**Contributing Subscribers:** Robert L. Edens, Eric P. Hanson, J. G. Zeratsky, Mervin E. Frank, Edward W. Carr, Steven Fisher, Thomas H. Grohne, Richard Washburn, Les Hancock, G. V. Elkins, C. A. Forsberg, Daniel R. Lunsford, J. D. Griffith, Michael Slade, Thomas M. Schaefges, General Electric, Louis L. Odette, Robert M. Connors, William J. McEown, Robert C. Luckey, Kim Leary, Robert N. Stabler, Dewitt S. Brown, John B. Palmer, E. Glion Curtis.

**Retaining Subscribers:** Mike Faithful and Osamu Shimizu.

**Sustaining Subscribers:** Gordon Eubanks, Jr. and Robert C. Birss, Ph.D.


# The Electric Phone Book

A computerized bulletin board works just like an ordinary bulletin board system except that instead of paper and thumbtacks it uses a terminal, a computer, and the dial-up telephone network.

The list below was developed from several sources including the Peripheral People in Mercer Island, Washington, and the People's Message System in Santee, California. It is being maintained by People's Computer Company's PCNET project, our effort to bring computers and telecommunications into the hands of everyone. While this is the most complete listing we have as of this writing, we would appreciate additions and corrections. Send them to PCNET, PCC, P.O. Box E, Menlo Park, CA 94025.

All the bulletin board systems listed here can be accessed by telephone using a 300-baud ASCII terminal and a Bell 103 modem. Most use carriage-return as a speed recognition character, after which they are self-teaching. All are free to anyone who calls, unlike Arpanet, which is restricted, and The Source and MicroNet, which cost money. The list has been sorted by area code; consult your local telephone directory for geographical correspondence. (Also printed in *Recreational Computing*, January-February 1981.)

**(201) 283-2724**
(201) 457-0893
**(201) 688-7117**
(201) 753-1225
**(201) 753-8152**
(201) 835-7228
**(201) 843-4563**
(201) 874-6833
**(201) 891-7441**
(201) 968-1074
**(202) 337-4694**
(202) 635-5730
**(203) 348-6353**
(203) 357-1920
**(203) 746-4644**
(205) 945-1489
**(206) 244-5438**
(206) 246-8983
**(206) 482-5134**
(206) 482-5590
**(206) 524-0203**
(206) 546-6239
**(206) 723-3282**
(206) 937-0444
**(209) 638-6392**
(212) 245-4363
**(212) 448-6576**
(212) 787-5520
**(212) 997-2186**
(213) 276-4276
**(213) 316-5706**
(213) 329-3715
**(213) 340-0135**
(213) 346-1849
**(213) 349-5728**
(213) 360-6332
**(213) 394-1505**
(213) 395-1592
**(213) 396-3905**
(213) 424-3506
**(213) 428-4718**
(213) 459-3177
**(213) 459-6400**
(213) 566-8035
**(213) 631-3186**
(213) 657-8803
**(213) 673-2206**
(213) 675-8803
**(213) 709-5423**
(213) 795-3788
**(213) 787-4004**
(213) 799-1632
**(213) 799-6514**
(213) 826-0325
**(213) 828-3400**
(213) 843-5390
**(214) 288-4859**
(214) 634-2668
**(214) 634-2775**
(214) 641-8759
**(216) 644-1965**
(216) 754-7855
**(301) 344-9156**
(303) 759-2625
**(303) 789-0936**
(305) 261-3639
**(305) 566-0805**
(305) 689-3234
**(305) 772-4444**
(305) 821-7401
**(305) 989-9647**
(309) 688-0470
**(309) 694-6531**
(312) 255-6489
**(312) 269-8083**
(312) 337-6631
**(312) 420-7995**
(312) 622-8802
**(312) 528-7141**
(312) 545-8086
**(312) 622-9609**
(312) 767-0202
**(312) 782-8180**
(312) 782-9751
**(312) 941-9009**
(312) 964-7768
**(313) 288-0335**
(313) 357-1422
**(313) 465-9531**
(313) 477-4471
**(313) 484-0732**
(313) 569-2063
**(313) 588-7054**
(314) 838-7784
**(316) 746-2078**
(319) 353-6528
**(319) 557-9618**
(404) 394-4220
**(404) 733-3461**
(404) 790-8614
**(404) 793-1045**
(404) 939-1520
**(404) 939-8429**
(404) 953-0723
**(405) 353-2554**
(405) 528-8009
**(408) 241-1956**
(408) 296-5799
**(408) 263-0248**
(408) 263-9650
**(414) 241-5406**

**(414) 241-8364**
(414) 282-8118
**(415) 348-2139**
(415) 348-2396
**(415) 493-7691**
(415) 527-0400
**(415) 647-9524**
(415) 661-0705
**(415) 683-4703**
(415) 792-8406
**(415) 851-3453**
(415) 948-1474
**(417) 862-7852**
(419) 865-1584
**(502) 245-8288**
(502) 896-9624
**(503) 646-5510**
(512) 657-0779
**(513) 671-2753**
(513) 874-2283
**(515) 279-8863**
(516) 938-9043
**(523) 223-3672**
(602) 866-0258
**(602) 956-5612**
(602) 957-4428
**(602) 957-9282**
(604) 687-2640
**(607) 754-5571**
(607) 797-6416
**(609) 983-5970**
(612) 561-6311
**(612) 929-8966**
(614) 272-2759
**(614) 649-7097**
(615) 254-9193
**(617) 354-4682**
(617) 388-5125
**(617) 431-1699**
(617) 649-7097
**(617) 692-3973**
(617) 864-3819
**(617) 897-0346**
(617) 963-8310
**(702) 826-7234**
(702) 873-9491
**(703) 281-2125**
(703) 281-2222
**(703) 379-0303**
(703) 620-4990
**(703) 734-1387**
(703) 750-0930
**(703) 893-9474**
(703) 978-7561
**(707) 448-9055**

**(713) 233-7943**
(713) 693-8080
**(713) 977-7019**
(714) 549-5689
**(714) 463-0461**
(714) 495-6458
**(714) 526-3687**
(714) 537-7913
**(714) 565-0961**
(714) 571-5550
**(714) 582-9557**
(714) 730-1206
**(714) 739-0711**
(714) 751-1422
**(714) 772-8868**
(714) 898-1984
**(714) 952-2110**
(714) 962-7979
**(714) 963-7222**
(801) 375-7000
**(801) 466-1737**
(801) 753-6800
**(802) 748-9089**
(802) 879-4981
**(803) 270-5372**
(803) 270-5392
**(803) 279-5392**
(803) 771-0922
**(803) 772-1592**
(804) 340-5246
**(805) 484-9904**
(805) 527-9321
**(805) 682-7876**
(805) 964-4115
**(806) 355-5610**
(813) 223-7688
**(816) 523-9121**
(816) 531-1050
**(816) 861-7040**
(816) 931-3135
**(817) 855-3916**
(817) 855-3918
**(817) 923-0009**
(901) 276-8196
**(901) 362-2222**
(901) 761-4743
**(902) 794-8198**
(904) 243-1257
**(904) 243-8565**
(913) 362-6398
**(913) 764-1520**
(913) 782-5115
**(915) 584-5393**
(916) 393-4459
**(918) 224-5347**

# The Radio Shack Pocket Computer

## by Dick Greenlaw

*Dick Greenlaw, 251 Colony Ct., Gahanna, Ohio 43230.*

The curiosity bug bit me after reading a review of Radio Shack's pocket computer in the December 1980 *Interface Age*. That review gives several examples, so I will stick to the surprises.

There are 26 variables plus an array. The array includes the 26 variables and continues through however much memory may be left over. Any of these data locations can be treated either as a string or a floating point variable. Note, however, that it is still the same location and can't be used both ways at the same time!

Numeric precision is 10 digits plus an exponent of ± 99. Strings are limited to 7 characters. You start with 1424 bytes free plus the 26 regular data locations. Keywords fit in one byte. Data words are 8 bytes.

String variables can be INPUT to, PRINTed from and assigned or tested for equality to other string variables and string constants. However, you can't take them apart or combine them.

I ran a couple of benchmarks from old *Kilobaud* articles. The pocket computer is 25 times slower than my Micropolis BASIC, which is in turn fairly slow compared to Microsoft. That was a disappointment, but probably reasonable enough for a computer which runs 300 hours on four camera batteries and keeps all of its program and data memory alive even when it is turned off.

Radio Shack has crammed into a pocket-sized computer a very clever interpreter with excellent debugging aids and both line and character level editing. Although the display is only 24 characters wide, horizontal cursor controls allow program lines up to 80 characters long.

The optional cassette interface allows storage and retrieval of programs, data and "reserved" programs. It also allows verification of programs and "reserved" programs.

Programs can also CHAIN to other programs on the tape. Motor control is provided and most recorders can be used. Files obviously have to be in order, but they don't have to be consecutive! All cassette operations require a file name which is a string constant or variable. Forward file searching is automatic and you could request the user to rewind. Unfortunately, I haven't found any way to read or catalog a file with an unknown name.

"Reserved programs" are definitions of strings assigned to keys as an aid to program entry. Typing shift-key inputs the user-assigned string for the key.

String constants can also be used as labels by placing them just after the line number. Then GOTO "THERE" and RUN "PROGRAM" become legal! Single character labels can be used to run routines by name when in a special run mode. Again, the shift key gives other keys user-assigned meanings.

In summary, the pocket computer is very clever and should appeal to users of programmable calculators. It has trig and log functions and their inverses! However, it is a bit slow. Lunar lander takes several seconds to respond to each burn period. My plans of cramming a small checkers game into it appear a bit optimistic because of the speed, and just possible in space. (Remember, each data location uses 8 bytes.)

Documentation consists of a very detailed book of over 100 pages and a summary card that fits in the carrying case. Radio Shack states that the book is not for beginners and recommends some introductory books.

■ ■

### Sinclair's ZX80

*(continued from page 15)*

newline key if there is an error (a little IC bug may come and bite your finger if you try)! The newline key is inactive until after the inverted letter "S" goes away. But if you need to change a line that was entered earlier for some other reason, simple! Move the up or down cursor control until it is on that line. Then hold the shift key and hit the "edit/newline" key and watch what happens.

The power in this method of text editing is greater than what you expect. Do any of *Dr. Dobb's* readers remember the Control A trick in MITS extended BASIC 4.0? It was not fully documented then, and even now the Microsoft manual does not tell all. Something like that is in ZX80 BASIC, and it is explained clearly. Just look in the index of the operating manual (what, an index too?) under EDIT and then read page 26. What? You didn't find it on page 26? Sorry, try the last paragraph on page 90.

Yes, it is true. The edit command may be used to change the line number of a program, and the original line is still there! This is perfect for those programs where a dozen lines are duplicates or have only a few changes to be made. Think about it!

If I seem excited about this feature, I am! Do you really need a line editor for a very simple BASIC system? I think so. When the Radio Shack TRS-80 was first introduced about four years ago, it was very hard to program if you were not a perfect typist. The only way you could edit a program line was to re-type the whole line. If you got it wrong the second time, try again and again.

Of course, I cannot compare the Sinclair ZX80 at two hundred dollars with the new Radio Shack Color Computer at four hundred. But at least my Sinclair has a simple way to edit a line. According to Radio Shack, their TRS-80 Color Computer and the TRS-80 Level III (for seven hundred dollars), both use Level I BASIC. This means that anyone who programs either of these will come into the same snag that the first TRS-80 had four years ago. This is why I am so pleased with the edit mode of the Sinclair ZX80.

*JOHN B. PALMER, P.O. Box 23, Boonville, CA 95415. The author has two other home computers, both older and more costly than his new Sinclair ZX80. They are an Apple II Plus with dual drives, and an Altair 8800b with dual, 8-inch drives, 56K and CP/M. Before his recent move to small-town living, he spent two years designing software and hardware for 8080-based test equipment at Dysan Corporation in Santa Clara, CA.*

■ ■

$2.50 ($3.00 In Canada)

# DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

**FOR USERS OF SMALL COMPUTER SYSTEMS**
Number 53 March 1981 Volume 6, Issue 3

**Analysis of the Use of the 6502's Opcodes**
*Howard W. Whitlock, Jr.*

**ERGO: A "Thinking Program"**
*Sylvan Rubin*

**Byte Manipulation and Byte Pointers**
*Edwin E. Freed*

**Small-C: Current Status**
*P.L. Woods*

**Interfacing the TRS-80 Voice Synthesizer to Other Micros**
*Michael Hightower* and *Peter B. Maggs*

**and, once again, even more...**



P.O. Box E
Menlo Park
CA 94025

# DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Volume 6, Issue 3
Number 53
March 1981

# Contents

## Articles

**Computer-Based High Speed Reading** . . . . . 10
*by Dave Caulkins*
**Analysis of the Use of the 6502's Opcodes** . . . . . 11
*by Howard W. Whitlock, Jr.*
**ERGO: A "Thinking Program"** . . . . . 16
*by Sylvan Rubin* and *Sheldon Softky*
**Using WORD-MASTER With the Compal-80** . . . 18
*by Anthony Skjellum*
**Byte Manipulation and Byte Pointers** . . . . . 19
*by Edwin E. Freed*
**Small-C: Current Status** . . . . . 20
*by P. L. Woods*
**DOSMOVER for the North Star** . . . . . 40
*by Thomas W. Cage*
**Video Underline Modification** . . . . . 42
*by Dannie E. Davis*
**Interfacing the TRS-80 Voice Synthesizer to Other Micros** . . . . . 46
*by Michael Hightower* and *Peter B. Maggs*

## Departments

Letters . . . . . 5
Book Reviews . . . . . 15
Software Reviews . . . . . 38
Of Interest . . . . . 50
Electric Phone Book . . . . . 52

**Publisher**
Ann Merchberger
**Editor**
Marlin Ouverson
**Contributing Editors**
Dave Caulkins
Dave Cortesi
**Art Director**
Clifford West
**Artist**
Barbara Ruzgerian
**Typesetter**
Renny Wiggins
**Proofreader**
Nancy Heubach
**Marketing Director**
Gavin W. Cullen
**Advertising Director**
Michael Madaj
**Circulation Manager**
Peter Clark
**Circulation Assistant**
Leah Dansby

© 1981 by People's Computer Company unless otherwise noted on specific articles. All rights reserved.

*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025.

Subscription Rates & Writer's Guidelines on page 18.

# Editorial

The kinetic energy which abounds at People's Computer Company can be dismaying to new-found friends and occasional visitors. Our small offices are charged with dynamics too rarely found in typical business settings. A few steps through our miniature maze of corridors leads the unwary from the home of two international magazines to the seat of PCC's computer literacy project (ComputerTown, USA!) and to the focus of PCNET's protocol development. Paths lead in several directions to the realms of marketing, sales, accounting, production and circulation.

On any day at PCC one may meet government officials and hackers, heads of state and solid state genii. Well, perhaps not heads of state. They come to meet Dr. Dobb's consortium. Thus cross the paths of the mighty and the humble (well, not *that* humble . . .).

It is the diversity which is so astounding, and the tangible evidence that each component is interactive with the whole. Each part functions under the intense scrutiny of Doc (peering through his bifocals) with the help of his workers and volunteers. The juxtaposition of various personalities and projects at PCC creates a homogenous kind of spontaneity (even the definition defies linear logic!).

One of the guiding hands of PCC once described this as a place where a thousand lillies may blossom. We allow the entrance of diverse elements into our schematic – the strong ideas thrive, the weak ones fade into the sunset. *Gigo!* This is the course nature follows. PCC and its publications do not censor, but provide a proving ground. Success or failure should be inherent and not a matter of bias.



Marlin Ouverson
Editor

# Letters

Dr. Dobb's Journal *welcomes letters to the editor as a forum for ideas, innovations, irascibility and even idiosyncrasies. Some letters may be edited for clarity and brevity. The Doctor likes hearing from you – keep on writing!*

*DDJ's* responsive readership can give you invaluable feedback on your ideas and innovations. Find out how you can reach them by contacting us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

## A Flare for FORTH

Dear *DDJ*:

Issue Number 51 arrived today, and I found Dr. Gordon's article ("What is FORTH?") interesting but confused.

Your production department ran into some sort of proofreading problem about 1/3 of the way down the lefthand column of page 12, producing some confusion about what was meant (*see box this page. – Ed.*).

Dr. Gordon's basic problem is that he does not have a running FORTH system, and Bruce Airy's NOVA implementation is seriously out of date. I suspect it was done from a very early FORTH model, and limited by the machine.

If Dr. Gordon could specify just what kind of computer he has maybe we at FIG could provide him with a system to work with, which would relieve some of his confusion.

Having a FORTH system, I tried out his FIBO program, and found that it does indeed work, but leaves garbage on the stack when it finishes.

The two numbers left are the last two Fibonacci numbers, which are waiting to be used to calculate the next one; but the program exited from the loop, leaving them behind. In order not to mess up the programs which follow, every routine must leave the stack exactly as it found it. This means that any values it puts on the stack while it is running must either be used up, or DROPped when it finishes execution.

The attached listing of Screen #50 shows a cleaned-up version of FIBO, called FIBONACCI (figFORTH supports names with up to 31 characters), which is a bit more general – it takes a parameter on the stack which tells it how many of the Fibonacci numbers you want it to calculate.

Sincerely,
S. B. Bassett
Guest Editor, FORTH Dimensions
c/o 151 University Avenue
Palo Alto, CA 94301

```
        OK
  OK
  OK
50 LIST
SCR # 50

( SCR#50DEC – FIBONACCI NUMBERS )

: FIBO 0 1 11 1 DO
   CR DUP .
   SWAP OVER +
   LOOP ;

: FIBONACCI ( # OF FIB.NOS. WANTED – > )
   0 1          ( SET UP FOR CALCULATIONS )
   ROT          ( GETS # WANTED )
   0 DO         ( SET UP LOOP – 0 IS LEGAL )
      CR        ( GO TO NEXT LINE )
      DUP .     ( DUP TOP # & PRINT )
      SWAP OVER ( SWITCH FOR ADDITION )
      +         ( ADD 2 TOP NUMBERS )
   LOOP         ( GO BACK FOR NEXT NUMBER )
   DROP DROP ( CLEAN UP THE STACK )
;


 OK
50 LOAD 50  OK
  OK
  OK
```

```
FIBO
1
1
2
3
5
8
13
21
34
55  OK
. 89  OK
. 55  OK
. 0 EMPTY STACK
  OK
  OK
  OK
10  OK

FIBONACCI
1
1
2
3
5
8
13
21
34
55  OK
. 0 EMPTY STACK
. 0 EMPTY STACK
```

## . . . another sad story

Dear *DDJ*/Ex Machina:

Regarding the Hazeltine terminal letters in your Jan. 81 issue: Over the last three years, our lab has purchased eight Hazeltine 1500s. The first was selected for its price ($860) in mid-1977. In early 1977 we bought another, and in mid-78 we bought two more.

At that time we also purchased the service manual, since we intended to fix them ourselves. A little while later the first unit required a replacement picture tube. At that time we removed some very overheated parts from the video board,

and brought that board up to the revision level expressed by the schematics and the two new terminals. In November of 1978 we bought our last 4 Hazeltines. Two required replacement of the main logic board upon arrival, and a third required it a week later. (All in-warranty repair work was done on-site by TRW).

Since then, we have added a fan to the first unit to cool the once-more overheated video resistors (R13, R63 and R64), which we also replaced with double the design wattage. One of the other units also fried those resistors, but did not require a fan. After a year, another unit lost its Intel 8224 chip, which we replaced.

After 16 terminal-years of operation, they're chugging along pretty well, with some reservations. The keys are cheap switches, with a periodic cleaning requirement. We simply pull all of the keycaps and squirt in contact cleaner, wipe with a burnishing tool (or newspaper) and put back the keycaps. Somebody recently tried to remove a powercord by catching it in a door hinge. The strain relief gave its life to save the terminal's guts, and the repair was minimal.

My overall impressions are: The Hazeltine 1500 series uses a common master logic board, a common video board, and (1520's) an optional extra board for printer operations. The 1500 master board consists of an 8080 processor, 2K bytes of RAM, a character generator ROM (or two) and a program ROM (or two). There is also a large number of support chips and components. The keyboard is an integral part of the main logic board. All components (except for the character ROM) are soldered in place, and two cables lead to the power supply and video board. The RS232 connectors and option board/maintenance connectors are on the rear edge. We have found this master logic board to be quite well designed, and quite reliable – AFTER THE FIRST WEEK. At the time of our "four in, three bad" delivery, other people on the campus were reporting 50% dead-on-arrival.

The video/monitor board is another story. Many of the components appear to run at their design limits (1 watt resistors dumping .9 watts). This has led to the only design-fault repairs we have had. All of the units are visibly cooking up there (the boards are discoloring), but shouting at people who block the cooling vents keeps the repair rates down. Our first unit did not have ventilation slots that later ones grew, but it now has a fan. The first three did not have access to the vertical hold adjustment pot. We drilled the hole that later appeared in Hazeltine's newer cases.

Hazeltine will mail replacement parts, but I believe there's a $50 minimum. Their prices are such that meeting the minimum is no problem. To replace the main logic board is about $600. To replace the 8080 was (old price list) $75. We don't buy our parts from Hazeltine.

They used to sell a "kit" version of the 1500 for $799. The only assembly required was putting the power supply and main board into the case and bolting it together. It was almost worth it for the replacement part stock.

The dirty-key problem is a continual pain. My personal view is that the rubout/delete should NOT be a SHIFT function . . . it should be the lowercase code, or a separate key. The control key is located in a horrible spot (at the left end of the space bar). Everybody hits it instead of SHIFT (especially when doing deletes). The shift keys should be double sized. The CLEAR key should be a local function, not a send-to-computer, computer-echo-to-clear sequence.

Nice touches are: the terminal configuration switches are easy to get to and well labeled (if you know what they mean). The RS232 connector also supports current loop, and is extremely flexible in its handling of the control signals that RS232 can throw at you. The terminal can be run with the simplest 3 wire hookup (signal in, out and ground), but is fully capable of being controlled by clear-to-send, carrier-detect and the rest. The pass-through AUX port is very handy for classroom repeater applications and testing of other terminals. If you hold down RESET when turning on the terminal's power, the screen maps the control area of RAM. Our units give different screens, with different effects from later keystrokes, but it's a valuable quick-diagnosis tool. The built-in glare killer is very effective.

Since then we have switched to Zenith Z-91's (Heathkit H-19). These are available fully assembled for $740 in all the magazines. They have a very good keyboard layout (IBM Selectric). Their DEC VT52-emulation matches our two DEC computers very nicely. The first one we played with lost its flyback transformer after two hours. A call to Heath revealed that they had gone to an upgraded model, which was mailed to us. The three Z-19's we have received since then have worked flawlessly. From a self-repair point of view, the Heaths shine with their fully-socketed logic boards, although I grant that contact corrosion could be a source of future hassle. The video board is soldered, but well laid out and runs cool. The keys are the same cheap switch that the Hazeltine uses (sigh). The terminal configuration is set by internal DIP switches. The switches can be overridden by keyboard or computer command. Price and functionality (buzzword) will probably keep us here until something better comes along. The Hazeltine 1552 is too expensive (the dumb 1500 is $790) and the Heath will be easier to fix (far fewer parts, cooler video design).

When people ask (and they do) I tell them that there are cheaper terminals on the market, but if you're talking to a DEC (or Heath H11) computer, the Z19/H19 is the way to go.

My views toward Mr. Callas' problem are muted by the same lack of information that Hazeltine complains about. What is meant by "doesn't work properly"? The normal user's manual is not sufficient for some strange hookups. I've occasionally had to reference the schematics to determine which side of the connector was causing the zoo marching (or failing to march) across my screen. If he means that some random keys don't work, they're probably dirty. If the entire numeric keypad doesn't work, it's a real problem. Our experience with 3-of-4 failing leads me to mistrust a new Hazeltine, but many other factors (temperature, line noise, trying to get current-loop out of the AUX port (you can't)) can cause "faulty" operation. If TRW will only respond to "commercial locations," Mr. Callas can walk to the corner grocery store and call TRW from there. Maybe a local Radio Shack would be willing to have him camp out there, waiting for the terminal doctor. Since the usual on-site repair method is total replacement of the main logic board (in a 1500, that's all there is), each of his trips to TRW should have effectively resulted in his acquiring a "new" terminal. I do wonder what his symptoms were . . . could he demonstrate them to TRW? Did he try?

I have found Hazeltines to be quite reliable after the first few minutes of operation, with the occasional exception of the older video boards. Our campus has (probably) more than 100 of them,

and any I see in offices appear to be happily working. The only reasons I do not plan to buy more is price and performance relative to terminals which were not available at the time I started buying Hazeltines. The Hazeltine 1500 series is fast becoming obsolete for the self-maintainer, due to construction methods and parts count.

Richard Seymour
University of Washington
Nuclear Physics Lab GL-10
Seattle, WA 98195

## Interaction vs. Speed

Dear *DDJ*,

Anthony Skjellum's preamble to his Z-80 video driver package in *DDJ* #51 has prompted a few thoughts on time-dependent coding. He states: "Scrolling is the rate dependent step in text display, therefore extra attention has been put toward minimizing the execution time of [the scroll routine] and to decoding the linefeed character . . . . If scrolling could be removed, the display speed would be much faster . . . . Instead of scrolling, the cursor is made to home . . . . The only disadvantage is a slight loss in readability which I feel is made up for by the gain in throughput." Just the sort of remarks that send system programmers back for another squint at trustworthy old driver code!

Our console display, like Skjellum's, is 1K (16 lines by 64 columns) and memory mapped. Scrolling is handled in the customary way: move the last 15 lines up 1 line, blank the last line, write the cursor to its first column. Our driver was written four years ago in 8080 code, without benefit of such luxuries as LDIR and bit manipulation instructions. The scroll handler, which was speed-optimized with some care, is 45 bytes long, including updating of the byte variable which holds the cursor X coordinate, and runs in 48220 clock cycles. Since our CPU is Z-80 at 4MHz and the driver is downloaded to RAM on cold boot, a scroll operation takes 12.055 milliseconds. Then there is overhead to test an escape sequence flag byte (through a slow LDA addr, ORA A, JNZ addr sequence,) strip the cursor, detect the linefeed, branch to the scroll routine, restore the cursor, and return, requiring another 57 bytes (420 clock cycles, or 0.105 milliseconds). An infinite loop to pass linefeeds continuously to the video driver requires 8 bytes and 34 cycles or .0085 milliseconds. Thus such a loop will scroll the display up one line every 12.1685 milliseconds or 82.179398 times per second. This is about 2.74 times faster than standard video frame rate (30 Hz) WHICH IS THE ACTUAL TIME DEPENDENCY IN THIS CASE. All sixteen lines will scroll off the screen in about 0.195 second. Is this a rate of data throughput that dissatisfies the human eye? If it is, then hardware scrolling (the old VDM-1 had it), like hardware random number generation, is to be preferred.

I haven't nickel-and-dimed my way through Skjellum's drivers (which are obviously faster) in the same way, and the foregoing in no way implies a criticism of them: they work fine within his stated limits, a good tutorial for people new to a nontrivial problem . . . and I'm grateful for the reminder that our own system drivers need recoding for the Z-80. However, such recoding will be done very largely in the interest of wringing the last bytes out of precious space in the memory map, with careful attention to user legibility, where it is often desirable to slow down a console display, rather than speed it up. Some code is time dependent only within rather wide limits, and not to worry if it doesn't tear along like a Maserati. This applies especially to user interface I/O. Human beings are notoriously "slow peripherals" and benefit more from interaction than speed, assuming the system isn't hopelessly pokey.

We're no strangers to time dependency around here. Raster graphics and sound synthesis are the main concerns of this lab. Floppy disk I/O is a molasses nightmare, and waiting for the so-called Fast Fourier Transform to do its job, or watching a clever cellular automaton crawl through a 64K pixel array, can bring the bravest assembly language hacker to the verge of tears! So this is not one of those "BASIC is always good enough" protests. It's simply to offer the suggestion that we take a hard look at the actual constraints of software design before tormenting ourselves about squeezing the last nanosecond out of a loop. To do so may not always be better, but it is ALWAYS harder, and there may be more gratifying places to use our energy.

Thanks for a publication of continuing excellence. Lab copies quickly become battered like old teddy bears. That's always high praise.

Best regards,
Hollis Frampton
Digital Arts Lab
Center for Media Study
SUNY at Buffalo
Buffalo, NY 14214

## A Doctor's Advocate . . .

Dear Doctor,

It is over a year since I have taken word processor in hand and dashed off a letter to my beloved *Dr. Dobbs*. The immediate cause is a letter from a Mr. Ickler of Corpus Cristi, whose obvious sexism shows his social conscience to be as antiquated as his knowledge of computers. The passage that referred to the editor as a "girl journalism student" shows his sexual bias (come to think of it, what ever happened to that adjective "female"?).

When he says, "I am looking for details on writing systems utilities such as spoolers, designing compilers, text editors and so on. Maybe a tutorial on CP/M concepts is useful, but PASCAL[sic]?", he shows his knowledge of system design to be equally out of date. At many companies, including the small computer company for which I work (Apple), such system software is often written in Pascal.

Anybody who claims to be interested in writing system software but decries Pascal is certainly not in any position to criticize the editor of *DDJ* on technical grounds. Mind you, I am not taking up the cudgel for the Pascal language or the UCSD operating system, nor am I attacking the CP/M operating system – a pox on both their houses – I am merely noting how neatly male chauvinism goes along with hasty, uninformed judgment.

That problem laid to rest, let me voice my support of *DDJ*: may you continue on your merry orthodontic way, living from hand to mouth, telling the truth and representing computer consumers as you have been.

Jef Raskin
10696 Flora Vista
Cupertino, CA 95014

## CP/M DUMP

Dear *DDJ*:

Enclosed is a software modification for the CP/M DUMP Utility Program that will greatly enhance the readability. This mod will print 2 SPACES after the line number, 2 SPACES after the 4th and 12th bytes, and 3 SPACES after the 8th byte.

```
Following   LXI   SP,STKTOP   in the SET UP STACK routine add
DD210402    LXI   IX,0204H    ; Set up byte counters

following   CALL  PHEX   in the GLOOP routine add
3E20        MVI   A,' '        ; Double space after Line Number
CD6101      CALL  PCHAR

following   CALL  PHEX   in the NONUM routine add
DD2D        DCR   LX           ; Decrement 4th byte counter
C2D301      JNZ   GLOOP
3E20        MVI   A,' '        ; Double space after 4th and 12th byte
CD6101      CALL  PCHAR
DD2E04      MVI   LX,4
DD25        DCR   HX           ; Decrement 8th byte counter
C2D301      JNZ   GLOOP
3E20        MVI   A,' '        ; Triple space after 8th byte
CD6101      CALL  PCHAR
DD2601      MVI   HX,2
```

Sincerely,
Tom Cage
P. O. Box 806
Roswell, GA 30075

```
A>DUMP TEST.123

0000 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0010 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0020 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0030 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0040 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0050 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0060 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0070 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0080 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0090 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00A0 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00B0 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00C0 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00D0 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00E0 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00F0 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

A>DUMP1 TEST.123

0000  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
0010  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
0020  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
0030  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
0040  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
0050  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
0060  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
0070  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
0080  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
0090  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
00A0  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
00B0  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
00C0  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
00D0  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
00E0  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
00F0  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
```

## Bitter Buyer

Dear Editor:

Please accept the following book review for publication in your magazine.

It's a sad day when a publisher can successfully market high-priced rubbish; more interesting than what it tells us about the author and publisher are its implications about the marketplace. "Buyer beware" could hardly be overemphasized in today's fast-moving microcomputer marketplace. I recently bought a copy of *The CP/M Handbook with MP/M* by Dr. Rodney Zaks (a SYBEX publication) hoping to obtain a good reference work on CP/M. What a shock!

Chapters 1, 3, and 4 amount to a primer on operating a CP/M computer, using PIP, and using ED. Beyond the limited value of this hand-holding the book is a veritable wasteland. By the skillful use of ill-defined (and undefined) terminology, Dr. Zaks confuses the novice computer user. On the other hand, a total lack of in-depth information makes the book useless for the seasoned user. Dr. Zaks succeeded in missing both audiences.

The text is a cluttered mass of bad grammar. Unrelated thoughts are concatenated into rambling sentences, and these are further grouped into arbitrary paragraphs. Careless errors abound. In several places the text is actually incompatible with referenced figures. But what I find most intolerable is the inadequate presentation of facts. The reader is presented with statements of the following types:

1. Vague and misleading:
   "The theoretical difference between a compiler and an interpreter is that a compiler executes the program more efficiently while an interpreter allows interactive program development." (pg. 26)

2. Inconsistent:
   "Whenever you use a filename as an argument to a command, you must type the *entire* filename, including the extension, if any. The only exception is when you use a file as a transient command . . .." (pg. 58)

*(Continued on page 13)*

# Computer-Based High Speed Reading

## Introduction

It's fairly well known that the highest data rate path from the outside world into the human brain is through the eyes. For most people, reading is the primary technique for getting information produced by other humans into our brains. The popularity of speed reading courses indicates a widely felt need to speed up this form of information transfer.

Some interesting work in this area has been done by Mary Potter*, a researcher at MIT. Her idea was simple: present text material sequentially, one word at a time. In the experiment set up, the subject looked at a rectangular window in which the words to be read appeared one after the other. The interesting aspect of this arrangement is that reading takes place without any muscle movement; there is no mechanical limit to the speed at which the reading can take place. Ms. Potter used modified movie projectors to present the text; it occurred to me and to others that a computer-based implementation would be easier to develop and modify.

A friend of mine at Xerox PARC (John Warnock) first told me about Ms. Potter's work and also wrote a program for a PDP-11. He found that reading speeds of 1000 words per minute were feasible. Those experiments were preliminary and not closely controlled. In Ms. Potter's work, the upper limit had appeared to be 720 wpm. The BASIC program given below permits a range of speeds from 225 to 450 wpm; faster implementations are possible but would require machine language for the word-at-a-time display routine.

Just providing a high bandwidth path from eyes to brain is no guarantee that use of such a path will allow high-speed reading with good understanding and retention of the material read. The subjective judgment of the word-at-a-time reader is especially questionable if the same person has previously input the material to be read.

Given these warnings, I still find the technique intriguing. I hope others agree, and will experiment with it further. Any technique which has the possibility of increasing the rate at which text can be read is worth looking into.


**by Dave Caulkins**

*Dave Caulkins, 1263 El Camino Real, Box E, Menlo Park, CA 94025.*


## Computer-Based Word-At-A-Time Reading

The BASIC program whose listing is given below runs in a Commodore PET equipped with a Compu/Think disk system. The program allows the user to enter natural language text, which is then stored on a disk file. This same text can be read from the file and presented to the user for reading in word-at-a-time form. The user has continuous control of the rate at which the words appear. By entering a number from 1 (fast) to 9 (slow) he can vary the rate at which the words appear from 450 to 225 words per minute.

The rate is dynamically variable during the word-at-a-time display.

## The BASIC Program

Some peculiarities of PET BASIC should be explained. The PET uses special characters for cursor control; unfortunately, they do not print on my printer. I've added comments to indicate these characters; an example appears in line 27. Here V$ is set equal to a string of 10 move-cursor-down characters. Because they do not print, all that appears in the listing is V$="". The PET has an OS-defined time variable TI; it is incremented every 1/60 second under interrupt control.

Lines 24-40 set up variables for rapid access. In PET BASIC, as in many other Microsoft BASICs, variables execute with a speed proportional to the order in which they are defined. Variables always execute more rapidly than constants. Similarly, program states appearing early in the program execute more quickly than statements which follow them.

In line 80, the GOTO skips over the fast routines to get to the set up and user command handling starting on line 360. Line 370 fills the text word array W$ with a non-printing character and line 374 opens the disk file TXT1 for reading; line 376 turns off the disk drive motor turned on in 374. In 412 the user is prompted to select disk file reading or writing. If writing (W) is selected, the program branches to 480 where instructions for entering text are given. The disk file TXT1 is opened for writing in 510, the disk drive is turned off in 520 and the text string entry subroutine at 600 is called. When a carriage return signalling the end of a line of text is detected in 610, the subroutine returns control to 520; the disk motor is turned on in 525 and the text line is written to the disk in 530. If the text entry terminate string #E# is detected in 535, the disk file is closed in 550 and the program restarted in 560. The user will be returned to the read or write prompt of 412, and can read the file just written, or can write a new one.

If reading is selected in 420, the user is given a menu of reading rates in 423-433 so that an initial word rate may be selected in 440. In 448 a subroutine at 3000 is called; this displays two * characters, one above and one below the place on the screen where the center of each word will appear. The program then branches to 146, the start of the fast word-at-a-time display routines. Here the cursor is positioned to the center of the screen in 146, and a FOR-NEXT loop started to read successive words from the text word array W$. The timing variable IT is initialized in 148 and the next word read from the array in 150. If the next array cell contains the non-printing character loaded in 370 (this will be the case on first entry to 146) the program branches to 300 where the next string is read from the disk. In 304-344 the text string is parsed and individual words are identified by their delimiting spaces; each word is placed in one cell of W$. Successive strings are read from the disk and their constituent words placed in W$ until the terminating mark #E# is encountered; when this occurs the disk drive motor is turned off in 346; the program branches back to 146.

In 216 the word is displayed, centered in the visual field defined by the two * characters written by the subroutine at 3000. In 217 the cursor is moved up one space (again, this character does not print in the listing and appears as " "), and the subroutine at 224 is called. This reads a character from the keyboard using GET so there will be no pause if no key is pressed. If a numeric key is pressed, its value is used to form the delay variable D which in turn causes the program to dwell in 228 for the amount of time required to provide the inter-word delay corresponding to the word display rate selected by the user. ■■


**(LISTING ON PAGE 34)**


* "Comprehension And Memory In Rapid Sequential Reading," Potter, M. C.; Kroll, J. F.; and Harris, Carol. From *Attention And Performance VIII*, edited by R. Nickerson. Published in 1980 by Erlbaum (N.J.).

# Analysis of the Use of the 6502's Opcodes

by Howard W. Whitlock, Jr.

Despite its remarkably sparce (151) set of opcodes, the 6502 CPU is considered to be more powerful than its cousin, the 6800. This is generally ascribed to its "post-indexed" addressing mode, represented mnemonically by "LDA (zpg),Y". This instruction computes the effective address of its operand by an unsigned addition of the Y register (8 bits) to a 2-byte zero page pointer (at zpg & zpg+1).

I have written a simple program that analyzes the Commodore PET's usage of the 6502 opcodes. I was particularly interested in the frequency of usage of the above type of indirection in this reasonably big piece of software. I had noticed in my own work that the other type of indirection on the 6502, "pre-indexed" indirection, was almost never of use and wondered whether this was just peculiar to my programming habits. The program written merely steps through the BASIC system, from its start at $C000 to its end at $FFFF, avoiding data tables, messages and other non-instructional areas, and counts the number of times each opcode is used. Here is what I find:

The PET BASIC interpreter occupies a little less than 14K of instruction memory. It consists of 6355 instructions, give or take a few. The "average" instruction is thus about two bytes long. The 6502 CPU possesses 151 discrete one-byte opcodes; the remaining 105 are not implemented. Their frequency of use in the PET ranges from very heavy (the JSR instruction itself accounts for more than 10% of the total) to not at all. Table 1 summarizes the situation for those twenty percent of the instructions that are most frequently used. We see that

1) The 80/20 rule is followed closely. Thirty opcodes (20% of the total) account for 5131 (81%) of the total 6355 used.
2) There are only three instructions in Table 1 that employ 16-bit operands. Two of these, JSR and JMP, have no substitute on the 6502. The third is store accumulator, STA adr16, but it occurs only 72 times. Its complement, LDA adr16, is interestingly used only 38 times. Compare this with the zero page STA and LDA instructions that are used 511 and 437 times respectively. If we assume there are about 13 subroutines (136 RTS's), the "average" subroutine is called five times (684 JSR's).
3) The most popular instructions (#3-20) are described fairly well by:

   Frequency=28+1325/X

   where X is the ranking, with a correlation coefficient of 0.978. This is similar to Zipf's law, which states that the *n*th most frequent word is encountered 1/*n* as frequently as the most common.

*Howard W. Whitlock, Jr., 1614 Norman Way, Madcity, WI 53705.*

**Table 1**

**The thirty most-used instructions in PET BASIC and their frequency of use.**

| FREQUENCY | INSTRUCTION | FREQUENCY | INSTRUCTION |
|---|---|---|---|
| 648 | JSR adr16 | 116 | STX zpg |
| 511 | STA zpg | 110 | LDX #data |
| 437 | LDA zpg | 109 | LDA (zpg),Y |
| 388 | BNE loc | 96 | LDY zpg |
| 320 | LDA #data | 92 | LDX zpg |
| 267 | BEQ loc | 75 | BPL loc |
| 221 | JMP adr16 | 73 | STA adr16 |
| 197 | LDY #data | 72 | STA (zpg),Y |
| 153 | CMP #data | 72 | INC zpg |
| 144 | ADC #data | 67 | BCS loc |
| 136 | TRS | 60 | TAX |
| 135 | STY zpg | 56 | CLC |
| 127 | BCC loc | 55 | ADC zpg |
| 127 | PHA | 55 | ADC #data |
| 121 | INY | 54 | BMI loc |

**Table 2**

**The forty-one least-used (i.e., never) instructions, in numerical order.**

| | | |
|---|---|---|
| BRK | AND adr16,X | ADC adr16,X |
| ORA (zpg,X) | AND adr16,X | ROR adr16,X |
| ASL adr16 | ROL adr16,X | STX zpg,Y |
| ORA (zpg),Y | EOR (zpg,X) | LDA (zpg,X) |
| ORA zpg,X | EOR adr16 | LDX zpg,Y |
| ORA adr16,Y | LSR adr16 | CLV |
| ORA adr16,X | EOR (zpg),Y | LDY adr16,X |
| ASL adr16,X | EOR zpg,X | CPY adr16 |
| AND (zpg,X) | EOR adr16,Y | CMP zpg,X |
| AND adr16 | EOR adr16,X | DEC adr16,X |
| ROL adr16 | LSR adr16,X | SBC (zpg,X) |
| AND (zpg),Y | ADC (zpg,X) | SED |
| AND zpg,X | ROR adr16 | INC adr16,X |
| ROL zpg,X | ADC zpg,X | |

## Table 3

**All used opcodes and frequency; in decreasing order. Those opcodes employing post-indexed addressing are flagged with a *; those using pre-indexing with a †.**

| | | | |
|---|---|---|---|
| OPCODE: 20 | COUNT= 684 | OPCODE: 4A | COUNT= 19 |
| OPCODE: 85 | COUNT= 511 | OPCODE: 45 | COUNT= 18 |
| OPCODE: A5 | COUNT= 437 | OPCODE: E4 | COUNT= 17 |
| OPCODE: D0 | COUNT= 388 | OPCODE: D1 * | COUNT= 17 |
| OPCODE: A9 | COUNT= 320 | OPCODE: C0 | COUNT= 17 |
| OPCODE: F0 | COUNT= 267 | OPCODE: 66 | COUNT= 16 |
| OPCODE: 4C | COUNT= 221 | OPCODE: 99 | COUNT= 15 |
| OPCODE: A0 | COUNT= 197 | OPCODE: EA | COUNT= 14 |
| OPCODE: C9 | COUNT= 154 | OPCODE: 46 | COUNT= 14 |
| OPCODE: 68 | COUNT= 144 | OPCODE: 58 | COUNT= 13 |
| OPCODE: 60 | COUNT= 136 | OPCODE: 78 | COUNT= 12 |
| OPCODE: 84 | COUNT= 135 | OPCODE: 2A | COUNT= 12 |
| OPCODE: 90 | COUNT= 127 | OPCODE: 26 | COUNT= 11 |
| OPCODE: 48 | COUNT= 127 | OPCODE: 05 | COUNT= 11 |
| OPCODE: C8 | COUNT= 121 | OPCODE: 28 | COUNT= 10 |
| OPCODE: 86 | COUNT= 116 | OPCODE: 9A | COUNT= 9 |
| OPCODE: A2 | COUNT= 110 | OPCODE: 08 | COUNT= 9 |
| OPCODE: B1 * | COUNT= 109 | OPCODE: B4 | COUNT= 8 |
| OPCODE: A4 | COUNT= 96 | OPCODE: 50 | COUNT= 8 |
| OPCODE: A6 | COUNT= 92 | OPCODE: 25 | COUNT= 8 |
| OPCODE: 10 | COUNT= 75 | OPCODE: 06 | COUNT= 8 |
| OPCODE: 8D | COUNT= 73 | OPCODE: DD | COUNT= 7 |
| OPCODE: E6 | COUNT= 72 | OPCODE: BA | COUNT= 7 |
| OPCODE: 91 * | COUNT= 72 | OPCODE: 94 | COUNT= 7 |
| OPCODE: B0 | COUNT= 67 | OPCODE: 8E | COUNT= 7 |
| OPCODE: AA | COUNT= 60 | OPCODE: 70 | COUNT= 6 |
| OPCODE: 18 | COUNT= 56 | OPCODE: 6C | COUNT= 6 |
| OPCODE: 69 | COUNT= 55 | OPCODE: 76 | COUNT= 5 |
| OPCODE: 65 | COUNT= 55 | OPCODE: F5 | COUNT= 4 |
| OPCODE: 30 | COUNT= 54 | OPCODE: AE | COUNT= 4 |
| OPCODE: E8 | COUNT= 49 | OPCODE: 79 | COUNT= 4 |
| OPCODE: C6 | COUNT= 48 | OPCODE: F1 * | COUNT= 3 |
| OPCODE: 8A | COUNT= 48 | OPCODE: CD | COUNT= 3 |
| OPCODE: 88 | COUNT= 48 | OPCODE: 8C | COUNT= 3 |
| OPCODE: 98 | COUNT= 46 | OPCODE: F6 | COUNT= 2 |
| OPCODE: A8 | COUNT= 45 | OPCODE: EE | COUNT= 2 |
| OPCODE: CA | COUNT= 43 | OPCODE: D9 | COUNT= 2 |
| OPCODE: C5 | COUNT= 43 | OPCODE: D8 | COUNT= 2 |
| OPCODE: 38 | COUNT= 40 | OPCODE: AC | COUNT= 2 |
| OPCODE: 29 | COUNT= 40 | OPCODE: 71 * | COUNT= 2 |
| OPCODE: AD | COUNT= 38 | OPCODE: 56 | COUNT= 2 |
| OPCODE: 49 | COUNT= 37 | OPCODE: 40 | COUNT= 2 |
| OPCODE: BD | COUNT= 35 | OPCODE: 16 | COUNT= 2 |
| OPCODE: 09 | COUNT= 31 | OPCODE: FD | COUNT= 1 |
| OPCODE: E5 | COUNT= 30 | OPCODE: F9 | COUNT= 1 |

*(Continued on next page)*

An amazing 27% of the 6502's opcodes are not used at all! These are contained in Table 2. We see that

1) The not-used instructions are heavily represented by logical (especially XOR) instructions.
2) While there are no pre-indexed, indirect memory references in the top 20%, they are fairly common (six of eight ) in the never-used table. The only ones used at all (see Table 2) are STA (zpg,X) and CMP (zpg,X). These are used in conjunction with a zero-page string descriptor stack.
3) Neither BRK nor SED (set decimal mode for arithmetic) are used.
4) The only post-indexed instructions never used are the logical ones, ORA, AND, and EOR. I have included all used opcodes as Table 3.

I have also counted all adjacent opcode pairs drawn from the instructions of Table 1. The ten most frequently used doublets are in Table 4. Almost all of the doublets are encountered more frequently than expected in a statistical sense (Fa*Fb/6355). There are some notable exceptions, however. The two compare-and-branch pairs, CMP #data: BNE loc, and CMP #data: BEQ loc, are much more common than expected. On the other hand, the doublet BNE loc: JSR loc is (relatively) uncommon. Rather than delving into the Freudian implications of these and related observations, I invite interested readers to send me a SASE and $1.00 for the complete printout.

## Conclusions

My original suspicions concerning the worthlessness of the 6502's pre-indexed instructions seem confirmed. On the other hand, post-indexed indirection, although much more common than pre-indexing, is not nearly as common as I had assumed. The small number of instructions actually used is surprising. The absence of any "super" two-op constructs is apparent from Table 4.

■■

| | | | |
|---|---|---|---|
| OPCODE: 0A | COUNT= 30 | OPCODE: ED | COUNT= 1 |
| OPCODE: 2C | COUNT= 29 | OPCODE: EC | COUNT= 1 |
| OPCODE: E9 | COUNT= 28 | OPCODE: D6 | COUNT= 1 |
| OPCODE: 24 | COUNT= 28 | OPCODE: CE | COUNT= 1 |
| OPCODE: E0 | COUNT= 25 | OPCODE: C1 | COUNT= 1 |
| OPCODE: 95 | COUNT= 22 | OPCODE: BE | COUNT= 1 |
| OPCODE: B9 | COUNT= 21 | OPCODE: 81 | COUNT= 1 |
| OPCODE: C4 | COUNT= 20 | OPCODE: 6D | COUNT= 1 |
| OPCODE: B5 | COUNT= 19 | OPCODE: 6A | COUNT= 1 |
| OPCODE: 9D | COUNT= 19 | OPCODE: 0D | COUNT= 1 |

**Table 4**

**The ten most common opcode pairs, and their calculated* and found frequency of occurence.**

| OPCODE-1 | OPCODE-2 | FREQUENCY | (CALC) |
|---|---|---|---|
| JSR loc | JSR loc | 145 | (74) |
| LDA #data | STA zpg | 106 | (26) |
| JSR loc | LDA zpg | 82 | (47) |
| STA zpg | LDA zpg | 76 | (36) |
| CMP #data | BNE loc | 73 | (9) |
| JSR loc | LDA #data | 71 | (34) |
| BNE loc | JSR loc | 57 | (42) |
| LDY #data | JSR loc | 55 | (21) |
| JMP loc | JSR loc | 55 | (24) |
| CMP #data | BEQ loc | 52 | (6) |

Fa*Fb/6355, where Fa is the frequency of occurence of the first, and Fb of the second opcode. See Table 1.




**"There are times when I worry about you, George!"**

*(Continued from page 8)*

That is an interesting statement in its own right, but compare it to this one:

"To execute ASM, . . . you only have to specify the primary name of the source file." (pg. 82)

3. Erroneous:
   "Files may have one of the four attributes:
   – R/O (read only)
   – R/W (read/write)
   – SYS (system . . .)
   – DIR (directory . . .)" (pg. 94)
4. Incomplete:
   "XX (positions 13 & 14): This is a field not used in CP/M 1.4, normally 0." (pg. 118)
5. Absurd:
   "A BIOS is specified by passing a function number and an information address." (pg. 192)
6. Amusing:
   "if you are in a duplicating mood [sic], PIP is the next most useful program." (pg.279)
   What is the most useful program for those in a "duplicating mood?" – not the faintest clue is given.

Reading these statements in context doesn't help matters at all. What the book lacks in content, it makes up in redundancy. Chapter 5, "Inside CP/M (and MP/M)," classically illustrates this curious literary style. In seventeen pages (181-197) Dr. Zaks manages to present the CP/M memory map three times. And twice he labels the fields of a File Control Block; this, with a spattering of floppy diskette statistics, he calls a "detailed description" of the CP/M file system.

And what should the novice user do when he encounters problems? Listen to this sage advice:

"When something goes wrong, you may want to stop everything. Do not pull the electrical cord out of the outlet." (pg. 279)

"Check the mechanicals again . . . remove boards, clean connections . . . . If the malfunction can be attibuted to a board, remove components from sockets, clean connections, and reinsert them." (pg. 280)

If you find this humorous, then obviously you haven't spent $13.95 for this book. Good for you!

Jim Hendrix
Rt. 1, Box 74-B-1
Oxford, MS 38655

■■

# Book Reviews

**PROGRAMMING TECHNIQUES FOR LEVEL II BASIC**
**By William Barden, Jr.**
Published by **Radio Shack**
**224 pages, $4.95**
Reviewed by **John B. Palmer**

This is not another book about how to learn to program in BASIC. If you are fluent in BASIC and wish to learn details of Radio Shack's Level II BASIC, then this book is for you. Anyone having trouble with BASIC had best avoid this book.

For most TRS-80 users, the best part of the book might be chapters seven and eight, where the author gets into dimensions, tables, lists and sorts. He gives some program examples and good illustrations of the well-known Bubble-sort (ugh!) and the better Shell-Metzer sort (hooray!). Also mentioned are linked lists, merges and deletions. This is good stuff for anyone trying to maintain a mailing list in BASIC.

It's too bad the program examples are always so brief. The author tries to put a lot of information into a book of 224 pages. I could not find any complete, practical applications of the techniques which are detailed. Yet I feel the book is very good in overall coverage of its subject. If you are in need of an obscure detail or a special trick in Level II, check the four-page index of this book.

Of the twelve chapters in the book, my own favorites are the last two. They are where you learn the very clever trick of BASIC-to-machine-language interface. William Barden, Jr. tries to sugar coat the horrible details of Z-80 code with his strange sense of humor.

But his humor is not the worst part of the book. The greatest weakness is the lack of a theme to tie together a collection of facts, tricks and algorithms. It would have been better if there were at least one application example to tie everything together. The title of the book mentions technique. It has technique; it does not have style.

You may find that much information covered in this book has already been published in letters, memos and newsletters. It is very handy to now have the secrets of Level II BASIC in one volume. Suppose you want to change the keyboard input routine of your TRS-80. Look in the index and then turn to page 215. Wow! on the facing page there is a very interesting, top-secret diagram of the Level II DCB's! Here is everything you need to know about the I/O structure.

Many users will find the chapter on cassette tape technique to be helpful. The author exposes the weakness of the Radio Shack tape format (unusual in view of the fact that the book is published by Radio Shack). There are some hints for overcoming these limitations. I think the author should have given a lot more attention to practical examples of using the tape for storage of data, but he does detail how the BASIC interpreter puts data onto the tape.

I paid four dollars and ninety-five cents for this book at a local Radio Shack store. It is better than some books at twice the price. Too bad it does not have spiral binding and will not lay flat on the desk. It is not a replacement for the regular manual you get with Level II. But when the Level II manual puts you to sleep, pick up a copy of *Programming Techniques for Level II BASIC* by William Barden, Jr. and you will again have sleepless nights with your TRS-80.

Post Script: some information in this book may be in error. Also, future releases of Level II BASIC may not have the same machine code addresses mentioned in the book.

**BASIC TECHNIQUES OF COMBINATORIAL THEORY**
**By Daniel I. A. Cohen**
Published by **John Wiley and Sons**
**$20.95**
Reviewed by **Jeff Rubens**

The "Curriculum 1978" report of the Association for Computing Machinery states, in effect, that the traditional mathematics curriculum is still appropriate for computer science. Nonetheless, there is a strengthening opinion among educators that students majoring in computer-oriented disciplines need more discrete mathematics and not necessarily a great deal of calculus and analysis. Since few people are exposed to much non-traditional mathematics in school, a sensible move for those who are contemplating study in computer science is to get a college-level text devoted to this "other" mathematics. This could provide a valuable clue to whether one is heading in the right direction. *Basic Techniques of Combinatorial Theory* is the sort of book you might consider for this purpose.

The content and level are designed for a reader with limited mathematical background. A semester of calculus is recommended, but if you know nothing of calculus there are only a few pages you will have to skip. It would be foolish to think, though, that you could easily understand this book if you had to struggle to get through high-school geometry. Although essentially self-contained, this is still a technical book. The more familiarity you have with ideas of deductive reasoning and proofs, the easier the book will be. The most difficult sections, on Polya enumeration, will be easier for those who have been exposed to algebraic structures.

Cohen's approach to instruction is markedly different from the one most commonly found in mathematics texts. He proceeds from the bottom up, showing the lesser or partial results along the road to the major theorems. This technique not only gives the reader a reasonable idea of how the subject developed historically, but also makes many complex results easier to understand than the traditional technique (which first hits you with the entire truck and tells you about wheels, engines, and so forth only while you are picking yourself up off the ground). The author succeeds admirably in bringing off this pedagogic idea. For example, his development of Ramsey's Theorem along these lines is splendid, especially when compared with the treatment of this subject elsewhere. It seems likely that this feature is responsible for the book's classroom success.

Another valuable instructional aid is the provision of several proofs of many of the results. Those with good understanding see various ideas neatly tied together; those groping at comprehension get several chances, from different points of view. Unfortunately, things being the way they are, some alternative proofs must be delayed. The references back, usually by number alone, seem sterile and impersonal. (Also, page numbers would have been helpful.) In contrast, the author goes to great lengths to credit results appropriately, frequently citing two or more researchers who attained the same result independently.

In order to achieve his objective of a meaty book within the scope of readers with little formal training, Cohen concentrates his energies on carefully selected aspects of combinatorics. The bulk of the book is, one way or another, about counting, with side trips into related realms such as generating functions, re-

*(Continued on next page)*

# ERGO: A "Thinking Program"

This article is about a "thinking program" that we wrote some time ago. It is one of a series we developed to investigate some ideas on educational technology. The purpose of publishing it now in this Journal is to show it to readers with the interest and skill to do even better. We felt it would be constructive to demonstrate that it is not really very difficult to design a program that does genuine thinking, and it does not have to be very large, so it can be a challenging project for computer hobbyists.

ERGO thinks about any kind of information supplied to it, and does deductive and inductive logical analysis of its available data to draw new conclusions not explicitly contained in its original source information, similar to the way a person would do it. As a result, it generally "knows" considerably more than the specific facts given to it.


**by Sheldon Softky and Sylvan Rubin**

*Sheldon Softky and Sylvan Rubin, c/o P.O. Box 3074, Santa Clara, CA 95051.*


It is entirely different from smart game-playing programs, because it can think about anything. It is not very clever, however, and doesn't engage in entertaining conversations. This version does sometimes ask random questions based on its knowledge, so it is somewhat conversational.

To be more specific about the internal features of ERGO, it normally gets an initial "information load" from a data file that it reads when it is started. Then it deduces everything it can from that data and adds it to its memory. Any statements made to it are also added and combined with the facts already known, to generate more deductions. If it is asked a question, it gives all relevant answers, and then also does inductive analysis to find any more "possible" answers, but these are not stored in memory. One problem we had with it was that if it was given a highly interrelated set of facts, it deduced so many more facts that it sometimes ran out of computer memory.

When you look at the listing, you'll see that by far the biggest part of ERGO is devoted to trying to comprehend the English language. The essential notion is that the verb is the most important word in a sentence, because it states the relationship between the subject and the object, which determines what kind of connections there are with other objects. The input file, and the internal data structures are organized in terms of sets of relations, one set for each verb. ERGO only knows about the verbs "is" and "has." These are two of the most logically powerful verbs, because they imply extensive relational connections among many objects. Most verbs only mean "do," regardless of their great variety, and are therefore logically very simple.

The listing is published, not so much for anyone to exactly copy it, but to provide a starting point for readers to produce new and different thinking programs, hopefully much more powerful than ERGO. ■■


**(LISTING ON PAGE 30)**



*(continued from previous page)*


currence relations and permutations. One chapter deals with graph theory. The topic selection satisfies the author's requirements and, perhaps coincidentally, is also excellent for readers interested in computer science.

Each chapter provides an impressive collection of exercises. For many, this alone will make the book worthwhile. The problems are not formally graded as to difficulty (but various levels are represented), and almost all are of a strictly mathematical nature. The reader is left to discover the more practical applications for himself – and if he has been attentive, he probably will.

A certain amount of reader awareness is important in the use of this book. It should be kept in mind that this is an introduction to combinatorics, not a survey. Don't expect to get a head start on such things as systems of distinct representatives, optimizations or block designs. Similarly, be forewarned that only a thin surface of graph theory is scratched, even as regards a list of its subtopics. If you are practical-minded, rest assured that there are real-world applications (including many important ones) for this material and its corollaries. Finally, be on guard to watch for typographical errors. There are quite a few, several of which could be disconcerting to a beginning combinatoricist. If you are lacking in mathematical self-confidence and don't have a friendly professor nearby to answer questions, you might want to wait for the second, presumably corrected, printing. There should be another printing, for this is a useful book in an important area and its pluses far outweigh its minuses.

**Godel, Escher, Bach**
**By Douglas R. Hofstadter**
Published by **Random House**
**777 pages, $8.95**
Reviewed by **H. T. Gordon**

Its title – a string of 3 names, all Germanic like the author's – presages a singular book, as do the quasi-mystical subtitles: "an eternal golden braid" and "a metaphorical fugue on minds and machines in the spirit of Lewis Carroll." The minds that interest Hofstadter do not include many Germanic ones that I find no less fascinating: Brahms and Wagner, Goethe, Dürer and Rembrandt, Freud and Jung, Nietzsche, Spengler, and the nadiric Hitler. *Their* work is pervaded by the unconscious component of mind, either not or else too-complexly mathematical. Hofstadter is young (ca. 34), has a Ph.D. in mathematical physics (ca. 1976), and is now in computer science with a pro-AI (artificial intelligence) bias. His long list of reference works (p. 746-) includes most of the AI ones, many in mathematical logic (his *forte*), some in biology (not his *forte*), a few on music and art (to justify the minor Bach and major Escher "strands" of his golden braid, in which the Gödel-like strand is dominant), and a bit on Zen. Zen because he is enthralled by paradox – that I see as an antirational upwelling from the un-

conscious – as were Lewis Carroll, Zeno of Elea, and surely many other mathematical minds that have been incarnate in human cerebra from the moment of origin – lost in the mists of time – of *Homo sapiens*. A long, though not quite eternal, "braid"!

The author's father is a Nobel-laureate in high-energy physics, who has since 1950 been a professor at Stanford. In variety-show parlance, that's a hard act to follow. The psychosociology of everyone who gravitates to computers – including me – is unusual. Growing up in the Palo Alto microcosm – as, in a sense, *DDJ* itself has – is an *exotic* experience. The so-called private universities (they are lavishly subsidized by Washington) are foci of intellectual and many other (not all admirable) forces, to which Stanford adds the peculiar California lifestyle. Less weird than southern California perhaps, but weird enough. I have intentionally (though lightly) stressed the wellsprings of this book, though only the more "in" readers of this review may fully grasp its relevance.

The author concisely describes his own book (hidden in the reference to Gebstadter, p. 748) as "a formidable hodge-podge, turgid and confused." That's more accurate than the rave reviews it's had, but I would add that some of it is ingenious, imaginative, even brilliant; e.g., the brief sketch of computers in Chapter X (p. 285). The book may have originated as an introduction to number theory, symbolic logic, and the propositional calculus aimed at students in computer science. Early AI work showed that this kind of "thinking" can to some extent be programmed (cf. pp. 26-7). The first 250-odd pages especially have a lot of this, with long passages that could be lecture notes (learnable if one so desires). Not everyone is aware that the number 5 can be written as SSSSSO! The underlying theme is really AI, that becomes predominant in the last half of the book. Seeing that this kind of hard reading would not "sell," Hofstadter spiced it up with interludes on linguistic and genetic coding, music, Zen, graphics and an awful lot of Escher (no less than 36 reproductions!) Every chapter has a Zenoic-Carrollian "dialogue" involving several imaginary characters, principally Achilles and the Tortoise, first used by Zeno to "prove" what Ecclesiastes merely asserted, that the race is not to the swift. Zeno's trick was to refuse to give Achilles enough time to overtake the Tortoise, but Hofstadter's obfuscations often make no sense at all, being a mystifying mysticism glazed over by terminology that makes sense in other contexts. This can be entertaining in many ways. Some readers will extract from the nonsense the particular sense that pleases them, and admire the writer's insight (since it's their own!). Others will only see that it's "deep," and admire the Zen mind that must surpass their own. Others will be reminded how rare sense is, and how easily it is transformed into nonsense, wondering if they themselves avoid the subtle pitfalls that entrap even minds as bright as Hofstadter's.

People who are aware of the moronic level of present-day AI work will be struck by the fact that so much of this book is at an empyrean level of complexity – near if not beyond the limits of understanding possible to the most mature and experienced human minds. No AI program could even comprehend, let alone paraphrase, Hofstadter's prose. But his paradigm of human verbalization is Lewis Carroll's famous Jabberwock poem ('Twas brillig, and the slithy toves . . .) *plus* French and German *translations* of it (p. 366). One need not be an AI skeptic to wonder if there will ever be a computer-generated English interpretation or a foreign-language translation of that brief masterpiece. It's also questionable whether a computer-graphics program will be able to draw something that a human critic would take as the work of Escher, or whether a pattern-recognition program will identify an Escher drawing. Also, are music-generator programs ready to write a fugue that would be mistaken for a work of Bach? Perhaps the classic "million monkeys banging away on a million keyboards for a million years" could stumble on it once, but that wasn't quite Bach's technique! Still, his work was so highly structured that an imitation might be programmable. (For non-musical readers, a fugue has one melodic theme continuously present in many variations, with subdivision into several "parts" that each contain an independent though consonant melody, creating diversity within an underlying unity. "Rules" are known that restrict the composition, but no one can tell what proportion of machine-made fugues would be near Bach-level!)

The book is too mixed-up to be called a fugue. It's more like a kaleidoscope, with elements being somewhat haphazardly rearranged into patterns, one or more of which are likely to prove interesting to nearly any kind of reader. That's why it's being so widely read. There's even a mystical streak (on p. xxi, the book is said to be a statement of the author's religion!) and an AI version of GOD is cited in the index, with ITs picture (non-sexist, since IT's a computer) on p. 143. This new ITian mythology has enemies, and Hofstadter crosses swords with the English philosopher J. R. Lucas, who first used the incompleteness theorem of Gödel in an anti-AI article (*Minds, Machines, and Gödel*, 1961). Symmetry suggests the name IDian (though this strays from Freud's concept of the ID) for the anti-ITian point of view. Though myself a IDian, I tend to agree that Gödel did *not* prove that AI is impossible. Readers who would like a less argumentative exposition of the current dilemmas in mathematics can find it in Morris Kline's *Mathematics, The Loss of Certainty* (Oxford Univ. Press, 1980). The only way to prove that a hardware/software system can do anything the human mind can do is to build one that does. That's a very tough problem, but the real IDian point is that it's not the *only* problem. The more complex a program becomes, the more likely it will contain very subtle and undetectable bugs that will cause it to do things you wish it hadn't! It will have "a mind of its own," more probably crazy than sane, possibly more dangerous than useful. In the 1979 edition of his *What Computers Can't Do, The Limits of Artificial Intelligence*, Hubert Dreyfus draws an anlaogy between believers in AI and the ancient alchemists, who believe it would be possible to transmute lead into gold, if only you kept trying. Their work led to modern chemistry, that *cannot* do transmutation though it does things that are useful as well as (in many senses of the word) dangerous. The newer science of nuclear physics *can* do transmutations and understands how formidable the problem of changing lead into gold really is, so it's not even trying. But even the useful things it's doing involve serious hazards. Although computer programs are still stupid, they've been wiring me a lot of junk mail with a "personalized" look, and I'm not looking forward to locking horns with a really bright human-simulation program!

*(Continued on page 40)*

# Using WORD-MASTER with the Compal-80

by Anthony Skjellum


## Abstract

WORD-MASTER is a powerful, screen-oriented text editor which runs under the CP/M operating system. The editor makes use of hardware-dependent features which require system-dependent software routines. The standard package includes such patches for several standard hardware configurations. This article provides the routines necessary to interface the WORD-MASTER to the Compal-80 computer. Similar steps could be used for other computers which use the Polymorphic VTI display.


## Introduction

The WORD-MASTER utilizes several standard terminal features to provide screen-oriented editing capability. These are cursor address, "delete characters to end-of-line" (deol), and clear screen. Many video terminals provide these features internally. They are accessed via an "escape sequence" which instructs the terminal to perform the requested function. However, the Compal-80 uses a memory-mapped display which has no intrinsic features for display manipulation. Terminal functions such as clear screen are emulated with software within the Compal-80 (v1.5) monitor program. This program provides the lowest-level interface between the operator and system hardware and contains a video driver for the VTI display. However, the WORD-MASTER patches must still provide the cursor addressing and "delete to end-of-line" (deol) features, since they are not implemented in the video driver.

## Terminal Functions

Before continuing, it is well to define exactly what is meant by "cursor addressing" and "delete to end-of-line." Essentially, cursor addressing is the process by which the cursor is moved from its present position to another location. The new location is specified by a row-and-column address. The patch in Listing I (TCURSOR) converts the row-and-column address provided by WORD-MASTER into an absolute address for the cursor and moves it there. The character under the cursor (STR) is also maintained correctly.

While cursor positioning retains display integrity, the "deol" function is used to delete parts of the display screen. Characters from the present cursor position to the end of the current line are replaced by spaces. This operation does not alter the position of the cursor. Both of these functions are essential to producing a screen-oriented editing environment.

## Software Patches

Within the WORD-MASTER program an area is specified for hardware-dependent routines. The length of the patches for both "deol" and cursor addressing is larger than the provided area. Therefore, the implementation was divided into two segments. Cursor addressing was placed directly within the patch area and "deol" was placed in CP/M's CBIOS (Customized Basic Input-Output System) under CONOUT: function (console output). The patch within CBIOS (COMOUT) interprets ↑D as the "deol" command character and acts accordingly when such a code is received.

Several routines are present within VTI.ASM (Listing I). For clarity, each will be described briefly. CLRSCRN is the clear screen function. It transmits a ↑F (FF) as required by the Compal monitor. As stated above, TCURSOR is called when cursor addressing needs to be performed. It handles all display manipulations as well as calculations. Finally, the HITE and WID constants are set at 16 and 64 to reflect the dimensions of the VTI display.

The routines which reside directly within WORD-MASTER are presented in Listing I (VTI.ASM). This assembly-language file is assembled with ASM and linked into WORD-MASTER with DDT and the CCP (Console Command Processor) Save command. The CBIOS program must also be reassembled to reflect the "deol" patch and new CP/M systems have to be generated with SYSGEN. The necessary routine is given in Listing II. Both programs directly manipulate the internal variables of the monitor's video driver and still utilize this code for text display, scrolling and cursor positioning.

## Conclusion

With the routines presented in this article, WORD-MASTER can be interfaced with the Compal-80 hardware environment. A similar approach could be used to interface other systems with memory-mapped displays. ■■

(LISTING ON PAGE 26)

*Copyright 1981 © by Anthony Skjellum. All rights reserved. 1695 Shenandoah Road, San Marino, CA 91108.*

*CP/M is a trademark of Digital Research. WORD-MASTER is a trademark of Micropro International. COMPAL is a trademark of COMputer Power And Light, Inc.*


**DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia**

**Subscription Rates:** $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. dollars, drawn on a U.S. bank.

**Writer's Guidelines:** All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuacript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

**Donating Subscribers**

**Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives five one-year subscriptions).

**Contributing Subscribers:** Robert L. Edens, Eric P. Hanson, J. G. Zeratsky, Mervin E. Frank, Edward W. Carr, Steven Fisher, Thomas H. Grohne, Richard Washburn, Les Hancock, G. V. Elkins, C. A. Forsberg, Daniel R. Lunsford, J. D. Griffith, Michael Slade, Thomas M. Schaefges, General Electric, Louis L. Odette, Robert M. Connors, William J. McEown, Robert C. Luckey, Kim Leary, Robert N. Stabler, Dewitt S. Brown, John B. Palmer, E. Glion Curtis, Roger Hanson, Jack Miller, Burks A. Smith.

**Retaining Subscribers:** Mike Faithful, Osamu Shimizu, Scott B. Guthery.

**Sustaining Subscribers:** Gordon Eubanks, Jr. and Robert C. Birss, Ph.D.

**Foreign Distributors**

**UK & Europe:** LP Enterprises, 8/11 Cambridge House, Cambridge Road, Barking, Essex, 1G11 8NT, England (Tel. 01-591-6511, TLX 892395). Hofacker-Verlag, Tegernseer Strasse 18, D-8150 Holzkirchen, West Germany. Computerland/Computer Store AB, Box 7134, Kungsgatan 19, S-10387 Stockholmn, Sweden.

**Asia & Australia:** Electronic Concepts Pty Ltd., 55 Clarence Street, Sydney, NSW2000, Australia. Computer Store, POB 31-261, 22B Milford Road, Milford, Auckland 9, New Zealand. ASCII Publishing Co., Ltd., Aoyama Building 5F, 5-16-1 Minami Aoyama, Minato-ku, Tokyo 107, Japan.

# Byte Manipulation and Byte Pointers

## by Edwin E. Freed

*Edwin E. Freed, Physics Department, Oklahoma State University, Stillwater, Oklahoma.*

On a microcomputer, bytes are usually assumed to be 8-bit quantities residing at unique addresses in memory. This is only a special case of a general concept in use on larger computers for some time. Some machines boast instruction sets that support operations on variable-length bytes located at arbitrary positions inside words. Some advantages of these operations are:

(1) Efficient memory usage is accomplished by allocating only as many bits as are necessary to represent a value. This is particularly useful in programming languages like Pascal which allow variables to be declared as a subrange of integers.

(2) Efficient array storage of data with unusual bit lengths can be implemented without wasting any space between elements. Pascal uses this in implementing the PACKED ARRAY facility.

(3) Testing and modification of both single bits and groups of bits in words are made simple.

(4) Assemblers and disassemblers, which frequently need to manipulate bytes within words, can make use of byte operations.

(5) Data packing programs, particularly those which use Huffman encoding, require some method for variable-length byte access for optimum operation.

I first became acquainted with byte manipulation instructions during several years of programming on a Decsystem-10. In making the transition to microcomputers, I found I missed having byte-mode operations readily available, and resolved to write routines to provide these capabilities. This article presents such a set of routines based on the DEC-10 instruction set, written in PDP-11 FORTRAN. If high-speed operation is desired, the routines could be easily translated into assembly language.

In order to deal with bytes, some quantity analogous to a memory address must be used to indicate the position of the byte in memory. This quantity is called a byte pointer, and it consists of three parts. The first part is the length of the byte in bits. The second is the bit position of the rightmost bit of the byte in a computer word, and the third is the address in memory of the word which contains the rightmost bit. On the PDP-11, the pointer is stored in two words, using one half-word (8 bits) for the size, another for the length, and a full word for the address. Access to the various fields of the pointers is provided by the routines POINT and UNPNT. These routines serve to insulate the user from the internal structure of the pointers.

The two basic operations performed using byte pointers are insertion and extraction of bytes. The two routines LDB (LoaD Byte) and DPB (DePosit Byte) accomplish these functions. An LDB is analogous to a memory read, and a DPB is analogous to a memory write. Note that in a load operation, the other bits in the word are ignored and the extracted quantity is right justified. In a deposit, the other bits in a word are unchanged.

In the DEC-10 implementation of byte pointers, the bytes were confined to a single word. On micros, where every bit counts, I feel the additional code to support bytes spread between words is justified.

Some method of stepping from byte to byte is needed to support tables of bytes and similar constructs. The routine IBP (Increment Byte Pointer) adjusts a pointer to address the next successive byte in memory. DBP (Decrement Byte Pointer) sets the pointer to the preceding byte. ADJBP is a more general-purpose routine that can be used to adjust a pointer by any offset, positive or negative. It is much slower than IBP and DBP. ADJABP is used to adjust the address portion only of a pointer. For programming efficiency, some combined operation routines are provided. ILDB performs the functions of an IBP followed by an LDB. DLDB, IDPB, and DDPB perform analogous functions (see listing).

All the routines are implemented as FORTRAN subroutines or functions. The specific call formats are included in the listing. Note that byte sizes larger than 16 bits are not supported. A word length of 16 bits is assumed. If this is not the case, various constants and tables will need to be changed.

One non-standard FORTRAN feature is used – the LOGICAL*1 variable type. This is the accepted way in PDP-11 FORTRAN to access 8-bit quantities. These are used to access the parts of a pointer easily and can be converted to INTEGER data at the expense of making the pointers longer. Some external assembly language routines are also used. SFTLFT (V,N) shifts the integer V left by N bits, and SFTRGT has the obvious effect. Both of these routines are included as MACRO-11 code; their implementation in other assembly languages should be easy. The external routines IPEEK and IPOKE are used to read and write into a physical memory location. Another routine to find the physical address of a FORTRAN variable is usually necessary. This routine called IADDR, along with IPEEK and IPOKE, is provided on the PDP-11 as library subroutines. Many other FORTRAN implementations will also provide them.

In concusion, byte pointers and byte manipulation routines are a useful tool that should be part of any programmer's bag of tricks. The coding of the routines is straightforward and can change many programming tasks from a headache into a pleasurable experience.

This software was developed as part of a subroutine package for physics research. The research was sponsored by the National Science Foundation. ■■

**(LISTING ON PAGE 22)**

THIS PUBLICATION IS AVAILABLE IN MICROFORM

**University Microfilms International**

300 North Zeeb Road
Dept. P.R.
Ann Arbor, MI 48106
U.S.A.

18 Bedford Row
Dept. P.R.
London, WC1R 4EJ
England

Name _____
Institution _____
Street _____
City _____
State _____ Zip _____

# Small-C: Current Status

by P. L. Woods

*P. L. Woods, 78 Dukes Avenue, Muswell Hill, London N10 2QA England.*

My original article "Small-C: An Implementor's Notes and a Bug Corrected" (*DDJ* #52) was written during the testing stage of the FORTRAN version of the Small-C compiler. This testing was completed at the beginning of November 1980, and I then started to construct a version of the compiler written in Small-C so that I could transfer the compiler across to my home computer system based on a Z-80.

This would, however, take some time, and in order to get my first program in Small-C up and running, and also so as to have something other than an assembler to demonstrate my system, I decided to use the FORTRAN version of the compiler to compile a version of the game "Nim" which I had written. Although the program was short, only forty-nine lines in length, the code generated was about 360 lines long. After typing this all in by hand, followed by a one thousand line library file for Small-C, it was clear that loading the compiler by typing in assembler was not really a viable proposition despite my earlier hopes.

I therefore set to work on two aids to lessen the amount to be typed. They both involved modifications to the FORTRAN version of the Small-C compiler. The first was trivial to carry out, and was to make the compiler generate Z-80 mnemonics rather than Intel 8080 code for its output. This avoided the need to make a mental translation from the 8080 code produced by the compiler to the Zilog mnemonics that my assembler accepts.

The second modification, which is now almost complete, was to add a second pass to the compiler so that it would generate its output as relocatable machine code rather than as assembler; the idea being that machine code takes far fewer key strokes than does the assembler, and so should be faster to enter. It is probably also more error prone, but simple checksums can easily be used to detect these errors. I have not yet been able to try this system, so do not know which is more boring – entering machine code or assembler. As an easier alternative, I am also on the lookout for a secondhand paper tape punch that I can attach to my Z-80 system so I can punch the cross-compiled code onto tape and load this mechanically with no effort on my part.

As an example of what my second pass of the compiler can produce, refer to the listing (see page 36). This is the first three functions for the original Small-C compiler. The listing has been edited somewhat from that produced by the compiler so as to accent those areas which have been modified. The first hundred lines or so, where the static (global) variables were defined, have been deleted together with most of the assembler mnemonics produced. The symbol definitions have, however, been left in to aid anyone who goes through the machine code produced, by hand.

The final section of the listing shows the relocatable machine code produced in hexadecimal notation. The first number on each line (in parentheses) is the address at which the code is to be loaded before relocation. Addresses to be relocated are indicated in two ways, depending on whether or not they are yet defined.

If the address is defined, then the fully computed address is printed (in byte-reversed order) after the appropriate opcode and followed by the $ symbol to indicate that the relocation offset must be added in at that point when the code is loaded. Thus the instruction placed at address 0003H is a reference to memory at 15B2H which must have the relocation offset added in as indicated by the $ sign.

If the address is not defined, usually because of a call to a precompiled library or system routine, or a reference to a static variable, then the second format is used. This is similar to the first format. The offset from the external (undefined) name follows the opcode (again in byte-reversed order), and this is followed by the external symbol, whose address must be added into the offset at this point when the loader knows its value. As an example, consider the instruction at 0012H. This is a subprogram call to CCMULT (in the run-time library) and there is a zero offset from this address.

The second section of the machine code produced, from address 01E3H, is the literal constant area.

The third section is a list of the addresses of all the labels generated by the compiler. These are *not* in byte-reversed order.

The fourth and final section is a symbol table which contains entries for all external symbols used or defined in the code, and would be used by the loader to link together separately compiled modules into an executable program. The type of the symbol is printed (variable or function), together with the address if known (*not* in byte-reversed order) or the word UNDEF if unknown (undefined).

The machine code is produced in almost exactly the same way Mr. Cain originally generated the assembler mnemonics. In fact, after every statement which generates a mnemonic, I have inserted an extra function call which causes the opcode associated with that mnemonic to be stored away until the code generation phase (second pass) of the compiler. There is thus not a full assembler built into the compiler, and as a result I have to inhibit the use of the "#ASM" and "#ENDASM" pseudo-operations. This turns out to be a not-inconvenient restriction.

A little further work has to be carried out during the first pass in order to remember the address of each symbol as it is defined so that memory reference and jump instructions may be correctly completed with their destination addresses. The symbol map is a by-product of this process.

This, then, is the progress since the writing of my original submission. I am afraid it seems at times to be a list of problems, after re-reading my original manuscript, but then one only ever writes up after a success, and then carefully omits most references to the difficulties experienced en route. ■■

(LISTING ON PAGE 36)

Since 1976, the most innovative magazine for the technically-sophisticated microcomputer user.

# Dr. Dobb's Journal

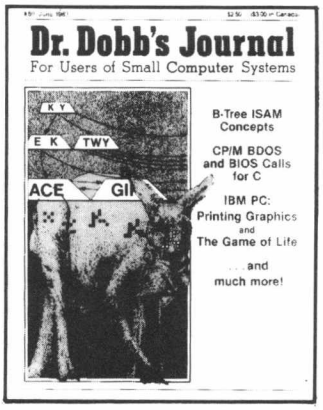
## For Users of Small Computer Systems




1976



1983




Have you been looking for a magazine that will provide you with serious articles on programming? *Dr. Dobb's Journal* can give you the technical depth you need to understand the latest software advances. We cover every microprocessor with articles on C, Forth, Unix, CP/M and much more. Our reputation for editorial quality is well known. Subscribe now, and discover what industry insiders have enjoyed since 1976.

Coming articles will cover:

- new language development
- 8- and 16-bit technology
- operating systems
- compilers, editors
- assemblers/cross-assemblers
- clever algorithms

**$25**

one year subscription
12 issues
saves $5 off the newsstand price

**YES!** Enter my subscription to *Dr. Dobb's Journal* and bill me later.

name

address

city state zip

**Mail to: DDJ, P.O. Box E., Menlo Park, CA 94025**

# Byte Manipulation

## Listing I

```
FORTRAN IV      V01C-03A    SAT 13-DEC-80

      C     POINT.FOR - ROUTINES TO SIMULATE DEC-10 BYTE POINTER OPERATIONS
      C     *** PDP-11 FORTRAN VERSION ***
      C
      C     WRITTEN BY NED FREED, 16-AUG-80
      C
      C     THIS PACKAGE OF ROUTINES IS DESIGNED TO GIVE THE FORTRAN PROGRAMMER
      C     SOME ACCESS TO THE ELEGANCE OF THE DEC-10 STYLE BYTE POINTER
      C     INSTRUCTION SET.
      C
      C     A BYTE POINTER CONSISTS OF A 2-WORD QUANTITY (IN THE PDP-11
      C     IMPLEMENTATION). IT CAN BE REAL*4, INTEGER*4 OR WHAT HAVE YOU,
      C     AS THE ROUTINES DO ALL NECESSARY MANIPULATION OF THE POINTERS,
      C     AND THEY WILL DECLARE THE POINTERS TO BE WHAT THEY NEED
      C     THEM TO BE. THE FIRST WORD CONTAINS TWO 8-BIT FIELDS. THE
      C     FIRST IS THE NUMBER OF BITS THE POINTER "POINTS" AT (UP TO 16),
      C     AND THE SECOND IS THE POSITION OF THE RIGHTMOST BIT OF
      C     THE GROUP IN THE WORD (FROM 0 TO 15). THE SECOND WORD IS
      C     THE ADDRESS OF THE WORD THAT CONTAINS THE RIGHTMOST BIT OF
      C     THE OF THE GROUP IN MEMORY. NOTE THAT THIS IS NOT JUST
      C     SOME OFFSET INTO AN ARRAY. IT IS THE TRUE PHYSICAL ADDRESS.
      C     THE COMPANION ROUTINE IADDR (IN PDP-11 SYSLIB) CAN BE USED
      C     TO GET THESE ADDRESSES AT RUNTIME. IN THIS IMPLEMENTATION,
      C     BYTE POINTERS EXHIBIT WRAPAROUND. THIS MEANS THAT A BYTE IS
      C     ALLOWED TO BE SPREAD BETWEEN TWO WORDS. THIS MAKES BYTE
      C     POINTER STORAGE VERY EFFICIENT.
      C
      C     THE FOLLOWING ROUTINES ARE CONTAINED IN THIS PACKAGE:
      C       POINT - SET UP A BYTE POINTER.
      C       LDB - LOAD A BYTE THAT A POINTER ADDRESSES.
      C       ILDB - INCREMENT A POINTER AND LOAD.
      C       DPB - DEPOSIT A BYTE VIA A POINTER.
      C       IDPB - INCREMENT POINTER AND DEPOSIT BYTE.
      C       IBP - INCREMENT A BYTE POINTER (BY 1).
      C       DBP - DECREMENT A BYTE POINTER (BY 1).
      C       DLDB - DECREMENT POINTER AND LOAD FROM IT.
      C       DDPB - DECREMENT POINTER AND DEPOSIT.
      C       UNPNT - TAKE APART A POINTER FOR INSPECTION.
      C       ADJBP - ADJUST A BYTE POINTER BY ANY OFFSET.
      C       ADJABP - ADJUST ONLY THE ADDRESS PART OF A POINTER BY ANY OFFSET.
      C
      C     A NOTE ON PORTABILITY. THESE ROUTINES ARE WRITTEN FOR THE
      C     FORTRAN AVAILABLE ON THE PDP-11 ONLY. EVERY ATTEMPT HAS BEEN MADE
      C     TO USE FAIRLY STANDARD FORTRAN, BUT NOT AT THE EXPENSE OF BAD
      C     CODING PRACTICE. IN PARTICULAR, THE FOLLOWING OBJECTIONS ARE
      C     NOTED:
      C       1) THE INTEGER*2 NATURE OF THE ADDRESS PART OF THE POINTER
      C          MAY BE INADAQUATE FOR THE ADDRESSING OF MANY MACHINES.
      C       2) ONLY 16 BITS ARE ASSUMED PER ADDRESS. THIS MAY BE ACCEPTABLE
      C          ON SOME LARGER MACHINES, BUT THE LARGER WORD SIZES WILL BE
      C          WASTED.
      C       3) THE ROUTINES IPEEK AND IPOKE ARE CALLED FREQUENTLY FROM THE
      C          DEC SYSLIB PACKAGE, AND WOULD HAVE TO BE RECODED.
      C       4) THE MACRO ROUTINES CONTAINED IN LOGICF.MAC ARE NECESSARY
      C          FOR POINTER WORK, AND WILL NEED TO BE RECODED.
      C       5) THE METHOD USED TO ACCESS THE SIZE AND BIT PARTS OF A
      C          POINTER IS TO DECLARE AN INTEGER*2 VARIABLE EQUIVALENT
      C          TO TWO LOGICAL*1 VARIABLES, AND THEN ACCESS THE TWO
      C          8-BIT FIELDS THUS CREATED SEPARATELY. THIS IS A FEATURE
      C          OF PDP-11 FORTRAN, AND MAY REQUIRE ALTERATION.
      C
      C
      C     POINT - CREATE A BYTE POINTER
0005          POINTR(2) = POINTR(2) + POINTR(1)
0006  50      IF (POINTR(2).LT.16) RETURN
0008          POINTR(2) = POINTR(2) - 16
0009          LADDR(1) = POINTR(3)
0010          LADDR(2) = POINTR(4)
0011          IADDR = IADDR + 2
0012          POINTR(3) = LADDR(1)
0013          POINTR(4) = LADDR(2)
0014          GO TO 50
        C
        C     END OF SUBROUTINE IBP
0015          END


      C     DBP - DECREMENT A BYTE POINTER
      C      CALL: CALL DBP(POINTER)
      C      ARGUMENTS: AS FOR IBP.
      C
0001          SUBROUTINE DBP(POINTR)
      C
      C     VARIABLE DECLARATIONS
0002          INTEGER*2 IADDR
0003          LOGICAL*1 POINTR(4),LADDR(2)
0004          EQUIVALENCE (IADDR,LADDR(1))
      C
0005          POINTR(2) = POINTR(2) - POINTR(1)
0006  50      IF (POINTR(2).GE.0) RETURN
0008          POINTR(2) = POINTR(2) + 16
0009          LADDR(1) = POINTR(3)
0010          LADDR(2) = POINTR(4)
0011          IADDR = IADDR - 2
0012          POINTR(3) = LADDR(1)
0013          POINTR(4) = LADDR(2)
0014          GO TO 50
      C
      C     END OF SUBROUTINE DBP
0015          END

      C     ADJABP - ADJUST JUST THE ADDRESS PART OF A POINTER
      C      CALL: CALL ADJABP(POINTER,CHANGE)
      C      ARGUMENTS:
      C        POINTER - A STANDARD 2-WORD POINTER.
      C        CHANGE - A SIGNED INTEGER*2 VALUE TO ADD TO THE ADDRESS.
      C
0001          SUBROUTINE ADJABP(POINTR,CHANGE)
      C
      C     DECLARE THE VARIABLES
0002          INTEGER*2 POINTR(2),CHANGE
      C
0003          POINTR(2) = POINTR(2) + CHANGE
0004          RETURN
      C
      C     END OF SUBROUTINE ADJABP
0005          END

      C     ADJBP - ADJUST A BYTE POINTER BY SOME AMOUNT
      C      CALL: CALL ADJBP(POINTER,OFFSET)
      C      ARGUMENTS:
      C        POINTER - A BYTE POINTER.
      C        OFFSET - A SIGNED INTEGER*2 INDICATING THE NUMBER OF BYTES
      C                 TO OFFSET THE POINTER BY.
      C
      C     NOTE THAT ADJBP WITH OFFSET.EQ.0 CAN BE USED TO NORMALIZE BYTE
      C     POINTERS.
      C
```

```
      C      CALL: CALL POINT(SIZE,BIT,ADDR,POINTER)
      C      ARGUMENTS:
      C        SIZE - THE SIZE OF THE BIT FIELD (LESS THAN 17).
      C        BIT - THE RIGHTMOST BIT OF THE FIELD (LESS THAN 16).
      C        ADDR - THE ADDRESS IN MEMORY OF THE STUFF.
      C        POINTER - THE RESULTANT TWO-WORD POINTER VALUE.
      C
0001        SUBROUTINE POINT(SIZE,BIT,ADDR,POINTR)
      C
      C     VARIABLE DECLARATIONS
0002        INTEGER*2 SIZE,BIT,ADDR,IADDR
0003        LOGICAL*1 POINTR(4),LADDR(2)
0004        EQUIVALENCE (IADDR,LADDR(1))
      C
0005        POINTR(1) = SIZE .AND. "37
0006        POINTR(2) = BIT .AND. "17
0007        IADDR = ADDR
0008        POINTR(3) = LADDR(1)
0009        POINTR(4) = LADDR(2)
0010        RETURN
      C
      C     END OF SUBROUTINE POINT
0011        END


      ..
      C     UNPNT - TAKE APART A POINTER FOR INSPECTION
      C      CALL: CALL UNPNT(POINTER,SIZE,BIT,ADDR)
      C      ARGUMENTS: SAME AS FOR POINT ROUTINE ABOVE.
      C
0001        SUBROUTINE UNPNT(POINTR,SIZE,BIT,ADDR)
      C
      C     VARIABLE DECLARATIONS
0002        INTEGER*2 SIZE,BIT,ADDR,IADDR
0003        LOGICAL*1 POINTR(4),LADDR(2)
0004        EQUIVALENCE (IADDR,LADDR(1))
      C
0005        SIZE = POINTR(1)
0006        BIT = POINTR(2)
0007        LADDR(1) = POINTR(3)
0008        LADDR(2) = POINTR(4)
0009        ADDR = IADDR
0010        RETURN
      C
      C     END OF SUBROUTINE UNPNT
0011        END


      C     IBP - INCREMENT A BYTE POINTER
      C      CALL: CALL IBP(POINTER)
      C      ARGUMENTS:
      C        POINTER - ANY OLE BYTE POINTER YA FEEL LIKE.
      C
0001        SUBROUTINE IBP(POINTR)
      C
      C     VARIABLE DECLARATIONS
0002        INTEGER*2 IADDR
0003        LOGICAL*1 POINTR(4),LADDR(2)
0004        EQUIVALENCE (IADDR,LADDR(1))
      C
```

*(Continued on the top of page 22)*

```
0001        SUBROUTINE ADJBP(POINTR,OFFSET)
      C
      C     DECLARE VARIABLES
0002        INTEGER*2 OFFSET,IADDR,RBIT,SIZE,BITVAL
0003        LOGICAL*1 POINTR(4),LADDR(2)
0004        EQUIVALENCE (IADDR,LADDR(1))
      C
      C     GET THE PROPER VALUES TO DEAL WITH
0005        SIZE = POINTR(1)
0006        RBIT = POINTR(2)
0007        LADDR(1) = POINTR(3)
0008        LADDR(2) = POINTR(4)
      C
      C     DO THE WORD OFFSETTING
0009        IADDR = IADDR + ((OFFSET * SIZE) / 16) * 2
      C
      C     NOW DO THE REQUIRED NUMBER OF ACTUAL BIT MOTIONS
0010        BITVAL = MOD(IABS(OFFSET * SIZE),1)
0011        IF (OFFSET.GT.0) GO TO 200
      C
      C     DECREMENT OPERATION
0013  100   RBIT = RBIT - BITVAL
0014  150   IF (RBIT.GE.0) GO TO 250
0016        IADDR = IADDR - 2
0017        RBIT = RBIT + 16
0018        GO TO 150
      C
      C     INCREMENT OPERATION
0019  200   RBIT = RBIT + BITVAL
0020  250   IF (RBIT.LT.16) GO TO 300
0022        IADDR = IADDR + 2
0023        RBIT = RBIT - 16
0024        GO TO 250
      C
      C     RESTORE ADDRESS AND RETURN
0025  300   POINTR(1) = SIZE
0026        POINTR(2) = RBIT
0027        POINTR(3) = LADDR(1)
0028        POINTR(4) = LADDR(2)
0029        RETURN
      C
      C     END OF SUBROUTINE ADJBP
0030        END


      C     LDB - LOAD A BYTE FROM A BYTE POINTER
      C      CALL: VALUE=LDB(POINTER)
      C      ARGUMENTS:
      C        VALUE - THE BYTE RETURNED BY THE POINTER.
      C        POINTER - THE POINTER TO USE.
      C
0001        INTEGER FUNCTION LDB(POINTR)
      C
      C     VARIABLES TO DECLARE
0002        INTEGER*2 IADDR,IBIT,ILEFT
0003        INTEGER*2 IPEEK,SFTLFT,SFTRGT
0004        LOGICAL*1 POINTR(4),LADDR(2)
0005        INTEGER*2 POWER(16)
```

*(Continued on next page)*

# Byte Manipulation

```
0006          EQUIVALENCE (IADDR,LADDR(1))
      C
      C       DATA FOR THE (2**N) - 1 POWERS-THAT-BE
0007          DATA POWER /1,3,7,15,31,63,127,255,511,1023,2047,
     *4095,8191,16383,32767,-1/
      C
      C       EXTRACT THE LOW ORDER BITS FROM THE OBVIOUS WORD
0008          LADDR(1) = POINTR(3)
0009          LADDR(2) = POINTR(4)
0010          IBIT = IPEEK(IADDR)
0011          IBIT = SFTRGT(IBIT,15-POINTR(2)) .AND. POWER(POINTR(1))
      C
      C       SEE IF ANY ARE NEEDED FROM THE PREVIOUS WORD
0012          ILEFT = POINTR(1) - (POINTR(2) + 1)
0013          IF (ILEFT.GT.0) GO TO 100
0015          LDB = IBIT
0016          RETURN
      C
      C       GET THE STUFF FROM THE PREVIOUS WORD, THEN
0017  100     ILEFT = IPEEK(IADDR-2) .AND. POWER(ILEFT)
0018          LDB = IBIT + SFTLFT(ILEFT,POINTR(2)+1)
0019          RETURN
      C
      C       END OF SUBROUTINE LDB
0020          END


      C       ILDB - INCREMENT, THEN LOAD FROM A BYTE POINTER
      C        CALL: VALUE = ILDB(POINTER)
      C        ARGUMENTS: AS FOR LDB.
      C
0001          INTEGER FUNCTION ILDB(POINTR)
      C
      C       VARIABLES TO DECLARE
0002          INTEGER*2 IADDR,IBIT,ILEFT
0003          INTEGER*2 IPEEK,SFTLFT,SFTRGT
0004          LOGICAL*1 POINTR(4),LADDR(2)
0005          INTEGER*2 POWER(16)
0006          EQUIVALENCE (IADDR,LADDR(1))
      C
      C       DATA FOR THE (2**N) - 1 POWERS-THAT-BE
0007          DATA POWER /1,3,7,15,31,63,127,255,511,1023,2047,
     *4095,8191,16383,32767,-1/
      C
      C       DO THE INCREMENT OPERATION
0008          LADDR(1) = POINTR(3)
0009          LADDR(2) = POINTR(4)
0010          POINTR(2) = POINTR(2) + POINTR(1)
0011  50      IF (POINTR(2).LT.16) GO TO 75
0013          POINTR(2) = POINTR(2) - 16
0014          IADDR = IADDR + 2
0015          POINTR(3) = LADDR(1)
0016          POINTR(4) = LADDR(2)
0017          GO TO 50
      C
      C       EXTRACT THE LOW ORDER BITS FROM THE OBVIOUS WORD
0018  75      IBIT = IPEEK(IADDR)
0019          IBIT = SFTRGT(IBIT,15-POINTR(2)) .AND. POWER(POINTR(1))
      C
      C       SEE IF ANY ARE NEEDED FROM THE PREVIOUS WORD
0020          ILEFT = POINTR(1) - (POINTR(2) + 1)
0021          IF (ILEFT.GT.0) GO TO 100
0023          ILDB = IBIT
0024          RETURN
      C
```

```
0014          ILEFT = POINTR(1) - (POINTR(2) + 1)
0015          IF (ILEFT.LE.0) RETURN
      C
      C       CREATE A NEW MASK FOR THE PREVIOUS WORD, THEN
0017          IBIT = IPEEK(IADDR-2) .AND. (.NOT.(POWER(ILEFT)))
      C
      C       THEN INSTALL IT ALL
0018          CALL IPOKE(IADDR-2,IBIT + SFTRGT(STUFF,POINTR(2)+1))
0019          RETURN
      C
      C       END OF SUBROUTINE DPB
0020          END

      C       IDPB - INCREMENT, THEN DEPOSIT A BYTE VIA A BYTE POINTER
      C       CALL: CALL IDPB(POINTER,VALUE)
      C       ARGUMENTS:
      C         POINTER - A STANDARD BYTE POINTER.
      C         VALUE - THE QUANTITY TO BE SHOVED IN THERE.
      C
0001          SUBROUTINE IDPB(POINTR,STUFF)
      C
      C       POTPOURRI OF VARIABLES
0002          INTEGER*2 STUFF,IADDR,IMASK,IBIT,ILEFT
0003          INTEGER*2 SFTLFT,SFTRGT,IPEEK,IPOKE
0004          LOGICAL*1 POINTR(4),LADDR(2)
0005          INTEGER*2 POWER(16)
0006          EQUIVALENCE (IADDR,LADDR(1))
0007          EXTERNAL SFTRGT,SFTLFT
      C
      C       DATA FOR THE POWERS
0008          DATA POWER /1,3,7,15,31,63,127,255,511,1023,2047,
     *4095,8191,16383,32767,-1/
      C
      C       FIRST, GET THE ADDRESS STRAIGHTENED OUT
0009          LADDR(1) = POINTR(3)
0010          LADDR(2) = POINTR(4)
      C
      C       DO THE DECREMENT, THEN
0011          POINTR(2) = POINTR(2) + POINTR(1)
0012  50      IF (POINTR(2).LT.16) GO TO 75
0014          POINTR(2) = POINTR(2) - 16
0015          IADDR = IADDR + 2
0016          POINTR(3) = LADDR(1)
0017          POINTR(4) = LADDR(2)
0018          GO TO 50
      C
      C       CREATE A BIT MASK FOR THE FIRST WORD
0019  75      IMASK = SFTLFT(POWER(POINTR(1)),15-POINTR(2))
      C
      C       GET ONLY THE VALID BITS IN THE OLD WORD
0020          IBIT = IPEEK(IADDR) .AND. (.NOT.(IMASK))
      C
      C       AND INSTALL THE NEW STUFF
0021          CALL IPOKE(IADDR,IBIT + (SFTLFT(STUFF,15-POINTR(2)).AND.IMASK))
      C
      C       SEE IF ANY BITS FALL INTO THE PREVIOUS WORD
0022          ILEFT = POINTR(1) - (POINTR(2) + 1)
0023          IF (ILEFT.LE.0) RETURN
      C
      C       CREATE A NEW MASK FOR THE PREVIOUS WORD, THEN
0025          IBIT = IPEEK(IADDR-2) .AND. (.NOT.(POWER(ILEFT)))
      C
      C       THEN INSTALL IT ALL
0026          CALL IPOKE(IADDR-2,IBIT + SFTRGT(STUFF,POINTR(2)+1))
0027          RETURN
```

```
      C     GET THE STUFF FROM THE PREVIOUS WORD, THEN
0025  100   ILEFT = IPEEK(IADDR-2) .AND. POWER(ILEFT)
0026        ILDB = IBIT + SFTLFT(ILEFT,POINTR(2)+1)
0027        RETURN
      C
      C     END OF SUBROUTINE ILDB
0028        END


      C     DLDB - DECREMENT, THEN LOAD FROM A BYTE POINTER
      C      CALL: VALUE = ILDB(POINTER)
      C      ARGUMENTS: AS FOR LDB.
      C
0001        FUNCTION DLDB(POINTR)
      C
      C     VARIABLES
0002        INTEGER*2 POINTR(2),DLDB
      C
      C     DO THIS FUNCTION BY CALLS, AS IT IS INFREQUENTLY USED
0003        CALL DBP(POINTR)
0004        DLDB = LDB(POINTR)
0005        RETURN
      C
      C     END OF SUBROUTINE DLDB
0006        END

      C     DPB - DEPOSIT A BYTE VIA A BYTE POINTER
      C     CALL: CALL DPB(POINTER,VALUE)
      C     ARGUMENTS:
      C       POINTER - A STANDARD BYTE POINTER.
      C       VALUE - THE QUANTITY TO BE SHOVED IN THERE.
      C
0001        SUBROUTINE DPB(POINTR,STUFF)
      C
      C     POTPOURRI OF VARIABLES
0002        INTEGER*2 STUFF,IADDR,IMASK,IBIT,ILEFT
0003        INTEGER*2 SFTLFT,SFTRGT,IPEEK,IPOKE
0004        LOGICAL*1 POINTR(4),LADDR(2)
0005        INTEGER*2 POWER(16)
0006        EQUIVALENCE (IADDR,LADDR(1))
0007        EXTERNAL SFTRGT,SFTLFT
      C
      C     DATA FOR THE POWERS
0008        DATA POWER /1,3,7,15,31,63,127,255,511,1023,2047,
           *4095,8191,16383,32767,-1/
      C
      C     FIRST, GET THE ADDRESS STRAIGHTENED OUT
0009        LADDR(1) = POINTR(3)
0010        LADDR(2) = POINTR(4)
      C
      C     CREATE A BIT MASK FOR THE FIRST WORD
0011        IMASK = SFTLFT(POWER(POINTR(1)),15-POINTR(2))
      C
      C     GET ONLY THE VALID BITS IN THE OLD WORD
0012        IBIT = IPEEK(IADDR) .AND. (.NOT.(IMASK))
      C
      C     AND INSTALL THE NEW STUFF
0013        CALL IPOKE(IADDR,IBIT + (SFTLFT(STUFF,15-POINTR(2)).AND.IMASK))
      C
      C     SEE IF ANY BITS FALL INTO THE PREVIOUS WORD
```

*(Continued on the top of page 24)*

```
      C
      C     END OF SUBROUTINE IDPB
0028        END

      C     DDPB - DECREMENT, THEN DEPOSIT VIA A BYTE POINTER
      C      CALL: CALL DDPB(POINTER,VALUE)
      C      ARGUMENTS: AS IN DPB.
      C
0001        SUBROUTINE DDPB(POINTR,VALUE)
      C
      C     STANDARD VARIABLES
0002        INTEGER*2 POINTR(2),VALUE
      C
      C     DO IT BY CALLS - SLOWER, BUT SMALLER
0003        CALL DBP(POINTR)
0004        CALL DPB(POINTR,VALUE)
0005        RETURN
      C
      C     END OF POINT.FOR
0006        END
```

**end**

# Byte Manipulation

## Listing II

```
LOGICF.MAC - FORTRAN LOGICAL OP RT-11 MACRO VM02-12


1                       .TITLE  LOGICF.MAC - FORTRAN LOGICAL OPERATIONS
2                   ;
3                   ; WRITTEN BY NED FREED, 25-AUG-80
4                   ;
5                   ;THIS PACKAGE OF ROUTINES PROVIDES FORTRAN WITH A COLLECTION OF
6                   ;LOGICAL OPERATIONS AND OTHER NICETIES THAT ARE BEST CODED IN
7                   ;MACHINE LANGUAGE. THE ROUTINES ARE:
8                   ;    ROLLFT - RORATE A 16-BIT QUANTITY N BITS LEFT, CIRCULAR.
9                   ;    ROLRGT - ROTATE A 16-BIT QUANTITY N BITS RIGHT, CIRCULAR.
10                  ;    SFTLFT - SHIFT A 16-BIT QUANTITY N BITS LEFT, NONCIRCULAR.
11                  ;    SFTRGT - SHIFT A 16-BIT QUANTITY N BITS RIGHT, NONCIRCULAR.
12                  ;
13                  ;THIS SET OF ROUTINES SHOULD BE INSTALLED IN THE SAME SUBROUTINE
14                  ;LIBRARY AS THE POINT PACKAGE.
15                  ;
16                  ;GET SOME STUFF FROM SYSMAC.SML
17
18                      .MCALL  .REGDEF,..V2..
19  00000           .REGDEF
20  00000               ..V2..
21                  ;
22                  ;
23                  ;ROLLFT - ROUTINE TO ROTATE A 16-BIT QUANTITY N BITS LEFT
24                  ;CALL: VALUE=ROLLFT(QUAN16,NUMBIT)
25                  ;ARGUMENTS:
26                  ;    QUAN16 - THE INTEGER 16-BIT QUANTITY TO BE ROTATED.
27                  ;    NUMBIT - NUMBER OF BITS TO ROTATE OVER.
28                  ;
29          000000'         .CSECT
30                          .GLOBL  ROLLFT
31
32  00000 005725 ROLLFT: TST     (R5)+           ;INCREMENT PAST ARGUMENT COUNT
33  00002 013500         MOV     @(R5)+,R0       ;GET THAT INTEGER VALUE
34  00004 013501         MOV     @(R5)+,R1       ;AND GET THE COUNTER OF BITS
35  00006 001405         BEQ     2$              ;RETURN IF ZERO, ALSO
36
37  00010 000241 1$:     CLC                     ;CLEAR THE CARRY IN CASE
38  00012 006100         ROL     R0              ;ROLL IT OVER THERE
39  00014 005500         ADC     R0              ;ADD IN THE HIGH BIT TOO
40  00016 105301         DECB    R1              ;DECREMENT THE COUNT
41  00020 001373         BNE     1$              ;AND LOOP FOR MORE
42  00022 000207 2$:     RTS     PC              ;BACK TO CALLER
43                  ;
44                  ;
45                  ;ROLRGT - ROUTINE TO ROTATE A 16-BIT QUANTITY N BITS RIGHT
46                  ;CALL: VALUE=ROLRGT(QUAN16,NUMBIT)
47                  ;ARGUMENTS:
48                  ;    QUAN16 - THE INTEGER 16-BIT QUANTITY TO BE ROTATED.
49                  ;    NUMBIT - NUMBER OF BITS TO ROTATE OVER.
50                  ;
51          000024'         .CSECT
52                          .GLOBL  ROLRGT
53
54  00024 005725 ROLRGT: TST     (R5)+           ;INCREMENT PAST ARGUMENT COUNT
55  00026 013500         MOV     @(R5)+,R0       ;GET THAT INTEGER VALUE
56  00030 013501         MOV     @(R5)+,R1 ;AND GET THE COUNTER OF BITS
57  00032 001407         BEQ     2$              ;RETURN IF ZERO, ALSO



58
59  00034 000241 1$:     CLC                     ;CLEAR THE CARRY IN CASE
```

# Using WORD-MASTER

## Listing

LISTING I.

```
                ;*************************************************
                ;*                                               *
                ;*  USER-MODIFYABLE ROUTINES AND CONSTANTS FOR   *
                ;*  HARDWARE-DEPENDENT TERMINAL CHARACTERISTICS  *
                ;*  AND FUNCTIONS USED BY WORD-MASTER.           *
                ;*                                               *
                ;*                      RELEASE 1.061  3/10/79   *
                ;*                                               *
                ;*************************************************
                        ;
                        ;
                        ; THIS PATCH ROUTINE MODIFIED FOR COMPAL-80
                        ; BY A. SKJELLUM.
                        ;
                        ; COPYRIGHT 1980 (C)  A. SKJELLUM.
                        ; ALL RIGHTS RESERVED
                        ;
                        ; VTI.ASM CREATED FROM THE EXAMPLE PROGRAM SOL.ASM
                        ; PROVIDED BY MICROPRO INTERNATIONAL.
                        ;
                        ;
                        ;
0180                    ORG     180H
                        ;
                        ;
01EF =          OUTCHR  EQU     01EFH   ;LINKS TO WORD-MASTER
29B8 =          MEMORY  EQU     29B8H
                        ;
                        ;
                        ;
                CLRSCRN:
0180 3E06               MVI     A,CHCLS ; CLEAR CODE
0182 C3EF01             JMP     OUTCHR  ; SEND IT AND EXIT
                        ;
                ;*************************************************
                ;*                                               *
                ;*    SUBROUTINE TO POSITION CURSOR AT           *
                ;*    LINE L (0=TOP), COLUMN H (0=LEFT)          *
                ;*                                               *
                ;*************************************************
                        ;
                        ;
0196                    ORG     CLRSCRN+16H             ;FORCE TO CORRECT LOC.
                        ;
```

```
60 00036 006000          ROR     R0              ;ROLL IT OVER THERE
61 00040 103002          BCC     3$              ;BRANCH ON OVER IF NO CARRY IN
62 00042 062700          ADD     #100000,R0      ;ADD IT IN THERE
         100000
63 00046 105301 3$:      DECB    R1              ;DECREMENT THE COUNT
64 00050 001371          BNE     1$              ;AND LOOP FOR MORE
65 00052 000207 2$:      RTS     PC              ;BACK TO CALLER
66                ;
67                ;
68                ;SFTLFT - ROUTINE TO SHIFT A 16-BIT QUANTITY N BITS LEFT
69                ;CALL: VALUE=SFTLFT(QUAN16,NUMBIT)
70                ;ARGUMENTS:
71                ;   QUAN16 - THE INTEGER 16-BIT QUANTITY TO BE ROTATED.
72                ;   NUMBIT - NUMBER OF BITS TO ROTATE OVER.
73                ;
74       000054'          .CSECT
75                        .GLOBL  SFTLFT
76
77 00054 005725 SFTLFT:     TST     (R5)+           ;INCREMENT PAST ARGUMENT COUNT
78 00056 013500          MOV     @(R5)+,R0       ;GET THAT INTEGER VALUE
79 00060 013501          MOV     @(R5)+,R1       ;AND GET THE COUNTER OF BITS
80 00062 001403          BEQ     2$              ;RETURN IF ZERO, ALSO
81
82 00064 006300 1$:      ASL     R0              ;SHIFT IT OVER THERE
83 00066 105301          DECB    R1              ;DECREMENT THE COUNT
84 00070 001375          BNE     1$              ;AND LOOP FOR MORE
85 00072 000207 2$:      RTS     PC              ;BACK TO CALLER
86                ;
87                ;
88                ;SFTRGT - ROUTINE TO SHIFT A 16-BIT QUANTITY N BITS RIGHT
89                ;CALL: VALUE=SFTRGT(QUAN16,NUMBIT)
90                ;ARGUMENTS:
91                ;   QUAN16 - THE INTEGER 16-BIT QUANTITY TO BE ROTATED.
92                ;   NUMBIT - NUMBER OF BITS TO ROTATE OVER.
93                ;
94       000074'          .CSECT
95                        .GLOBL  SFTRGT
96
97 00074 005725 SFTRGT: TST     (R5)+           ;INCREMENT PAST ARGUMENT COUNT
98 00076 013500          MOV     @(R5)+,R0       ;GET THAT INTEGER VALUE
99 00100 013501          MOV     @(R5)+,R1       ;AND GET THE COUNTER OF BITS
100 0102 001405    BEQ     2$              ;RETURN IF ZERO, ALSO
101
102 0104 006200 1$:      ASR     R0              ;SHIFT IT OVER THERE
103 0106 042700          BIC     #100000,R0      ;CLEAR THE TOP UNCONDITIONALLY
         100000
104 0112 105301          DECB    R1              ;DECREMENT THE COUNT
105 0114 001373          BNE     1$              ;AND LOOP FOR MORE
106 0116 000207 2$:      RTS     PC              ;BACK TO CALLER
107               ;
108               ;
109               ;END OF LOGICF.MAC
110      000001'          .END
```

**end**

```
                TCURSOR:
0196 D5                 PUSH    D       ;SAVE DE PAIR
0197 1100F8             LXI     D,SCRN  ;VALUE OF VIDEO DISPLAY
019A D5                 PUSH    D       ;ALSO SAVE IT FOR LATER
                        ;
019B E5                 PUSH    H       ;SAVE INPUT
019C 2AFBED             LHLD    CURS    ;GET OLD POS
019F 19                 DAD     D
01A0 3AFEED             LDA     STR     ;GET CHAR UNDER CURSOR
01A3 77                 MOV     M,A     ;UNSAVE IT
                        ;
                        ;
01A4 E1                 POP     H       ;UNSAVE INPUT
01A5 5C                 MOV     E,H     ;SAVE X VALUE
01A6 2600               MVI     H,0     ;ZERO HIGH ORDER
01A8 54                 MOV     D,H     ;MAKE X INTO 16 BIT QUANT
                        ;
01A9 29                 DAD     H       ;MULT BY 64
01AA 29                 DAD     H
01AB 29                 DAD     H
01AC 29                 DAD     H
01AD 29                 DAD     H
01AE 29                 DAD     H
01AF 19                 DAD     D       ;PLUS X
01B0 22FBED             SHLD    CURS    ;SAVE IT
01B3 C3C401             JMP     CONTIN  ;CONTINUE THERE...
                        ;

                ;*****************************************************
                ;*                                                   *
                ;*      COMPAL-80 1.5 MONITOR EQUATES                *
                ;*                                                   *
                ;*****************************************************
                        ;
                        ;
EDFE =          STR     EQU     0EDFEH  ;CHAR UNDER CURSOR
EDFB =          CURS    EQU     0EDFBH  ;LOCATION OF RELATIVE CURSOR.
F800 =          SCRN    EQU     0F800H  ;BEGINNING ADDRESS OF VTI
0006 =          CHCLS   EQU     6       ;CLEAR SCREEN CODE
0004 =          CHEOL   EQU     4       ;SELECTED DEOL CHAR
                        ;
                        ;
                        ;
                        ;
                        ;
```

*(Continued on next page)*

# Using WORD-MASTER

```
                ;*********************************************************
                ;*                                                       *
                ;* PBEGMEM POINTS TO BEGINNING OF MEMORY TO USE          *
                ;* FOR EDIT BUFFER AND SCRATCHPAD. IF SPACE IS NEEDED    *
                ;* FOR PATCHES, PUT THEM WHERE THIS POINTS AND           *
                ;* INCREASE THIS POINTER. REMEMBER TO USE A LARGE        *
                ;* ENOUGH "SAVE" COMMAND.                                *
                ;*                                                       *
                ;*********************************************************
                        ;
                        ;
01B7                    ORG     CLRSCRN+37H     ;FORCE TO CORRECT LOC.
                        ;
                        ;
01B7 B829       PBEGMEM: DW MEMORY



                ;*********************************************************
                ;*                                                       *
                ;*      SIZE: TAKEN FROM THE FOLLOWING.                  *
                ;*                                                       *
                ;*********************************************************
                        ;
01B9 10         HITE:   DB 16   ;MUST BE EXACT SCREEN HEIGHT IN LINES
01BA 40         WID:    DB 64   ;MUST BE <= EXACT SCREEN WIDTH


                ;*********************************************************
                ;*                                                       *
                ;* EREOL CONTAINS THE CHARACTER(S) TO ERASE SCREEN       *
                ;* TO END-OF-LINE WITHOUT MOVING CURSOR, IF SUCH A       *
                ;* CHARACTER IS AVAILABLE IN THE TERMINAL HARDWARE.      *
                ;* IF 0, WILL BE SIMULATED BY SOFTWARE.                  *
                ;* AUTOMATICALLY SET TO CTL-U WHEN VIO IS IN USE.        *
                ;*                                                       *
                ;*********************************************************
                        ;
01BB 04         EREOL:  DB      CHEOL   ;(FIRST) CHARACTER, OR 0 IF NONE
01BC 00                 DB      0       ;SECOND CHARACTER IF TERMINAL
                                        ;...REQUIRES 2-CHARACTER
                                        ;...SEQUENCE, ELSE A 0.
```

```
0006 CHCLS      0004 CHEOL      0180 CLRSCRN    01C4 CONTIN     EDFB CURS
01C1 DELCLR     01C2 DELCUS     01C3 DELERE     01BB EREOL      01B9 HITE
29B8 MEMORY     01BD NOVIO      01EF OUTCHR     01B7 PBEGMEM    F800 SCRN
EDFE STR        0196 TCURSOR    01BA WID
```

**end**

## Listing

```
                Listing II.


comout: mov     a,c     ; get char in acc.
        cpi     cheol   ; delete to end of line?
        jz      deol    ; do delete
        ;
        jmp     crt     ; otherwise output value
        ;
        ; perform delete to end of line
        ;
deol:   lhld    curs    ; get cursor address
        lxi     d,scrn  ; address of screen
        dad     d       ; make absolute address
        push    h       ; save extra copy
        mov     a,l     ; get low order
        ani     63      ; get position on current line
        mov     b,a     ; save in b
        mvi     a,64    ; max. value is 64
```

```
                ;*********************************************************
                ;*                                                       *
                ;* NOVIO: IF NON-0, WILL NOT LOOK FOR IMSAI VIO.         *
                ;*                                                       *
                ;*********************************************************
                        ;
01BD FF         NOVIO:  DB 0FFH         ;PATCHED FOR COMPAL-80
01BE 000000             DB 0,0,0        ;RESERVED FOR EXPANSION



                ;*********************************************************
                ;*                                                       *
                ;* DELAYS EXECUTED AFTER VARIOUS TERMINAL FUNCTIONS,     *
                ;* BEFORE NEXT CHARACTER IS SENT TO TERMINAL. THESE      *
                ;* ALLOW TIME FOR TERMINAL TO RESPOND, AS REQUIRED       *
                ;* BY SOME TERMINALS WHEN USED AT HIGH BAUD RATES.       *
                ;* INCREASE IF YOU EXPERIENCE, FOR EXAMPLE, LOSS OF      *
                ;* CHARACTERS AFTER CLEAR SCREEN.  EACH DELAY IS         *
                ;* APPROX NUMBER OF MILLISECONDS ON 4MHZ PROCESSOR;      *
                ;* DELAY IS TWICE AS LONG AS SHOWN FOR 2MHZ 8080.        *
                ;*                                                       *
                ;*********************************************************
                        ;
01C1 19         DELCLR: DB 25   ;DELAY AFTER CLEAR SCREEN: 25+ MSEC.
01C2 0A         DELCUS: DB 10   ;DELAY AFTER POSITION CURSOR: 10+MSEC.
01C3 05         DELERE: DB 5    ;DELAY AFTER ERASE TO EOL: 5+ MSEC.

                        ;
                        ; CONTINUATION OF ADDRESSING PATCH FOR TCURSOR:
                        ;
                        ;
                CONTIN:
01C4 D1                 POP     D       ;GET DE=SCRN;
01C5 19                 DAD     D       ;MAKE ABS ADDR.
01C6 7E                 MOV     A,M     ;GET CHAR UNDER CURSOR
01C7 32FEED             STA     STR     ;SAVE IT
01CA 36FF               MVI     M,255   ;PLACE CURSOR ON DISP
01CC D1                 POP     D       ;UNSAVE DE PAIR.
01CD C9                 RET             ;AND EXIT...
                        ;

01CE                    END
```

*(continued on the top of page 28)*

```
                sub     b       ; get actual count
                mvi     b,space ; fill character
deol1:          mov     m,b     ; fill position
                inx     h       ; point to next one
                dcr     a       ; decrement count
                jnz     deol1   ; and repeat til done
                pop     h       ; get original position
                mvi     m,255   ; save a cursor char there
                mvi     a,space ; char under cursor now a space
                sta     str     ; save it too
                ret             ; and exit
                ;
                ;
                ;
space           equ     0A0H    ; 32 w/ parity added
scrn            equ     0f800h  ; start of VTI
curs            equ     0edfbh  ; loc. of relative cursor
str             equ     0edfeh  ; char under cursor
cheol           equ     4       ; selected deol char
crt             equ     0e0c2h  ; output routine in 1.5 monitor
```

**end**

# ERGO

## Listing

```
0 ! /ERGO3/ (RUBIN)
100 LOGICAL Q,Q0,Q1,Q2,P0
120 REAL R
140 INTEGER A,D0,V,I,I0,J,J0,J1,J2,J3,K,L,L0,L1,L2,L3,W0,W1,W2,
    Y0,Y1,Z1,Z2,Z(2)
160 STRING D,P,T,X,X0,X1,X2,X3,X4,W(50),A(0:50),A(0:50),N$
200 OPEN /GRAM/,INPUT,1
210 INPUT FROM 1: Z1,Z2
220 STRING S(2,5*(Z1+Z2),2)
230 INPUT FROM 1: S(1,J,1),S(1,J,2) FOR J=1 TO Z1
240 INPUT FROM 1: S(2,J,1),S(2,J,2) FOR J=1 TO Z2
250 INPUT FROM 1: NO
260 STRING N(2*NO)
270 INPUT FROM 1: N(J) FOR J=1 TO 2*NO
290 CLOSE 1
300 Z(1)=Z1, Z(2)=Z2
310 R=RND(-1), L1,Q0=0
320 P=SPACE(1)
340 GOSUB 5000
400 PRINT "WHAT IS YOUR NAME":
410 INPUT N$
420 I=INDEX(N$,P)
430 N$=LEFT(N$,I-1) IF I>2
500 ! QUESTION FOR THE STUDENT
510 GOTO 700 IF L1<3 OR Q0=0
520 R=RND
530 GOTO 700 IF R>.3
540 R=INT(R*100), R=R MOD L1 +1
550 PRINT
560 PRINT "WHAT ARE "+A(R)+"?"
570 INPUT X3
572 Q=0, I=INDEX(X3,P), D=LEFT(X3,I-1)
574 X3=SUBSTR(X3,I+1), Q=1 IF D="IT" OR D="THEY" OR D="IS" OR
    D="ARE"
576 GOTO 572 IF Q
580 X1=A(R), I0=1, P0=0
590 GOSUB 3910
600 GOTO 1200
700 PRINT
710 PRINT "GO AHEAD, "+N$
720 INPUT X0
730 PRINT
740 IF X0="NO" THEN GOSUB 6400
1860 NEXT K
1870 L1=L1+1, A(L1)=S(1,J,F0) UNLESS Q
1880 NEXT J
1890 IF L3 <> -1, THEN L2=L0-L3, L0=L3
1900 GOTO 2700 IF L1=0
2000 GOTO 2200 IF L0=0
2010 PRINT X44" ":
2020 PRINT "ARE": IF I0=1
2030 GOTO 2070 IF I0=1
2040 PRINT "HAS": IF P0=0 AND W1=1
2050 PRINT "HAVE": IF P0=1 AND W1=1
2060 PRINT "IS A PROPERTY OF": IF W1=0
2070 PRINT " ":
2100 FOR J=1 TO L0
2110 PRINT IF POS>71-LENGTH(A(J))
2120 PRINT A(J)+", ":
2130 NEXT J
2140 PRINT "."
2150 PRINT
2160 GOTO 500 IF L0=L1
2200 GOTO 2300 IF L2=0
2210 PRINT "SOME "+X4+" ARE ":
2220 FOR J = L0+1 TO L0+L2
2230 PRINT IF POS>71-LENGTH(A(J))
2240 PRINT A(J)+", ":
2250 NEXT J
2260 PRINT "."
2270 PRINT
2280 GOTO 500 IF L0+ L2=L1
2300 PRINT "SOME "+X4+" MAY ":
2310 PRINT "BE ": IF I0=1
2320 GOTO 3250 IF I0=1
2330 PRINT "HAVE ": IF W1=1
2340 PRINT "BE A PROPERTY OF ": IF W1=0
2350 FOR J = L0+L2+1 TO L1
2360 PRINT IF POS>71-LENGTH(A(J))
2370 PRINT A(J)+", ":
2380 NEXT J
2390 PRINT "."
2400 GOTO 500
2700 PRINT "I DON'T KNOW ABOUT THAT. DID YOU SPELL IT
    RIGHT?"
```

```
750 IF X0="PAUSE" THEN PAUSE
760 GOSUB 3000
780 IF W1=2 THEN 6100
790 IF V=0 OR V=1 AND A<3 THEN 2800
800 IF W(1)="WHAT" THEN 1600
1200 ! RESPONSE TO SIMPLE QUESTION
1210 Q1,Q2=0
1220 FOR J=1 TO Z(I0)
1230 Q1 = (X2=S(I0,J,1) AND X4=S(I0,J,2)
1240 Q2 = (X2=S(I0,J,2) AND X4=S(I0,J,1)
1250 J=Z(I0) IF Q1 OR Q2
1260 NEXT J
1270 GOTO 1420 IF Q1
1280 GOTO 1440 IF Q2

1400 PRINT "NOT THAT I KNOW"
1410 GOTO 500
1420 PRINT "YES, THAT'S RIGHT"
1430 GOTO 500
1440 PRINT "SOME "+X2+" ARE "+X4+"." IF I0=1
1450 PRINT "NO, "+X4+" HAVE "+X2+"." IF I0=2
1460 GOTO 500
1600 ! RESPONSE TO 'WHAT'
1610 L0,L1,L2=0, Q0=1, L3=-1, A(0)=X4
1620 IF W0=2 THEN F0=2,F1=1 ELSE F0=1, F1=2
1640 ! SIMPLE MATCHING
1650 L1=L1+1, A(L1)=S(I0,J,F1) IF X4=S(I0,J,F0) FOR J=1 TO Z(I0)
1660 L0=L1
1680 ! MAYBE
1700 IF W0 <> 0 THEN 1750 ELSE W0=2,L3=L0
1710 GO TO 1620
1750 FOR J=1 TO Z(1)
1770 FOR L=1 TO L0
1780 Q = A(L)=S(1,J,F1)
1790 L=L0 IF Q
1800 NEXT L
1810 GOTO 1880 UNLESS Q
1830 FOR K=0 TO L1
1840 Q = S(1,J,F0)=A(K)
1850 K=L1 IF Q
```

*(Continued on the top of page 30)*

```
2710 GOTO 500
2800 PRINT "I DON'T UNDERSTAND;":
2810 IF V=0 THEN PRINT "NEEDS A VERB." ELSE PRINT "SUBJECT
     AND OBJECT MUST BE SEPARATED."
2812 PRINT "BY VERB OR ARTICLE 'A' OR 'AN'." IF V
2820 GOTO 500
3000 ! SENTENCE ANALYSIS
3010 J=1, L=LENGTH(X0), X=X0
3020 I=INDEX(X,P)
3030 GOTO 3060 IF I=0
3040 L=L-I, W(J)=LEFT(X,I-1), X=RIGHT(X,L), J=J+1
3050 GOTO 3020
3060 T=RIGHT(X,1), J0=J
3070 IF T="." OR T="?" OR T="!" OR T="/" THEN W(J)=
     LEFT(X,L-1) ELSE W(J)=X
3100 D0,Y0,Y1,A,V=0 , X1,X3=""
3120 FOR J = 1 TO J0
3130 A=J IF W(J)="A" OR W(J)="AN"
3140 V=J IF W(J)="IS" OR W(J)="ARE" OR W(J)="HAS" OR
     W(J)="HAVE"
3150 D0=J IF W(J)="DO" OR W(J)="DOES"
3160 Y0=J IF W(J)="YOU"
3170 Y1=J IF W(J)="KNOW" OR W(J)="TELL"
3200 NEXT J
3210 IF Y0 AND Y1-Y0=1 THEN 6000
3220 IF V=0 OR V=1 AND A<3 THEN RETURN
3230 I0=1 IF W(V)="IS" OR W(V)="ARE"
3240 I0=2 IF W(V)="HAS" OR W(V)="HAVE"
3250 P0=W(V)="ARE" OR W(V)="HAVE"
3260 P0=0 IF W(D0)="DOES" IF D0
3300 IF I0=1 THEN W0=0 ELSE IF V=2 THEN W0=2 ELSE W0=1
3310 IF W(1)="WHAT" AND D0=2 THEN 3700 ELSE W1=0
3330 IF V>1 THEN 3500
3400 J1=2, J2=A-1, J3=A
3410 GOTO 3600
3500 J1=1, J2=V-1, J3=V+1
3510 IF D0=1 THEN J1=2
3600 FOR J = J1 TO J2
3610 X1 = X1+W(J)
3620 X1 = X1+P UNLESS J=J2
```

*(Continued on next page)*

# ERGO

```
3630 NEXT J
3640 FOR J = J3 TO J0
3650 X3 = X3+W(J)
3660 X3 = X3+P UNLESS J=J0
3670 NEXT J
3680 GOTO 3910
3700 ! SPECIAL RULE FOR VERB AT END
3710 W1=1
3720 FOR J=3 TO J0-1
3730 X3=X3+W(J)
3740 X3=X3+P UNLESS J=J0-1
3750 NEXT J
3900 GOTO 3940 IF W1
3910 X=X1
3920 GOSUB 4000
3930 X2=X
3940 X=X3
3950 GOSUB 4000
3960 X4=X
3970 RETURN
4000 ! TEST FOR SINGULAR NOUN & DROP ARTICLE
4010 Q=0
4020 X=SUBSTR(X,3), Q=1 IF LEFT(X,2)="A "
4030 X=SUBSTR(X,4), Q=1 IF LEFT(X,3)="AN "
4040 I=INDEX(X,P), D=LEFT(X,I-1)
4050 X=SUBSTR(X,I+1), Q=0 IF D="ALL" OR D="ANY" OR
     D="SOME" OR D="FEW"
4090 RETURN UNLESS Q
4100 ! CHANGE SINGULAR TO PLURAL NOUNS
4120 FOR J=1 TO 2*N0-1 BY 2
4130 Q = X=N(J)
4140 J=2*N0-1 IF Q
4150 NEXT J
4160 X=N(J+1) IF Q
4170 RETURN IF Q
4200 L=LENGTH(X), D=RIGHT(X,2), I=1
4210 I=2 IF D="CH" OR D="SH"
4220 D=RIGHT(X,1)
4230 I=2 IF D="S" OR D="X" OR D="Z"
4240 I=3 IF D="Y"
4250 ON I GOTO 4260,4280,4300
4260 X=X+"S"
4270 RETURN
```

```
6010 W1=2
6020 FOR J=Y1+1 TO J0
6030 X3=X3+W(J)
6040 X3=X3+P UNLESS J=J0
6050 NEXT J
6060 Q=0, I=INDEX(X3,P), D=LEFT(X3,I-1)
6070 X3=SUBSTR(X3,I+1), Q=1 IF D="ME" OR D="ANYTHING" OR
     D="SOMETHING" OR D="ABOUT"
6080 IF Q THEN 6060 ELSE 3940
6100 Q=1
6110 FOR I0=1,2
6120 FOR J=1 TO Z(I0)
6130 FOR I=1,2
6140 GOTO 6220 IF X4<>S(I0,J,I)
6150 PRINT "HERE IS A LIST OF SOME THINGS I KNOW ABOUT "
     +X3+":" IF Q
6160 Q=0
6170 PRINT X4:
6180 PRINT " ARE " +S(I0,J,2) IF I=1 AND I0=1
6190 PRINT " MAY BE "+S(I0,J,1) IF I=2 AND I0=1
6200 PRINT " HAVE "+S(I0,J,2) IF I=1 AND I0=2
6210 PRINT " ARE PART OF "+S(I0,J,1) IF I=2 AND I0=2
6220 NEXT I
6230 NEXT J,I0
6250 PRINT "SORRY, I COULDN'T FIND ANYTHING ON THAT. ASK
     YOUR TEACHER." IF Q
6300 GOTO 500
6400 PRINT "ANYONE ELSE":
6410 INPUT D
6420 PRINT
6430 IF D="NO" THEN STOP ELSE 400
10000 PRINT
10010 PRINT "HELLO. I CAN THINK ABOUT ANYTHING YOU TELL
     ME, AND ANSWER QUESTIONS."
10020 PRINT "I ONLY UNDERSTAND VERY SIMPLE ENGLISH:"
10030 PRINT"  SIMPLE DECLARATIVE STATEMENTS USING THE
     VERB 'IS' OR 'HAS.' "
10040 PRINT"                          AN OBJECT IS A SET. A SET
     HAS PARTS."
10050 PRINT"  SIMPLE QUESTIONS:  IS AN OBJECT A SET? DO SETS
     HAVE PARTS?"
10060 PRINT"                          AN OBJECT IS A SET? A SET
     HAS PARTS?"
```

```
4280 X=X+"ES"
4290 RETURN
4300 X=LEFT(X,L-1), D=RIGHT(X,1)
4310 IF D="A" OR D="E" OR D="I" OR D="O" OR
     RIGHT(X,2)#"QU" THEN X=X+"YS" ELSE X=X+"IES"
4350 RETURN
5000 ! DEDUCE IMPLICATIONS  AND ADD TO S-LIST
5010 FOR I0=1,2
5020 FOR J=1 UNTIL J>Z(I0)
5030 FOR K=1 TO J-1, J+1 UNTIL K>Z(I0)
5040 GOTO 5070 UNLESS S(I0,J,2)=S(I0,K,1)
5052 FOR L=1 TO Z(I0)
5054 Q = (S(I0,J,1)=S(I0,L,1) AND S(I0,K,2)=S(I0,L,2))
5056 L=Z(I0) IF Q
5058 NEXT L
5060 Z(I0)=Z(I0)+1, S(I0,Z(I0),1)=S(I0,J,1), S(I0,Z(I0),2)=S(I0,K,2)
     UNLESS Q
5070 NEXT K
5080 NEXT J,I0
5100 FOR J=1 TO Z(1)
5110 FOR K=1 UNTIL K>Z(2)
5120 GOTO 5150 UNLESS S(1,J,2)=S(2,K,1)
5132 FOR L=1 TO Z(2)
5134 Q = (S(1,J,1)=S(2,L,1) AND S(2,K,2)=S(2,L,2))
5136 L=Z(2) IF Q
5138 NEXT L
5140 Z(2)=Z(2)+1, S(2,Z(2),1)=S(1,J,1), S(2,Z(2),2)=S(2,K,2) UNLESS Q
5150 NEXT K
5160 FOR K=1 UNTIL K>Z(2)
5170 GOTO 5200 UNLESS S(1,J,1)=S(2,K,2)
5182 FOR L=1 TO Z(2)
5184 Q = (S(1,J,2)=S(2,L,2) AND S(2,K,1)=S(2,L,1))
5186 L=Z(2) IF Q
5188 NEXT L
5190 Z(2)=Z(2)+1, S(2,Z(2),2)=S(1,J,2), S(2,Z(2),1)=S(2,K,1) UNLESS Q
5200 NEXT K
5210 NEXT J
5300 RETURN
6000 ! INQUIRY FOR AVAIL INFO.
```

*(Continued on the top of page 32)*

```
10070 PRINT"  GENERAL QUESTIONS:  WHAT IS AN OBJECT?
      WHAT HAS A PART?"
10080 PRINT"                      WHAT DOES A SET HAVE?"
10090 PRINT"                      WHAT DO YOU KNOW
      ABOUT SETS?"
10100 PRINT"                      CAN YOU TELL ME
      ANYTHING ABOUT SETS?"
10110 PRINT
10120 PRINT "YOU ASK ME QUESTIONS ABOUT THE SUBJECT, AND
      SOMETIMES I'LL ASK YOU A''
10122 PRINT"QUESTION. YOU ANSWER WITH JUST A SIMPLE WORD
      OR PHRASE, NOT A SENTENCE."
10130 PRINT
10150 PRINT "THE ONLY DATA FILE I HAVE SO FAR IS ON
      ELEMENTARY ENGLISH GRAMMAR."
10160 PRINT
10161 PRINT
11000 GOTO 100
GOTO 10000
```

**end**

# Computer Based Reading

## Listing

```
15 REM WORD-AT-A-TIME FAST READING PROGRAM
20 REM 12 NOV 80  D.CAULKINS
22 REM
23 REM SET UP VARIABLES FOR RAPID ACCESS
24 E$="#E#":C$="A":DIMW$(255):M$="
26 G=9:D=1:01=1:02=2:06=6:T0=20:F0=50:07=7
27 V$="":REM 10 MOVE-CURSOR-DOWN CHARACTERS
28 H$="":REM CURSOR-TO-HOME CHARACTER

30 P$="                    ":K$=""
40 B$="":REM 20 MOVE-CURSOR-LEFT CHARACTERS
80 GOTO360:REM SKIP OVER FAST ROUTINES TO SETUP AND USER COMMAND INPUT
85 REM
90 REM WORD DISPLAY ROUTINE
100 REM PUT CLOSE TO START OF CODE SO IT WILL BE FAST
146 PRINTH$:PRINTV$;

147 FORI=01TOF0
148 IT=TI:REM PUT CURRENT 1/60 SEC COUNTER VALUE IN IT
149 REM GET NEXT WORD FROM WORD ARRAY
150 C$=W$(I)
160 IFC$=""GOTO1800
165 IFC$=E$GOTO2600:REM IF END-OF-TEXT MARK,EXIT
167 W=LEN(C$)

170 IFASC(C$)=06GOTO300:REM IF NEXT ARRAY CELL EMPTY,READ MORE FROM DISK
180 REM CALCULATE VARIABLES TO DISPLAY WORD CENTERED IN VISUAL FIELD
203 L=INT(W/02):L=T0-L
216 PRINTLEFT$(M$,L);C$;LEFT$(M$,G)
217 PRINT"";:GOSUB224:G=W:NEXTI
219 REM SHOULD NEVER EXIT FROM FOR-NEXT HERE
220 $C:END:REM CLOSE DISK FILE AND RETURN TO BASIC.

222 REM WORD DISPLAY SPEED SUBROUTINE
224 GETK$:IFK$=""GOTO228:REM IF NO NEW KEYBOARD INPUT USE OLD DELAY VALUE
225 REM
226 D=07+VAL(K$)
228 IFTI<(IT+D)GOTO228:REM WAIT UNTIL TIME FOR NEXT WORD
230 RETURN
250 REM THE ROUTINE BELOW GETS STRINGS FROM THE DISK FILE
254 REM AND PUTS EACH SUCCESSIVE WORD FROM EACH STRING INTO A CELL IN THE

448 GOSUB3000:REM CLEAR SCREEN AND SET UP CENTER OF VISUAL FIELD MARKS
450 SYS(45095):GOTO146
480 REM DISK FILE WRITE ROUTINE
482 PRINT"ENTER TEXT TO BE READ; END EACH LINE WITH (CR)."
484 PRINT"USE UPPER CASE ONLY."
486 PRINT"TYPE A SPACE AND CARRIAGE RETURN AFTER THE LAST WORD IN THE TEXT,
488 PRINT"THEN TYPE #E# (CR) TO TERMINATE TEXT ENTRY."

500 I$="SEP80"
510 $O,1,"W","TXT1",I$:REM OPEN DISK FILE FOR WRITING
520 SYS(45095):GOSUB600:REM TURN OFF DISK DRIVE;GET TEXT INPUT FROM USER
525 SYS(45092):REM TURN ON DISK DRIVE MOTOR
530 $W,R$:REM WRITE TEXT INPUT TO DISK
535 IFR$="#E#"GOTO550:REM IF END-OF-TEXT MARK CLOSE FILE,
536 REM:TO USER COMMAND INPUT

537 R$=""
540 GOTO520:REM BACK FOR MORE USER TEXT INPUT
550 $C
555 R$=""
560 RUN
600 GETK$:IFK$=""THEN600
605 PRINTK$;
610 IFASC(K$)<>13THENR$=R$+K$:GOTO600

630 RETURN
1800 SYS(45095)
1820 $C
1830 END
1990 REM DISK READ SUBROUTINE
2010 SYS(45092):REM TURN ON DISK DRIVE MOTOR
2100 $R,R$:REM READ NEXT STRING FROM DISK

2300 IFR$=""GOTO2600
2400 RETURN
2500 GOTO2100
2600 $C
2610 PRINT:PRINT:PRINT:PRINT
2700 PRINT"END OF DISK FILE"
2710 RUN
```

```
258 REM STRING ARRAY W$
300 GOSUB1990:REM GET STRING FROM DISK
304 J=1:E=0
308 FORI=JTOLEN(R$)
312 C$=MID$(R$,I,1)
316 IFC$<>" "GOTO332
320 K=K+1

324 W$(K)=MID$(R$,J ,(I-J))
326 IFW$(K)="*E*"THENE=1
328 J=I+1
330 GOTO308
332 NEXTI
336 K=K+1

340 W$(K)=MID$(R$,J ,(I-J))
342 IFW$(K)="*E*"THENE=1
344 IFE<>1GOTO300
346 SYS(45095):GOTO146:REM SYS(45095) TURNS OFF DISK DRIVE MOTOR
360 REM START SET UP AND USER COMMAND INPUT
368 REM FILL ARRAY W$ WITH NON-PRINTING CHARACTERS

370 FORI=1TO255:W$(I)=CHR$(6):NEXTI
372 K=0
373 $C:REM $C CLOSES DISK FILES,IN CASE ANY ARE OPEN
374 $O,1,"R","TXT1",I$:REM OPEN DISK FILE
376 SYS(45095):REM TURN OFF DISK DRIVE MOTOR;OPEN TURNED IT ON
412 PRINT"R(EAD) OR W(RITE) ?"
414 GETC$:IFC$=""GOTO414

416 PRINTC$
420 IFC$="W"GOTO480
422 REM INITIAL WORD DISPLAY SPEED SETTING
423 PRINT"SPEED ? (1=FAST,9=SLOW)"
424 PRINT" RATES ARE AS FOLLOWS:"
425 PRINT" 1 - 450 WORDS PER MINUTE (WPM)"
426 PRINT" 2 - 400 WPM"

427 PRINT" 3 - 360 WPM"
428 PRINT" 4 - 327 WPM"
429 PRINT" 5 - 300 WPM"
430 PRINT" 6 - 277 WPM"
431 PRINT" 7 - 257 WPM"
432 PRINT" 8 - 240 WPM"
433 PRINT" 9 - 225 WPM"
440 INPUTD:D=D+07
```

*(Continued on the top of page 34)*

```
3000 REM DISPLAY CENTER OF VISUAL FIELD MARKS
3010 PRINT"":REM CLEAR SCREEN
3020 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
3030 PRINT"                    ";"*"
3032 PRINT
3035 PRINT"                    ";"*"
3040 IT=TI+300
3050 IFTI<ITGOTO3050
3060 RETURN
READY.
```

**end**

# Small-C
## Listing - Relocatable Machine Code

```
; Small-C Compiler Rev 1.1P
; £INCLUDE GLBDEF
; /*      DEFINE SYSTEM DEPENDENT PARAMETERS.                    */
; MAIN ()
MAIN:
;       {
;       GLBPTR = STARTGLB;      /* CLEAR GLOBAL SYMBOLS.          */
           LD    HL,SYMTAB
           LD    (GLBPTR),HL
;       LOCPTR = STARTLOC;      /* CLEAR LOGICAL SYMBOLS.         */
           LD    HL,SYMTAB
           PUSH  HL
           LD    HL,300
           PUSH  HL
           LD    HL,14
           POP   DE
           CALL  CCMULT
           POP   DE
           ADD   HL,DE
           PUSH  HL
           LD    HL,14
           POP   DE
           ADD   HL,DE
           LD    (LOCPTR),HL
;       WQPTR = WQ;             /* CLEAR WHILE QUEUE.             */
           LD    HL,WQ
           LD    (WQPTR),HL



;       MACPTR =                /* CLEAR THE MACRO POOL.          */
;       LITPTR =                /* CLEAR THE LITERAL POOL.        */
;       SP =                    /* STACK POINTER (RELATIVE).      */
;       ERRCNT =                /* NO ERRORS.                     */
;       EOF =                   /* NOT YET EOF.                   */
;       INPUT =                 /* NO INPUT FILE.                 */
;       INPUT2 =                /* OR INCLUDE FILE.               */
;       OUTPUT =                /* NO OPEN UNITS.                 */
;       NCMP =                  /* NO OPEN COMPOUND STATEMENTS.   */
;       LASTST =                /* NO LAST STATEMENT YET.         */
;       QUOTE[1] =
;       0;                      /* ... ALL SET TO ZERO ...        */
;       QUOTE[0] = '"';         /* FAKE A QUOTED LITERAL.         */
;       CMODE = 1;              /* ENABLE PROCESSING.             */
;       ASK ();                 /* GET USER OPTIONS.              */
;       OPENOUT ();             /* GET AN OUTPUT FILE.            */
;       OPENIN ();              /* AND INITIAL INPUT FILE.        */
;       HEADER ();              /* INTRO CODE.                    */
;       PARSE ();               /* PROCESS *ALL* INPUT.           */
;       DUMPLITS ();            /* THEN DUMP LITERAL POOL.        */
;       DUMPGLBS ();            /* AND ALL STATIC MEMORY.         */
;       ERRORSUMMARY ();        /* SUMMARISE ERRORS.              */
;       TRAILER ();             /* FOLLOW UP CODE.                */
;       CLOSEOUT ();            /* CLOSE THE OUTPUT (IF ANY).     */
;       RETURN;                 /* THEN EXIT TO SYSTEM.           */
;       }
; PARSE ()


PARSE:
;       {
;       WHILE (EOF == 0)    /* DO UNTIL NO MORE INPUT.            */
CC2:
;               {
;               IF (AMATCH("CHAR",4)) {DECLGLB (CCHAR); NS ();}
;               ELSE IF (AMATCH("INT",3)) {DECLGLB (CINT); NS ();}
;
CC4:
;               ELSE IF (MATCH("£ASM")) DOASM ();
;
CC6:
;               ELSE IF (MATCH("£INCLUDE")) DOINCLUDE ();
;
CC8:
;               ELSE IF (MATCH("£DEFINE")) ADDMAC ();
;
CC10:
;               ELSE NEWFUNC ();
;
CC12:
CC13:
CC11:
CC9:
```

```
; 0 ERRORS IN COMPILATION.
; (0000)      21 02 02 $        22 B2 15 $        21 02 02 $        D5
; (000A)      21 2C 01          D5       21 0E 00          D1       CD 00 00 CCMULT
; (0015)      D1       19       D5       21 0E 00          D1       19
; (001D)      22 B4 15 $        21 B6 15 $        22 7E 16 $        21 FC 22 $
; (0029)      D5       21 01 00          D1       19       D5       21 00 00
; (0033)      D1       CD 00 00 CCPCHAR           22 FA 22 $        22 E8 22 $
; (003D)      22 F0 22 $        22 F2 22 $        22 EE 22 $        22 EC 22 $
; (0049)      22 EA 22 $        22 E4 22 $        22 50 1E $        22 3A 22 $
; (0055)      21 FC 22 $        D5       21 00 00          D1       19
; (005E)      D5       21 22 00          D1       CD 00 00 CCPCHAR
; (0066)      21 01 00          22 F8 22 $        CD 00 00 ASK
; (006F)      CD 00 00 OPENOUT            CD 00 00 OPENIN           CD 00 00 HEADER
; (0078)      CD 8B 00 $        CD 1E 01 $        CD 00 00 DUMPGLBS
; (0081)      CD 00 00 ERRORSUM           CD 00 00 TRAILER          CD 00 00 CLOSEOUT
; (008A)      C9       2A EC 22 $        D5       21 00 00          D1
; (0093)      CD 00 00 CCEQ               7C       B5       CA 1D 01 $
; (009B)      21 E3 01 $        D5       21 04 00          D5       CD 00 00 AMATCH
; (00A6)      C1       C1       7C       B5       CA BB 00 $        21 01 00
; (00B0)      D5       CD 00 00 DECLGLB            C1       CD 00 00 NS
; (00B8)      C3 17 01 $        21 E8 01 $        D5       21 03 00
; (00C2)      D5       CD 00 00 AMATCH            C1       C1       7C
; (00C9)      B5       CA DB 00 $        21 02 00          D5       CD 00 00 DECLGLB
; (00D4)      C1       CD 00 00 NS                C3 17 01 $        21 EC 01 $
; (00DE)      D5       CD 00 00 MATCH             C1       7C       B5
; (00E5)      CA EE 00 $        CD 00 00 DOASM             C3 17 01 $
; (00EE)      21 F1 01 $        D5       CD 00 00 MATCH             C1


; (00F6)      7C       B5       CA 01 01 $        CD 00 00 DOINCLUD
; (00FE)      C3 17 01 $        21 FA 01 $        D5       CD 00 00 MATCH
; (0108)      C1       7C       B5       CA 14 01 $        CD 00 00 ADDMAC
; (0111)      C3 17 01 $        CD 00 00 NEWFUNC           CD 00 00 BLANKS
; (011A)      C3 8B 00 $        C9       C5       C5       2A 50 1E $
; (0123)      D5       21 00 00          D1       CD 00 00 CCEQ
; (012B)      7C       B5       CA 33 01 $        C1       C1       C9
; (0133)      2A E2 22 $        D5       CD 00 00 PRINTLAB          C1
; (013B)      CD 00 00 COL                21 00 00          39       D5
; (0143)      21 00 00          D1       CD 00 00 CCPINT            21 00 00
; (014D)      39       CD 00 00 CCGINT            D5       2A 50 1E $
; (0155)      D1       CD 00 00 CCLT              7C       B5       CA E0 01 $
; (015E)      CD 00 00 DEFBYTE            21 02 00          39       D5
; (0166)      21 0A 00          D1       CD 00 00 CCPINT            21 02 00
; (0170)      39       D5       CD 00 00 CCGINT            2A       D1
; (0177)      CD 00 00 CCPINT             23       7C       B5       CA DD 01 $
; (0180)      21 80 16 $        D5       21 02 00          39       D5
; (0189)      CD 00 00 CCGINT             23       D1       CD 00 00 CCPINT
; (0191)      2A       D1       19       CD 00 00 CCGCHAR           D5
; (0198)      21 7F 00          D1       CD 00 00 CCAND             D5
; (01A0)      CD 00 00 OUTDEC             C1       21 02 00          39
; (01A8)      CD 00 00 CCGINT             D5       21 00 00          D1
; (01B0)      CD 00 00 CCEQ               D5       21 02 00          39
; (01B8)      CD 00 00 CCGINT             D5       2A 50 1E $        D1
; (01C0)      CD 00 00 CCGE               D1       CD 00 00 CCOR
; (01C7)      7C       B5       CA D2 01 $        CD 00 00 NL
; (01CF)      C3 DD 01 $        21 2C 00          D5       CD 00 00 OUTBYTE


; (01D9)      C1       C3 6D 01 $        C3 4A 01 $        C1       C1
; (01E2)      C9
; (01E3)      43  48  41  52  00  49  4E  54  00  23  41  53  4D  00  23  49
; (01F3)      4E  43  4C  55  44  45  00  23  44  45  46  49  4E  45  00
;       CC1  01E3     CC2  008B     CC3  011D     CC4  00BB     CC5  0117
;       CC6  00DB     CC7  0117     CC8  00EE     CC9  0117     CC10  0101
;       CC11  0117    CC12  0114    CC13  0117    CC14  0133    CC15  014A
;       CC16  01E0    CC17  016D    CC18  01DD    CC19  01D2
;       SYMTAB    VARIABLE  0202    GLBPTR    VARIABLE  15B2    LOCPTR    VARIABLE  15B4
;       WQ        VARIABLE  15B6    WQPTR     VARIABLE  167E    LITQ      VARIABLE  1680
;       LITPTR    VARIABLE  1E50    MACQ      VARIABLE  1E52    MACPTR    VARIABLE  223A
;       LINE      VARIABLE  223C    MLINE     VARIABLE  228C    LPTR      VARIABLE  22DC
;       MPTR      VARIABLE  22DE    NXTLAB    VARIABLE  22E0    LITLAB    VARIABLE  22E2
;       SP        VARIABLE  22E4    ARGSTK    VARIABLE  22E6    NCMP      VARIABLE  22E8
;       ERRCNT    VARIABLE  22EA    EOF       VARIABLE  22EC    INPUT     VARIABLE  22EE
;       OUTPUT    VARIABLE  22F0    INPUT2    VARIABLE  22F2    GLBFLG    VARIABLE  22F4
;       CTEXT     VARIABLE  22F6    CMODE     VARIABLE  22F8    LASTST    VARIABLE  22FA
```

```
CC7:
CC5:
;               BLANKS ();      /* FORCE EOF IF PENDING.                */
;               }
CC3:
;       }
; DUMPLITS ()
DUMPLITS:



;       {INT J, K;
;       IF (LITPTR == 0) RETURN;      /* IF NOTHING THERE, EXIT...      */
;       PRINTLABEL (LITLAB); COL (); /* PRINT LITERAL LABEL.            */
CC14:
;       K = 0;                        /* INIT AN INDEX.                 */
;       WHILE (K < LITPTR)            /*    TO LOOP WITH.               */
CC15:
;               {DEFBYTE ();          /* PSEUDO OP TO DEFINE BYTE.      */
;               J = 10;               /* MAX BYTES PER LINE/            */
;               WHILE (J--)
CC17:
;                       {OUTDEC ((LITQ[K++]&127));
;                       IF ((J==0) ¦ (K>=LITPTR))
;                               {NL ();/* NEED <CR>                    */
;                               BREAK;
;                               }
;                       OUTBYTE (','); /* SEPERATE BYTES.              */
CC19:
;                       }
CC18:
;               }
CC16:
;       }
CC1:        DB      67,72,65,82,0,73,78,84,0,35
SYMTAB:     DS      5040
GLBPTR:     DS      2
LOCPTR:     DS      2



WQ:         DS      200
WQPTR:      DS      2
LITQ:       DS      2000
LITPTR:     DS      2
MACQ:       DS      1000
MACPTR:     DS      2
LINE:       DS      80
MLINE:      DS      80
LPTR:       DS      2
MPTR:       DS      2
NXTLAB:     DS      2
LITLAB:     DS      2
SP:         DS      2
ARGSTK:     DS      2
NCMP:       DS      2
ERRCNT:     DS      2
EOF:        DS      2
INPUT:      DS      2
OUTPUT:     DS      2
INPUT2:     DS      2
GLBFLG:     DS      2
CTEXT:      DS      2
CMODE:      DS      2
LASTST:     DS      2
QUOTE:      DS      2
CPTR:       DS      2
IPTR:       DS      2
```

*(Continued on the top of page 36)*

```
;       QUOTE     VARIABLE  22FC      CPTR      VARIABLE  22FE      IPTR      VARIABLE  2300
;       MAIN      FUNCTION  0000      CCMULT    FUNCTION  UNDEF     CCPCHAR   FUNCTION  UNDEF
;       ASK       FUNCTION  UNDEF     OPENOUT   FUNCTION  UNDEF     OPENIN    FUNCTION  UNDEF
;       HEADER    FUNCTION  UNDEF     PARSE     FUNCTION  008B      DUMPLITS  FUNCTION  011E
;       DUMPGLBS  FUNCTION  UNDEF     ERRORSUM  FUNCTION  UNDEF     TRAILER   FUNCTION  UNDEF
;       CLOSEOUT  FUNCTION  UNDEF     CCEQ      FUNCTION  UNDEF     AMATCH    FUNCTION  UNDEF
;       DECLGLB   FUNCTION  UNDEF     NS        FUNCTION  UNDEF     MATCH     FUNCTION  UNDEF
;       DOASM     FUNCTION  UNDEF     DOINCLUD  FUNCTION  UNDEF     ADDMAC    FUNCTION  UNDEF
;       NEWFUNC   FUNCTION  UNDEF     BLANKS    FUNCTION  UNDEF     PRINTLAB  FUNCTION  UNDEF
;       COL       FUNCTION  UNDEF     CCPINT    FUNCTION  UNDEF     CCGINT    FUNCTION  UNDEF



;       CCLT      FUNCTION  UNDEF     DEFBYTE   FUNCTION  UNDEF     OUTDEC    FUNCTION  UNDEF
;       CCGCHAR   FUNCTION  UNDEF     CCAND     FUNCTION  UNDEF     CCGE      FUNCTION  UNDEF
;       CCOR      FUNCTION  UNDEF     NL        FUNCTION  UNDEF     OUTBYTE   FUNCTION  UNDEF
```

**end**

# Software Reviews

**CLOAD Magazine, Inc.**
Published by **CLOAD**
**Box 1267**
**Goleta, CA 93017**
Price: **$42/year**
Reviewed by **John Hatfield**

CLOAD Magazine is a monthly offering costing $42 per year. Sounds like a lot of money for a magazine, however what you get is a cassette tape with 4 to 6 pre-recorded, debugged and tested programs for your level I or level II TRS-80. Not too shabby for only $3.50 per cassette!

CLOAD's programs cover a fairly wide spectrum of interests from the latest way to do battle among the stars to more serious programs like mailing lists to extremely challenging games like Psycho Logic which are guaranteed to send you to the funny farm if you're not there already from playing the latest version of Adventure. (Anybody figured out how to get by the Iron Statue yet?)

Tape loading on the four samples were either excellent or horrible. The two that loaded without incident were fresh copies direct from CLOAD. The two that didn't load at all were opened and apparently used before I got them.

CLOAD's program index is shown on the face of the cassettes along with the tape counter locations for each of the two recordings of each program.

Along with the tape each month comes a one- to three-page blurb from the Publisher containing announcements, special instructions if required for the programs on tape, and other items of interest to the publisher and us semi-serious computernicks.

If you like a variety of different programs and are not in love with entering code using the hunt and peck technique reading code by microscope, the CLOAD Magazine should be on your list of things to try.

**MON 3**
Published by **Howe Software**
**14 Lexington Road**
**New City, NY 10956**
Price: **$39.95**
Reviewed by **Steven Wexler**

MON 3 is a powerful monitor designed for the TRS-80 Lv.2 with at least 16K memory. The program carries out many different tasks; it is in effect several programs inside of one program. I will discuss some of the more important functions below.

ASCII DUMP. This command displays the contents of each memory location in a specified block of memory in ASCII format. This allows strings to be distinguished from other surrounding data.

HEX DUMP. This command displays the contents of memory in hexadecimal form.

DISASSEMBLE. Shows the symbolic instructions in Zilog mnemonics that correspond to the object code already in memory. Optionally, the disassembled code can be dumped to cassette and loaded into Radio Shack's assembler!

HEX ARITHMETIC. Takes two inputs and prints them, their sum, and their differences in both hexadecimal and decimal form.

FIND BYTE. This command will search a block of memory for all locations that contain a specific byte.

FIND WORD. Searches a block of memory for all locations that are the start of two byte numbers.

EDIT MEMORY. Allows you to enter new values into memory in hexadecimal form.

MOVE BLOCK. With this command, blocks of memory may be moved from one area of memory to another.

RELOCATE OBJECT CODE. Using this command a program can be altered and moved to run in a different area of memory! Not all programs can be relocated. With Relocate, Mon 3 can be run anywhere in memory!

JUMP. Jump transfers control to any location. It is assumed that a legitimate program resides there.

LOAD CASSETTE. This command reads a tape written in SYSTEM format into memory. You do not need to know what the file name is. The name of the tape file and the starting, ending, and entry locations of the file are supplied by the program.

PUNCH OBJECT TAPE. This command will write onto cassette the object code of a program in memory.

UNLOAD OBJECT PROGRAM. Unload is used for saving object programs that reside below 6C00H onto disk. After moving the target program above 6C00H, a block transfer routine is added to the target program so that when you run the target program, it is moved to the proper location, and then executed.

These are most, but not all of the commands available in Mon 3. The output of commands may optionally be directed to the line printer.

A disk based upgrade has been released called Mon 4. I have found Mon 3 to be an extremely useful program and can strongly recommend it.

**DISK KEYPLUS**
Published by **SJW, Inc.**
**P.O. Box 438**
**Huntingdon Valley, PA 19006**
Price: **$19.95**
Reviewed by **Marcia Kalish**

Disk Keyplus is a very powerful, underpriced multi-purpose TRS-80 integrated utility package with excellent documentation – a must buy! A 16K cassette version, without the disk-based utilities is also available for $14.95.

Disk Keyplus is a powerful collection of utilities for the TRS-80. 32K and 48K versions are available that are designed to run under either TRSDOS or NEWDOS.

Keying <SHIFT-CLEAR> will enter Keyplus Command Mode. You may enter Command Mode anytime the TRS-80 is accessing the Keyboard.

Once in Command Mode you may enable or disable specific utilities by typing a key.

Disk Keyplus contains the following routines:

AUTO-REPEAT. This utility allows you to repeat a key by pressing it without releasing it. This is particularly useful when editing BASIC programs.

AUTO INPUT FILE. This extremely powerful routine allows you to define a string that may be automatically input three different ways: at power up, during Keyplus initiation, or by hitting just two keys.

More flexible than the DOS AUTO command, Disk Keyplus may automatically execute a complex combination of commands and/or programs.

SAVE STATUS. This routine allows users to initialize Disk Keyplus with any combination of utilities enabled or disabled.

LOWERCASE VIDEO. If your TRS-80 has a hardware modification installed for lowercase video, this routine adds the software needed to make it work.

RESTORE LOST BASIC PROGRAM ROUTINE. This routine will restore a

BASIC program that has apparently been lost.

TYPEWRITER STYLE KEYBOARD. Lowercase alphabetic characters are generated without using the SHIFT key. Uppercase requires the SHIFT key.

DIRECT GRAPHIC ENTRY. Allows graphic characters to be typed directly from the keyboard. Three different modes are available.

BASIC SHORTHAND. Generates BASIC key words (i.e., GOTO, NEXT, INPUT, etc.) in a single key stroke. I found that this utility saves considerable time when typing a program.

USER DEFINABLE STRINGS. Two user-definable strings, up to 32 characters long, may be generated in a single keystroke. The contents of these strings may be changed at any time.

These are some but not all of the routines available in Disk Keyplus.

CONCLUSION: I received my copy of Disk Keyplus by first class mail just eight days after I ordered it. That's what I call service!

This is one of the most useful programs I own. The Auto Input Routine alone is worth the price. In short, impressive software at a bargain price.

**Editor/Assembler Plus for TRS-80**
Published by **Microsoft**
**10800 N. E. 8th**
**Bellevue, WA 98004**
Price: **$29.95**
Reviewed by **Ronald L. Nicol**

As an avid assembly language programmer, I have found the Microsoft Editor/Assembler Plus to be invaluable. It is a cassette-based, assembly language development system and is orders of magnitude superior to Radio Shack's Editor Assembler. Prior to my experience with this particular software, I had occasion to use Radio Shack's Editor Assembler and Apparat's Disk Editor Assembler and even though I have a disk system I much prefer the Editor/Assembler Plus to the others.

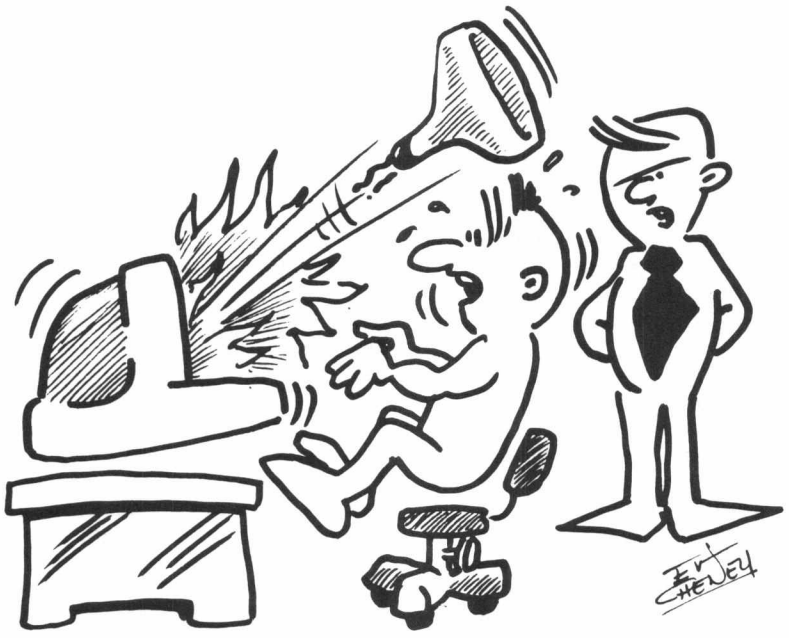
The Radio Shack Editor Assembler is a subset of Editor/Assembler Plus and the Editor/Assembler Plus is supplied with its own monitor, Z-Bug, which itself is much superior to T-Bug, the Radio Shack monitor. Z-Bug and the Editor/Assembler Plus can be co-resident. A powerful advantage of this arrangement is that symbolic debugging is allowed (i.e., locations in the in-memory object code can be referenced by Z-Bug by their labels vice their absolute memory locations). Z-Bug can display values as decimal and octal as well as hexadecimal. Z-Bug allows multiple breakpoints and also single-stepping through a program. All in all it is an impressive monitor.

The Editor/Assembler Plus fully supports Macro usage and conditional assemblies. A useful feature is the ability to assemble in memory while the Editor Assembler and Z-Bug are resident. This eliminates the need to dump object code to tape and reload using a monitor which overwrites the assembler. This reduces debugging time significantly, but this attribute is not without its disadvantage because, as always, there is a trade off between time and memory space, and having the Assembler/Editor and Z-Bug co-resident with the user program can lead to severe program size constraints. For example, if a program is assembled using the automatic origin feature of the Assembler, it will be assembled beginning at location 71ADH. The problem of space is not without a partial solution, however, through the use of the QUASH command which allows the user to purge Z-Bug and the Assembler. QUASHing just Z-Bug frees up 3.2K bytes, while QUASHing both frees 8.1K bytes.

The Editor portion of this software is extensive. In addition to standard editing features, it allows the following: collective editing (i.e., editing a contiguous set of lines), movement and copying of blocks of lines, a substitute command which finds and replaces a specified string, and automatic extend over a range of lines.

Assembler expression evaluation allows addition, subtraction, multiplication, division and logical operators which include Not, Shift, Modulo, Equal and Not Equal operators.

The software is supplied with excellent documentation and includes a reference card of all available functions. The Editor/Assembler Plus requires a 16K Level II TRS-80. The quality-to-price ratio of this program far exceeds any other that is presently available. ■■




**"I warned you not to use that program!"**

# DOSMOVER for the North Star

by Thomas W. Cage

The standard 2000 hex address of the North Star DOS 5.1DQ is inconvenient for many uses. A version for hex address 0100 is available from North Star, but I wanted a version for hex address F000; to get it, I would have to relocate the code of the N*DOS monitor.

In theory, relocation of an 8080 program is simple: find all three-byte instructions and alter their second and third bytes by a relocation factor. This can be made even simpler if the relocation factor (the distance the program is to move) is a multiple of 256. Then only the third byte of the instruction, the most significant half of the address, need be changed.

In practice, it's not so easy. N*DOS contains ASCII constants – messages, command names in the command table, etc. – and if these constants were relocated as if they were instructions, chaos would result. I studied N*DOS very carefully to find the parts that should not be altered.

I made a hex dump of the program on paper – hard copy is essential for this kind of work – and also an ASCII dump. The ASCII dump revealed the command table; the hex dump of the command table showed the addresses of the major routines. The ASCII dump also revealed message constants. With that information I could disassemble the program (I used the XEK dissassembler). With some trial and testing, this information allowed me to define the areas of N*DOS that should be relocated in order to make it run at its new address. I wrote DOSMOVER in North Star BASIC. I will relocate N*DOS version 5.1DQ. Run it as follows:

Load an unaltered copy of the standard N*DOS 5.1DQ at the new address that you want, load BASIC, load DOSMOVER, and type RUN. The prompt will ask for the starting address of the new version of DOS. The runtime after you type <CR> will be about two minutes and thirty-seven seconds. When the done message is printed out, type BYE. Create a disk file twelve sectors long and save the new version of N*DOS.

The DOSMOVER program assumes that the I/O program is the same as the one in the North Star System Software Manual, page H-1. If you are using a custom I/O routine, delete line 330. Change the high address bytes in your custom I/O routine with the monitor program, then save the new N*DOS. The I/O page is at sector 8 on a standard disk.

Since I wrote DOSMOVER, North Star has released a new version of DOS, N*DOS 5.2DQ. This comes as a Type 3 File with a BASIC program called MOVER. When MOVER is RUN against the FILE -DOS, N*DOS 5.2DQ can be located anywhere in memory. However, N*DOS 5.2DQ uses Z-80 relative addressing and therefore will not run on systems using 8080 and 8085 microprocessors.

■■

(LISTING ON PAGE 41)

## Book Reviews

*(Continued from page 17)*

Much of the writing is indeed metaphorical – abstractions, remote analogies, and generalities everywhere, often briefly touching programming problems. The Fibonacci sequence (that I dealt with concretely in *DDJ*#51) is used as an example of a "recursively enumerable set," and later analogized with biological "genotypes" and the deciphering of ancient script. Much of what he says about AI is uncritical or hypothetical – but never boring, and the chapter on Bongard problems (p. 641) was mostly new to me. The fact that the author is straddling – however precariously – so many diverse fields almost guarantees that, whoever you are, some of the near-800 pages (a bargain at a penny a page!) will be instructive or entertaining. Each reader must search out for himself in this chaos the passages that will enlighten or amuse him. *DDJ* readers might minimize perplexity by starting with chapter X, then XIX, then (having perhaps gotten used to the peculiar flickering style) shift to random browsing with a lot of skipping, until they strike their kind of gold. Some may find the author's own "overview" (p. viii-) helpful as a guide, though I did not.

Untamed thinking like Hofstadter's, when forcibly disciplined by the writing of a program that a computer will run, has at times created novel data structures and algorithms that crystallize ever more complex fragments of human intelligence, making possible the doing of specific operations with superhuman speed and precision. Perhaps they can only do this if sustained by faith that anything is possible. The role of skeptics is to remind them that a possible thing is not necessarily humanly desirable or what they hope and believe it is. (What would be the long-term effect of an unbeatable chess-playing program?) Hofstadter says (p. 676, speaking of AI) "And one must run on faith at this point, for there is so far to go!" To get to *where*? Had the alchemists succeeded, would our fate have been that of Croesus?

It's a book of questions and problems, not answers and solutions, nor even many clear hints on how to work anything out. It won the Pulitzer prize not for utility but "literary" value – food for thought, glimpses of a "big picture," and what other reviewers have seen as an "exciting" and "exhilarating" quality. One wonders if his next book, in the style of Lewis Carroll, won't be the *Adventures of Alice in Computerland*! With frenzy subdued into whimsy, and with greater insight into human minds and their emulation by machines, a fantasy might be an effective way to transmit to a multifarious readership some understanding of what computers really are.

■■

```
10  REM *** North Star DOS 5.1DQ Mover
20  REM *** Orig.  Tom Cade   15 September 1980
30  REM *** PUBLIC DOMAIN   30 October 1980
40  REM ***
50  REM *** B=Base Address, High Byte        M=Memory Address
60  REM *** L=Low Address                    H=High Address
70  REM *** D=Data at Address M              E=Error
80  REM *** Runtime about 2 minutes 37 seconds
90  REM ***
100 !\ !\ !\ INPUT "Enter DOS Starting Address In HEX ",M$
110 IF LEN(M$)=4 THEN 120\ !"Input Error"\ !\ GOTO 100
120 M=0\ FOR I=1 TO 4\ M1=ASC(M$(I,I))\ GOSUB 450\ IF E=1 THEN EXIT 100\ NEXT I
130 REM ***
140 REM *** Base Address
150 B=INT(32-(8192-M)/256)
160 REM ***
170 REM *** Modify DOS
180 FILL B*256,B\ REM *** DOS High Byte Starting Address for ROM BOOT
190 L=B*256+10\ H=B*256+42\ GOSUB 530
200 M=B*256+46\ D=EXAM(M)\ FILL M,D+B-32
210 M=B*256+50\ D=EXAM(M)\ FILL M,D+B-32\ REM ** High Byte Input Buffer Address
220 L=B*256+53\ H=B*256+80\ GOSUB 530
230 L=B*256+99\ H=B*256+654\ GOSUB 530
240 M=B*256+169\ FILL M,B-1
250 L=B*256+663\ H=B*256+1455\ GOSUB 530
260 L=B*256+1481\ H=B*256+1791\ GOSUB 530
270 M=B*256+1708\ D=EXAM(M)\ FILL M,D+B-32
280 M=B*256+1713\ D=EXAM(M)\ FILL M,D+B-32
290 M=B*256+1732\ D=EXAM(M)\ FILL M,D+B-32
300 L=B*256+1883\ H=B*256+1957\ GOSUB 530
310 REM ***
320 REM *** Standard I/O Routine, page H-1
330 L=B*256+2048\ H=B*256+2239\ GOSUB 530
340 REM ***
350 L=B*256+2304\ H=B*256+2388\ GOSUB 530
360 M=B*256+2309\ FILL M,32
370 L=B*256+2421\ H=B*256+2631\ GOSUB 530
380 FOR M=B*256+2635 TO B*256+2675\ REM *** Command Table
390     D=EXAM(M)\ FILL M,D+B-32\ M=M+3
400 NEXT M
410 L=B*256+2676\ H=B*256+3063\ GOSUB 530
420 M=B*256+2702\ D=EXAM(M)\ FILL M,D+B-32
430 GOTO 880


440 REM ***
450 REM *** Convert HEX to Decimal
460 M=M*16\ E=0
470 IF (M1>47) AND (M1<58) OR (M1>63) AND (M1<71) THEN 490
480 !"Invalid HEX Character"\ !\ E=1\ GOTO 510
490 IF M1<58 THEN M1=M1-48 ELSE M1=M1-55
500 M=M+M1
510 RETURN
520 REM ***
530 REM *** Test for 1, 2, or 3 Byte Instructions
540 FOR M=L TO H
550     D=EXAM(M)
560     REM *** 18 2 Byte Instructions
570     IF (D=198) OR (D=206) OR (D=230) OR (D=254) OR (D=219) THEN 700
580     IF (D=246) OR (D=211) OR (D=214) OR (D=222) OR (D=238) THEN 700
590     IF (D=062) OR (D=006) OR (D=014) OR (D=022) OR (D=030) THEN 700
600     IF (D=038) OR (D=046) OR (D=054) THEN 700
610     REM *** 26 3 Byte Instructions
620     IF (D=205) OR (D=204) OR (D=196) OR (D=244) OR (D=252) THEN 720
630     IF (D=220) OR (D=212) OR (D=236) OR (D=228) OR (D=049) THEN 720
640     IF (D=202) OR (D=194) OR (D=242) OR (D=250) OR (D=218) THEN 720
650     IF (D=210) OR (D=234) OR (D=226) THEN 720
660     IF (D=195) OR (D=017) OR (D=033) OR (D=034) OR (D=042) THEN 730
670     IF (D=050) OR (D=058) THEN 730
680     IF (D=001) THEN 710
690     GOTO 740\ REM *** 200 1 Byte Instructions and 12 Null Instructions
700     M=M+1\ GOTO 740
710     M=M+2\ GOTO 740
720     GOSUB 820\ GOTO 740
730     GOSUB 770
740 NEXT M
750 RETURN
760 REM ***
770 REM *** Test Range for High Byte
780 D2=EXAM(M+2)
790 IF (D2>31) AND (D2<45) THEN GOSUB 820 ELSE M=M+2
800 RETURN
810 REM ***
820 REM *** Modify High Order Address
830 D2=EXAM(M+2)
840 FILL M+2,D2+B-32
850 M=M+2
860 RETURN
870 REM ***
880 REM *** End of Program
890 !\ !\ !\ !"Finished"\ !"Save DOS on Disk"\ !\ !\ !
900 END
```

# Video Underline Modification

One rainy day I had finished my semi-annual look, feel, and wiggle inspection of all the cards, wiring, plugs and cabling in my Altair 8800B. The system diagnostic program verified that the cassette system was tuned and working fine, and all memory had passed the exhaustive pattern sensitivity test. I was looking at the video display and trying to remember any ideas for minor hardware or software changes that had occurred to me, when my thoughts turned to the joys and pains of screen text editors.

The little beast used on the video display to indicate a control location is well named a cursor. The aggravation of trying to guess where a blinking cursor will appear can give unwanted frustration to text entry. My cursor on the video display is the inverted video for any or all characters on display, and usually only one cursor is allowed on the display at any instant. I wondered if using an underline would look better and ease some operator strain. The more I considered using underline instead of total video inversion of a single character, the more the idea appealed to me. The next step was to find out how difficult changing the video inversion to an underline would be on my VDM-1 card (from Processor Technology).

Examination of the character-generation process in the VDM-1 manual led to a clear definition of what would have to be done to give the underline effect. The ninth row of dots (figure one) of the character would be inverted if D7 (the eighth bit of the ASCII character being formed) was high. Armed with this specification, an examination of the VDM-1 schematic led me to believe that a few cut-and-bypass changes, and the addition of an IC to decode the ninth row of dots, would allow this improvement.

The schematic of the modification was drawn with a checklist of cut-and-bypass changes, to insure that every change was done correctly. The actual layout was considered and a good site was chosen for the additional IC to be placed

*(Continued on page 49)*

**by Dannie E. Davis**

*Dannie E. Davis, 101st MI Bn, Ft. Riley, KS 66442.*

**Figure 1.**

**Dot matrix of character "y".**

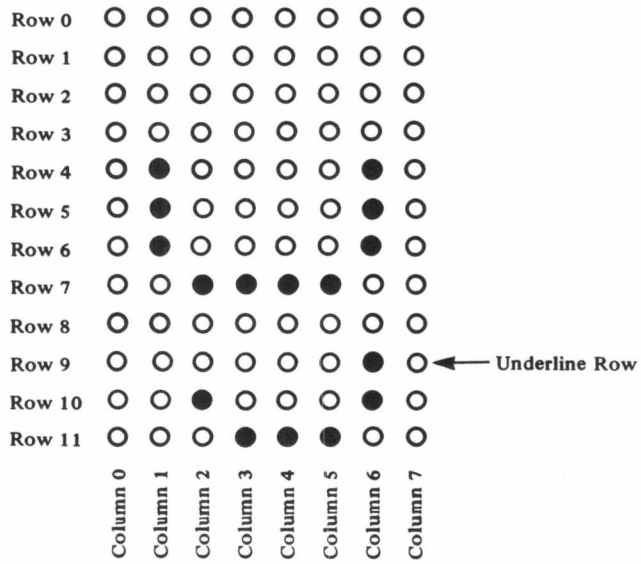
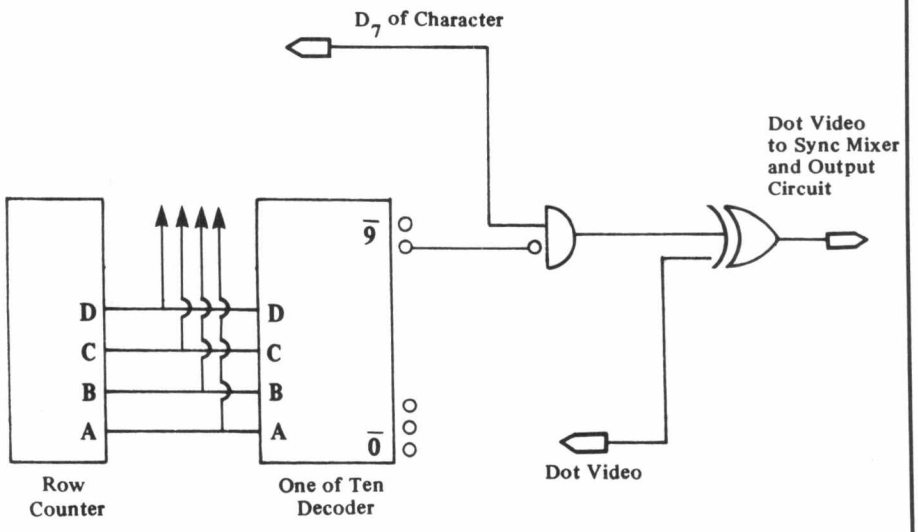



**Figure 2.**

**Simplified Underline Circuit**



# TRS-80 Voice Synthesizer on Other Microcomputers


## Abstract

The TRS-80 Synthesizer compares favorably in price ($399), quality of speech, and ease of use with other synthesizers. Sales and service for the synthesizer are widely available and software for it is beginning to appear. The main drawback to date has been the fact that it could only be used with TRS-80 computers. This article will show how to interface the synthesizer to the S-100 bus and to the Exidy Sorcerer bus, and also will describe general principles applicable to interfacing the syntheiszer to any microcomputer. (All the computers we have used to date have clocks that run at approximately 2 megahertz. There may be timing problems in using our circuitry on other computers.) The interface requires only wires and connectors; the total cost should be from zero to thirty dollars depending upon what is available in your scrap box. The only labor required is soldering three dozen connections.


## The TRS-80 Voice Synthesizer

The TRS-80 Voice Synthesizer is a small gray box. To operate it with a TRS-80, all one has to do is to plug the synthesizer's connecting cable into the computer and its power supply into an electric socket. Then the synthesizer will say whatever is sent to it in special codes. (For instance, if the computer sends it the string "H;U;RY(UU" it will say "How are you.") Software is advertised in computer magazines by Supersoft which automatically turns ordinary English into these special codes. The quality of the synthesizer's speech is close to that of synthesizers costing ten times as much. After a few hours of listening, the average user can understand it fairly well.

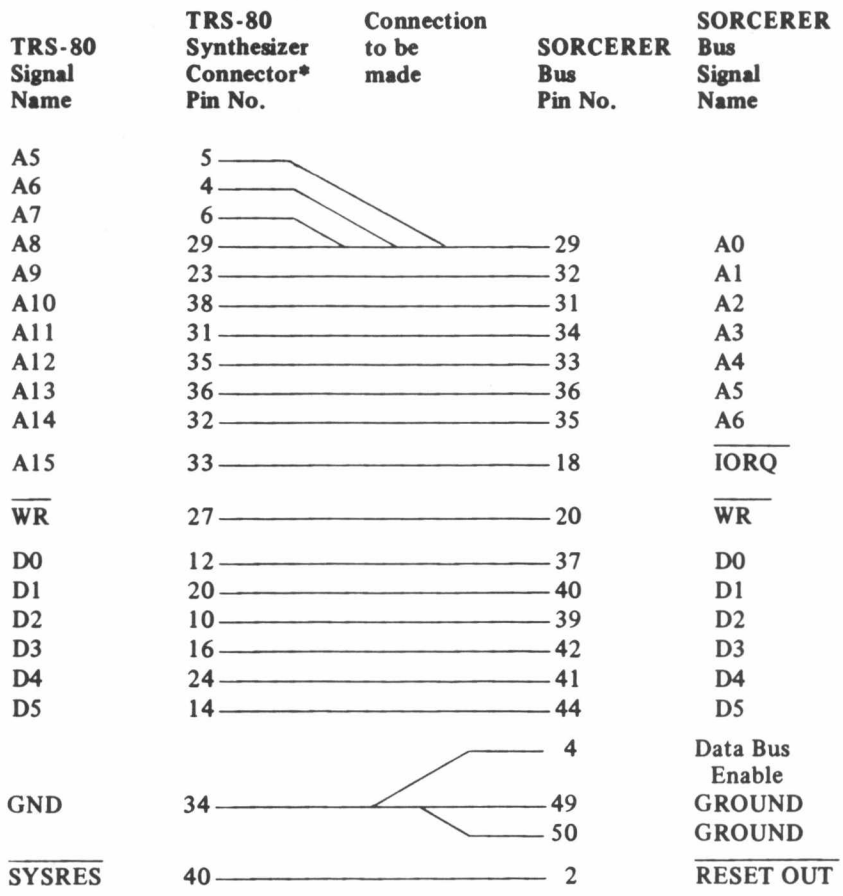
## How the TRS-80 Synthesizer Interfaces to the TRS-80

The first step in interfacing the TRS-80 synthesizer to your computer is to understand how Radio Shack interfaced it to the TRS-80. Radio Shack wanted to make it easy for anyone who knew BASIC to run the synthesizer. Therefore they designed the synthesizer so that anything written by a BASIC program on the bottom right corner of the screen would be received by the synthesizer.


**by Michael Hightower and Peter B. Maggs**


FIGURE 1

TRS-80 SYNTHESIZER CONNECTIONS TO THE EXIDY SORCERER




*WARNING – Synthesizer connector pin numbers are *not the same* as pin numbers on the connector on the TRS-80 keyboard unit.

If you had a Level II TRS-80, the first step would be to clear out the synthesizer with the following BASIC statement:

```
PRINT@ 992,"?-?"
```

Since the location 992 is in the lower right corner of the TRS-80 screen, this statement would send the characters "?", "-", and "?" to the synthesizer in that order. The question mark is a signal to the synthesizer to turn on if it is off and to turn off if it is on. (In Radio Shack terminology the first question mark "opens the window" and the second question mark "closes the window." The dash signals the synthesizer to be silent.)

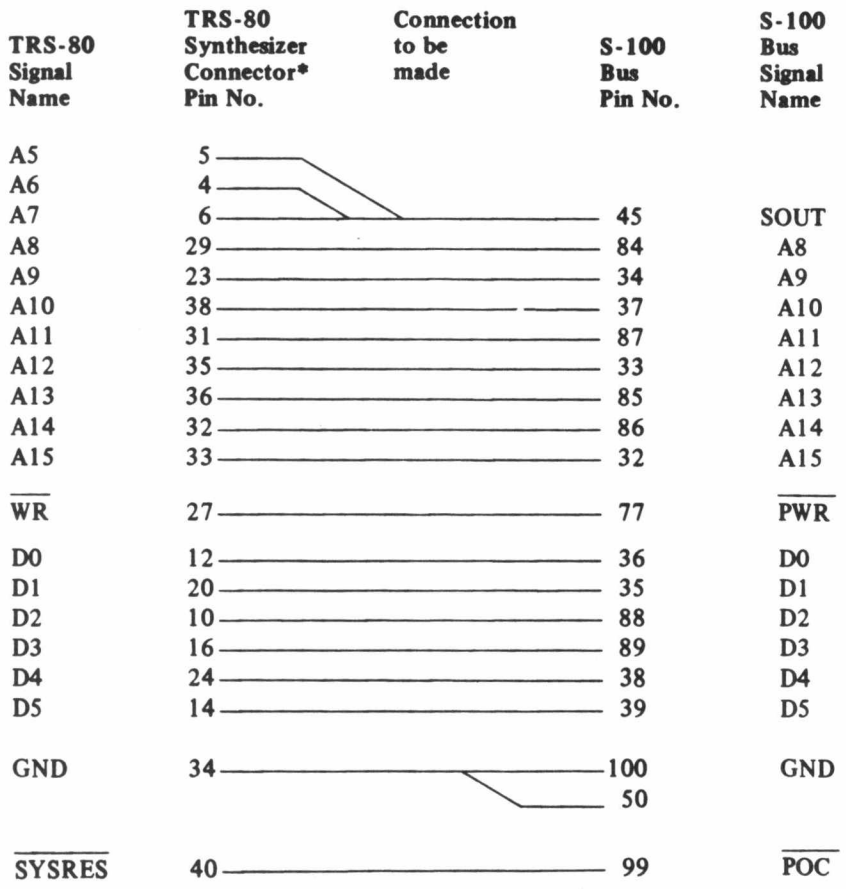
Then the synthesizer may be made to speak. For instance on a TRS-80 the following BASIC program would cause the synthesizer to say "one, two, three."

```
10 PRINT@ 992,"? W877NN ?"
20 PRINT@ 992,"? T((UU ?"
30 PRINT@ 992,"? ==R.& ?"
```

The synthesizer has a thirty-character buffer. In programming, one must take care not to get too far ahead in sending its words or the buffer will overflow and the synthesizer will stop speaking.

## FIGURE 2

**TRS-80 SYNTHESIZER CONNECTIONS TO S-100 BUS**




*WARNING – Synthesizer connector pin numbers are *not the same* as pin numbers on the connector on the TRS-80 keyboard unit.

How does the synthesizer know that something is written in the bottom right corner of the screen? In the TRS-80 computer, the video screen displays whatever is in memory locations 3C00H through 3FFFH (15360 decimal through 16383 decimal). Anything in the last thirty-two locations may be a signal to the synthesizer. The synthesizer is attached to the TRS-80 bus. It decodes bus address signals to determine if a location between 3FE0H and 3FFFH is being addressed. This will be so whenever address lines A14 and A15 are low and address lines A5, A6, A7, A8, A9, A10, A11, A12 and A13 are high. Then, when the Radio Shack Memory Write Strobe Output Signal goes low, the synthesizer accepts the data present on lines D0, D1, D2, D3, D4 and D5.

## Interfacing the Synthesizer to Other Computers

The secret of interfacing the synthesizer to other computers is to find the lines on its bus which correspond to the lines on the Radio Shack bus used by the synthesizer. In our interfacing, we have chosen to use port rather than memory-mapped output, because the memory of our computers was entirely dedicated to other purposes; however, it would be quite simple to adapt our circuit diagrams and programs for memory-mapped output.

The only parts needed for each of the computers we have interfaced it with to date (SOL, North Star running at 2 megahertz, Vector Graphic, Exidy Sorcerer), are one connector to fit the computer bus and one connector to fit the synthesizer connector. For the S-100 bus, any sort of 100-pin male S-100 connector or blank S-100 board will do. These are sold by almost all computer stores and by many mail order companies. For the Sorcerer bus, a connecting cable is available from Exidy dealers. We were unable to find a ready-made male connector to fit in the female connector on the Radio Shack synthesizer cable; you may have better luck. If you do not, buy a blank circuit board with one-tenth inch spacing between connectors and at least 20 connectors on each side of the board, and then cut it to fit exactly in the synthesizer connector. Essentially, the idea is to duplicate the male edge connector that protrudes from the left rear of the TRS-80 keyboard unit. We do not recommend

the alternative of cutting the connector off the synthesizer and soldering the wires straight to the S-100 connector. This will void your warranty and make it impossible to test your synthesizer on Radio Shack equipment.

The next step is to connect the two connectors as shown in Figure 2 for the S-100, or in Figure 1 for the Sorcerer bus. Check your wiring and soldering carefully, since most microcomputer buses have power and ground pins located close to one another; connecting them together could be fatal to your computer.

Once the adapter cable has been made, all you have to do is plug in the synthesizer and you are ready to make it talk. To do this, you will need some software. Typical BASIC and assembly language software is shown in Figures 3 and 4. Using this software requires an understanding of the system of phonetic codes explained in the TRS-80 Synthesizer manual. One of the authors has written more elaborate software that translates ordinary English into the needed phonetic codes and automatically buffers output to avoid getting ahead of the synthesizer. This software is available from Supersoft, Box 1628, Champaign, IL 61820. ■■

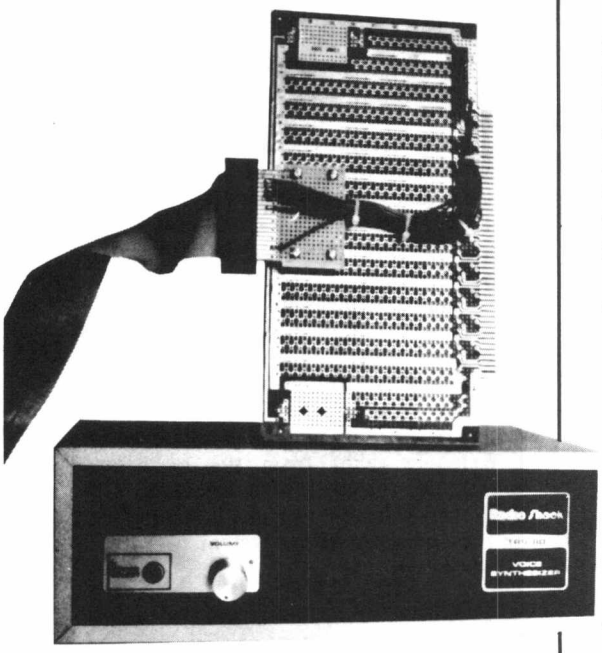



**Figure 3**

```
PAGE 01
A/VOICE
0001                    * 8080 PROGRAM FOR TRS-80
0002                    * VOICE SYNTHESIZER
0003                    *
0004                            ORG   256
0005                    PORT    EQU   63      ; OUTPUT PORT (3FH)
0006 0100 212401                LXI   H,INIT  ; INITIALIZE
0007 0103 CD0F01                CALL  TALK
0008 0106 212601                LXI   H,MSG   ; SEND MESSAGE
0009 0109 CD0F01                CALL  TALK
0010 010C C30C01        WAIT    JMP   WAIT    ; RESET TO ESCAPE
0011                    *
0012 010F 3E3F          TALK    MVI   A,63    ; CODE TO OPEN WINDOW
0013 0111 D33F                  OUT   PORT
0014 0113 7E            TLOOP   MOV   A,M
0015 0114 FEFF                  CPI   255     ; TEST FOR END
0016 0116 CA1F01                JZ    TDONE
0017 0119 D33F                  OUT   PORT    ; SEND OUT PHONEME
0018 011B 23                    INX   H       ; POINT TO NEXT ONE
0019 011C C31301                JMP   TLOOP
0020 011F 3E3F          TDONE   MVI   A, 63   ; CODE TO CLOSE WIN-
                                              DOW
0021 0121 D33F                  OUT   PORT
0022 0123 C9                    RET
0023                    *
0024 0124 2D            INIT    DB    45      ; CODE TO CLEAR SYN-
                                              THESIZER
0025 0125 FF                    DB    255     ; END OF MESSAGE
0026                    *
0027 0126 14            MSG     DB    20      ; CODE FOR T
0028 0127 33                    DB    51      ; CODE FOR E
0029 0128 13                    DB    19      ; CODE FOR S
0030 0129 14                    DB    20      ; CODE FOR T
0031 012A 20                    DB    32      ; CODE FOR PAUSE
0032 012B FF                    DB    255

**************************
```

**Figure 4**

```
LIST

100 REM BASIC PROGRAM TO USE THE
110 REM TRS-80 VOICE SYNTHESIZER
120 REM WRITTEN IN NORTH STAR BASIC
130 DIM S$(25)
140 REM INITIALIZE SYNTHESIZER
150 LET S$="-"
160 GOSUB 260
170 REM NOW SAY STRINGS THAT ARE TYPED IN
180 PRINT "TYPE IN PHONETIC CODES"
190 INPUT S$
200 GOSUB 240
210 REM LOOP BACK FOR MORE
220 GOTO 190
230 REM
240 REM SUBROUTINE TO SEND OUT PHONETIC CODES
250 REM START WITH ? TO OPEN WINDOW
260 OUT 63,ASC("?")
270 REM NOW SEND OUT STRING CHARACTER BY CHARACTER
280 FOR N=1 TO LEN(S$)
290 CHECK YOUR BASIC MANUAL FOR SUBSTRING NOTATION
300 REM OUTPUT PORT 63 DECIMAL IS 3F HEXADECIMAL
310 OUT 63,ASC(S$(N,N))
320 NEXT N
330 REM NOW A PAUSE FOR SILENCE
340 OUT 63,ASC(" ")
350 REM NOW A ? TO CLOSE WINDOW
360 OUT 63,ASC("?")
370 RETURN
READY
RUN

TYPE IN PHONETIC CODES
?H38L80U
?H;U;RY(UU
?
STOP IN LINE 190
READY
```


Michael Hightower received the B.S. and M.S. degrees in electrical engineering from the University of Illinois. He is Research Engineer at the Computer-Based Education Research Library at the University of Illinois in Urbana, Illinois. He joined the staff in 1975, participating in the design of the PLATO V terminal and resident operating system. Recently he has been involved in the design of a low cost PLATO terminal using a color CRT display and automated microcomputer system diagnostic equipment.

Peter B. Maggs majored in mathematics and foreign languages at Harvard College and received a law degree from Harvard Law School. He is currently Professor of Law at the University of Illinois in Champaign, Illinois. He has been active in computer applications in the law and in the development of talking computer hardware and software for blind and speech handicapped users. Talking computer terminals of his design have been in use by blind lawyers in the United States Department of Justice since 1978.


## Video Underline Modification

*(Continued from page 42)*

ing was modified to produce underline instead of character video inversion (figure two).

After carefully inspecting the finished modification, I was pleasantly surprised when the system operated perfectly on the first try. Full diagnostic testing found that any resident gremlins were on vacation.

I am very pleased by the change in the video appearance and find new ways to use the underline on the display. My favorite use is a fill-in-the-blank line for menu or local entries. This permits complex, static, interactive page entry, and is a welcome relief from the occasional nausea of scrolling text displays. This small hardware change has a large effect on the operation of my system.

■■

# Of Interest

by Dave Cortesi

One of the perks of working around an editorial office is that we get to read a lot of press releases. We learn of all the latest and (the press releases claim) greatest things. By reading between the lines we get hints of what's going on in the trade. Sometimes we compare the PR fluff in the releases to the *real* information – industry word of mouth, the infallible source – and thereby get a lot of laughs.

Starting this month we're going to experiment with the "Of Interest" section. Instead of excerpting from the press releases with only minor editing, we're going to comment more freely, juxtapose, and rewrite. Let us know how we're doing, won't you?

## Get Rich Quick – Become a Software Vendor . . .

Texas Instruments took a lot of knocks for not providing applications software for their 99/4 Home Computer. The company got the point, and is responding vigorously. According to the first of a flurry of releases, TI "recognizes that the company cannot produce all the software required for all the applications for which the TI-99/4 is well suited." As a result, "TI offers software development systems for sale to . . . software authors who want to develop Solid State Software command modules."

The new 99/4 development system includes an extra 32K of RAM (for a total of 48K), two disk drives and their controller, an assembly language module, and two new languages. One of these is the UCSD Pascal system from Softech! The other is TI Extended BASIC, with a number of graphics statements and the ability to write true subroutines with their own local variables.

If you would like to become a third-party software vendor for the TI-99/4 (and that appears to be the only way to get the development system, at least for now), call TI. If you live in Texas, call 800-692-4979; if you're unfortunate enough to live outside the Lone Star State call 800-858-4565.

## . . . Then Get Your Name in a Dictionary . . .

An organization called The American Microcomputer Association (POB 1266, Glen Burnie, Maryland 21061) is compiling a directory of microcomputer consultants. Write to them, they'll list you in it for $10. They promise to send you a verification printout before publication, and plan to place their directory in "Libraries, Computer Stores, and other key locations."

## . . . And Publish a Scholarly Article

The erstwhile *Journal of Design Automation and Fault Tolerant Computing* has decked itself out with a new title. It is now the *Journal of Digital Systems*, and has expanded its areas of interest to include computer architecture, performance evaluation, and the like. The first issue has handsome typography and looks like a worthy spot to place your thesis – but if you want appreciative readers, think of *DDJ*.

## Buy an ADA Manual . . .

That music you hear in the distance is the ADA bandwagon. Pony Express Services (100 West 57th St., NY, NY 10019) wants to help you jump on it. They'll sell you a manual for the Department of Defense's new standard language for just $100, and then keep it up to date for you. "ADA is certain to become the most-used computer language in the world," they burble happily. *DDJ* welcomes comments on that statement, and on ADA in general.

## . . . or One on FORTH

If that's too high a leap, even for a very fancy bandwagon, then a $20 step will get you onto the FORTH pogo stick. Send it to the Mitchell E. Timin Engineering Co. at 9575 Genesee Ave., Suite E-2, San Diego, CA 92121, and they'll send you a 67-page manual on Timin FORTH, "an enhancement to FIG-FORTH for CP/M." That's 29.8 cents/page, which raised our eyebrows, but then FORTH is a *very* concise language.

## Talk to a Publisher's Computer

Note another entry in your little black book of CBB phone numbers. McGraw-Hill Book Company has initiated a computer bulletin board. It operates from 6 pm to 8 am EST Monday to Thursday, and 6 pm Friday to 8 am Monday. Call it on 212-997-2186 and have it tell you about McGraw-Hill's titles. They use an Apple II and the ABBS software from Peripherals Unlimited.

## Buy a Bunch of CP/M Software

Lifeboat Associates (1651 Third Ave., NY, NY 10028) announces the availability of RAID, a new software debugging tool for the 8080/8085 under CP/M. We have to fault the publicist who wrote the release, which sketches a number of the program's features but fails to make clear exactly wherein RAID is an improvement on good old SID. There must be such points, for at $250 RAID costs twice as much. The manual should make it clear, at $25. *DDJ* would like to see articles on debuggers, especially comparative reviews of two or more of them.

Computer Control Systems (298 21st Terrace S.E., Largo, FL 33541) have produced Ultrasort II, a machine-language sort program that can be used alone or as a subroutine of CBASIC. This sort uses an interesting tehcnique to acquire working storage space; it rolls out the whole Transient Program Area to disk, uses all the space to do the sort, then rolls the calling program back in again. Wonder how it does on very short files?

Supersoft (POB 1628, 40 Main Street Suite 402, Champaign, IL 61820) has enhanced its CP/M diagnostic package. Each of the test components can be run from a submit file – an excellent move, in our opinion – and will write its report to a disk file, so presumably an entire checkout session could be automated into one submit command. *DDJ* will be printing articles on diagnostic programs in upcoming issues, and we wouldn't mind seeing more, including reviews of packages like this one.

The same company announces C compilers for the 8080 and the Z8000. The compiler is really one program, written in C, compiling itself three different ways. It is a two-pass program, claiming to optimize for both space and speed (a claim we beg leave to doubt, but then compiler optimization techniques are not normally the province of the authors of press releases). The CP/M version, producing 8080 object code, is priced at $175. A CP/M version cross-compiling to Z8000 code is $500, as is the Z8000-

coded compiler. The *DDJ* Editor would be in seventh heaven if someone who really understood compiler technology would write a thorough, comparative review of the C compilers available for the 8080.

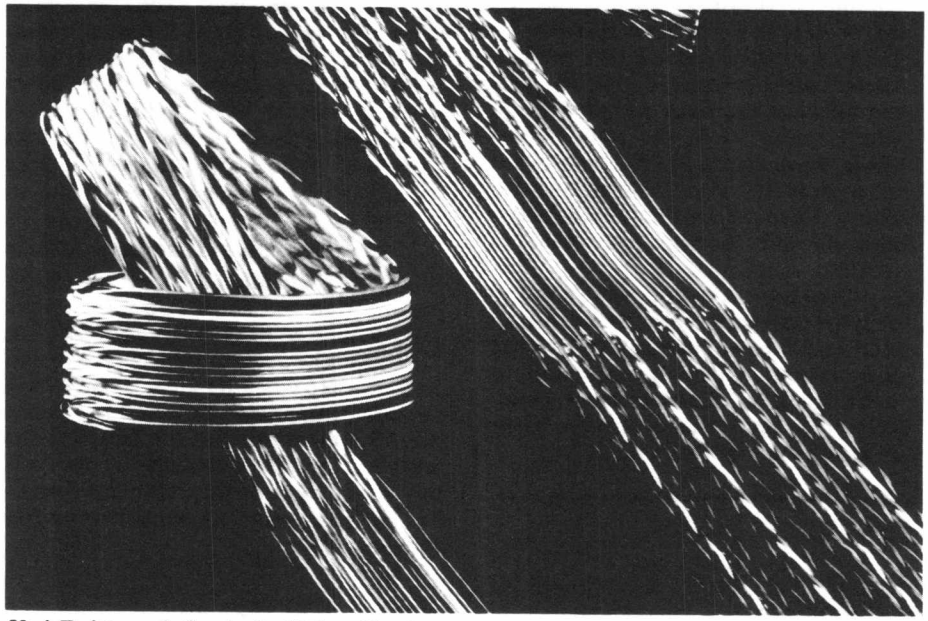
Digital Research hasn't been sitting on its byte, either. They've come up with a new access method for CP/M. Well, they call it a "record retrieval system," but an access method is what it is, and not a bad one to judge from the fairly sketchy information in the release. BT-80 (with a little TM just above and to the right, please) maintains a balanced beta-tree index or indices to a collection of records. The data can be spread across as many as 8 files to reach a maximum of 64MB – not too shabby for a microcomputer. Records can be inserted, deleted, and retrieved by key through as many as 4 indices concurrently, or scanned in key or reverse-key sequence. The PL/I-80 runtime library package is a pre-req; it and BT-80 together will cost you $300. Write to POB 579, Pacific Grove, CA 93950 or call 408-649-3896.

## Learn to Make Circuit Boards . . .

Those who want to work on the leading edge of home computing must be ready to make their own parts. Heathkit has added a course to their line of self-teach courses that may help prepare you. EI-3134 is a course in the construction of printed circuit boards. Besides the text, the course package contains all the components of two of Heath's simpler kits *except for the PCB's*, a stroke of motivational genius. Just complete the course, creating the missing boards as lab exercises. Price is $65; package available only in the continental USA; catalog from Heath, Dept 350-550, Benton Harbor, MI 49022.

## . . . Connect Them With Impossible Cable

Given the present concern over noise emission it had to happen, but it still looks topologically impossible. Belden Corp. has introduced twisted-pair ribbon cables. They call it Vari-Twist, and it's a flat cable composed of tiny twisted pairs of conductors. Adjacent pairs have opposing twists; periodically the whole thing flattens out to look like ribbon cable for an inch or so, then the pairs twist up again. How do they do it? Ask the Manager, Marketing Communications, Belden Corp., 2000 S. Batavia Ave., Geneva IL 60134. If you solve a sticky RFI problem, with Vari-Twist or without, write about it for *DDJ*.



**Vari-Twist: a solution to the RFI problem?**

## Control 64 Voltages at Once

An intriguing S-100 board has been announced by an outfit called Digital Multi-Media Control (2338 Patterson #12, Eugene, OR 97405, 503-342-3181). This single board contains 64 digital-to-analog (D/A) converters. Sixty-four channels on a single board? They do it by multiplexing one very hard-working 8-bit D/A converter across 64 latches. All 64 channels are memory-mapped to a single 256-byte page of address space. The makers suggest you could control 64 AC motors, or drive 64 lights in a disco. They note that a channel is just $8.25, for a board price of $514 (check it; there's some quantization error in those numbers).

## Talk a Little Faster

If you don't mind being non-standard, a new acoustic modem from Anderson Jacobson will let you communicate at 450 bits per second, 50% faster than the usual 300 bps (which it also supports). The AJ 347 is crystal-controlled, works in originate or answer mode, and will auto-answer. It costs $365; just ask at 521 Charcot Ave., San Jose, CA.

## Get an LSI-11 for Less

This is an older item; some of you may already have heard. Digital Equipment Corp. has entry-level versions of their LSI-11/2 and 11/23 products, and has reduced prices on all of those machines. For example, an LSI-11/23 with 32KB and two serial lines is now priced at $2900; an LSI-11/2 with 64KB and two serial lines has been reduced 20% to $2100. DEC didn't note whether those prices got you a case, or a power supply, or other details, but if you've been anxious to get into 16-bit machinery these might repay investigation. They didn't give us an address for more information, but a DEC salesman shouldn't be too hard to find.

## Go to a Conference; Present a Paper

**The 5th International Conference on Software Engineering** will be held March 9-12, 1981 at the Town and Country Hotel, San Diego. A highlight of the conference will be a Software Tool Fair featuring demonstrations of software tools and development systems. Write (quickly) to Ray Houghton, National Bureau of Standards, Technology Building, Room A265, Washington, DC 20234.

Did we pass that one on too late? You can still make **The Cincinnati Busi-**

ness Show at the Cincinnati Convention-Exposition Center, March 31 to April 2. The latest in office equipment and services, and a program of business seminars will be presented. Contact Ray Nemo, Show Manager, 5679 Creek Road, Cincinnati, OH 45242, 513-531-5959.

Educators, and authors of courseware, might like to check out **AEDS '81**, the 19th annual convention of the Association for Educational Data Systems. It happens May 5-8 in Minneapolis. Contact AEDS 1981 Convention Registration, 1201 16th Street NW, Washington, DC 20036.

If you plan it right you can follow that with **Microcomputers in Education**, the first annual conference of CUE, Computer-Using Educators. That'll be held May 15-16 in Bloomington, California (where?). This is a newer, possibly looser organization than AEDS. If you want to speak or display to the group, call Craig Walker, Conference Director, at 714-886-9118. To pre-register, write Forest Miller, Co-ordinator of Math Science, San Bernardino County Schools, 602 S. Tippecanoe, San Bernardino, CA 92415.

A whole week of artificial intelligence talks might be purgatory for some of us, but if it's heaven for you, you can get it at the **1981 International Joint Conference on Artificial Intelligence**, August 24-28. The deadline for papers has passed, but you can get attendance details from Louis G. Robinson, American Association for Artificial Intelligence, Stanford University, Box 3036, Stanford, CA 94305.

If you have something to say about programming productivity (and what programmer doesn't?) you might consider saying it to the **IEEE Computer Society** at their Fall 1981 conference. "Productivity – an Urgent Priority" will be the topic, and Dr. Harlan Mills will be general chairman. Submit your paper in four copies to one of the program chairmen, Drs. Raymond T. Yeh and Victor R. Basili, University of Maryland, Department of Computer Science, College Park, MD 20742. If you would rather just listen, write to COMPCON FALL '81, POB 639, Silver spring, MD 20901. ■■

# The Electric Phone Book

A computerized bulletin board works just like an ordinary bulletin board system except that instead of paper and thumbtacks it uses a terminal, a computer, and the dial-up telephone network.

The list below was developed from several sources including the Peripheral People in Mercer Island, Washington, and the People's Message System in Santee, California. It is being maintained by People's Computer Company's PCNET project, our effort to bring computers and telecommunications into the hands of everyone. While this is the most complete listing we have as of this writing, we would appreciate additions and corrections. Send them to PCNET, PCC, P.O. Box E, Menlo Park, CA 94025.

All the bulletin board systems listed here can be accessed by telephone using a 300-baud ASCII terminal and a Bell 103 modem. Most use carriage-return as a speed recognition character, after which they are self-teaching. All are free to anyone who calls, unlike Arpanet, which is restricted, and The Source and MicroNet, which cost money. The list has been sorted by area code; consult your local telephone directory for geographical correspondence. (Also printed in *Recreational Computing*, March/April 1981).

**(201) 283-2724**
(201) 457-0893
**(201) 688-7117**
(201) 753-1225
**(201) 753-8152**
(201) 835-7228
**(201) 843-4563**
(201) 874-6833
**(201) 891-7441**
(201) 968-1074
**(202) 337-4694**
(202) 635-5730
**(203) 348-6353**
(203) 357-1920
**(203) 746-4644**
(205) 945-1489
**(206) 244-5438**
(206) 246-8983
**(206) 482-5134**
(206) 482-5590
**(206) 524-0203**
(206) 546-6239
**(206) 723-3282**
(206) 937-0444
**(209) 638-6392**
(212) 245-4363
**(212) 448-6576**
(212) 787-5520
**(212) 997-2186**
(213) 276-4276
**(213) 316-5706**
(213) 329-3715
**(213) 340-0135**
(213) 346-1849
**(213) 349-5728**
(213) 360-6332
**(213) 394-1505**
(213) 395-1592
**(213) 396-3905**
(213) 424-3506
**(213) 428-4718**
(213) 459-3177
**(213) 459-6400**
(213) 566-8035
**(213) 631-3186**
(213) 657-8803

**(213) 673-2206**
(213) 675-8803
**(213) 709-5423**
(213) 795-3788
**(213) 787-4004**
(213) 799-1632
**(213) 799-6514**
(213) 826-0325
**(213) 828-3400**
(213) 843-5390
**(214) 288-4859**
(214) 634-2668
**(214) 634-2775**
(214) 641-8759
**(216) 644-1965**
(216) 754-7855
**(301) 344-9156**
(303) 759-2625
**(303) 789-0936**
(305) 261-3639
**(305) 566-0805**
(305) 689-3234
**(305) 772-4444**
(305) 821-7401
**(305) 989-9647**
(309) 688-0470
**(309) 694-6531**
(312) 255-6489
**(312) 269-8083**
(312) 337-6631
**(312) 420-7995**
(312) 622-8802
**(312) 528-7141**
(312) 545-8086
**(312) 622-9609**
(312) 767-0202
**(312) 782-8180**
(312) 782-9751
**(312) 941-9009**
(312) 964-7768
**(313) 288-0335**
(313) 357-1422
**(313) 465-9531**
(313) 477-4471
**(313) 484-0732**
(313) 569-2063

**(313) 588-7054**
(314) 838-7784
**(316) 746-2078**
(319) 353-6528
**(319) 557-9618**
(404) 394-4220
**(404) 733-3461**
(404) 790-8614
**(404) 793-1045**
(404) 939-1520
**(404) 939-8429**
(404) 953-0723
**(405) 353-2554**
(405) 528-8009
**(408) 241-1956**
(408) 296-5799
**(408) 263-0248**
(408) 263-9650
**(414) 241-5406**

**(414) 241-8364**
(414) 282-8118
**(415) 348-2139**
(415) 348-2396
**(415) 493-7691**
(415) 527-0400
**(415) 647-9524**
(415) 661-0705
**(415) 683-4703**
(415) 792-8406
**(415) 851-3453**
(415) 948-1474
**(417) 862-7852**
(419) 865-1584
**(502) 245-8288**
(502) 896-9624
**(503) 646-5510**
(512) 657-0779
**(513) 671-2753**
(513) 874-2283
**(515) 279-8863**
(516) 938-9043
**(523) 223-3672**
(602) 866-0258
**(602) 956-5612**
(602) 957-4428

**(602) 957-9282**
(604) 687-2640
**(607) 754-5571**
(607) 797-6416
**(609) 983-5970**
(612) 561-6311
**(612) 929-8966**
(614) 272-2759
**(614) 649-7097**
(615) 254-9193
**(617) 354-4682**
(617) 388-5125
**(617) 431-1699**
(617) 649-7097
**(617) 692-3973**
(617) 864-3819
**(617) 897-0346**
(617) 963-8310
**(702) 826-7234**
(702) 873-9491
**(703) 281-2125**
(703) 281-2222
**(703) 379-0303**
(703) 620-4990
**(703) 734-1387**
(703) 750-0930
**(703) 893-9474**
(703) 978-7561
**(707) 448-9055**

**(713) 233-7943**
(713) 693-8080
**(713) 977-7019**
(714) 449-5689
**(714) 463-0461**
(714) 495-6458
**(714) 526-3687**
(714) 537-7913
**(714) 565-0961**
(714) 571-5550
**(714) 582-9557**
(714) 730-1206
**(714) 739-0711**
(714) 751-1422
**(714) 772-8868**
(714) 898-1984

**(714) 952-2110**
(714) 962-7979
**(714) 963-7222**
(801) 375-7000
**(801) 466-1737**
(801) 753-6800
**(802) 748-9089**
(802) 879-4981
**(803) 270-5372**
(803) 270-5392
**(803) 279-5392**
(803) 771-0922
**(803) 772-1592**
(804) 340-5246
**(805) 484-9904**
(805) 527-9321
**(805) 682-7876**
(805) 964-4115
**(806) 355-5610**
(813) 223-7688
**(816) 523-9121**
(816) 531-1050
**(816) 861-7040**
(816) 931-3135
**(817) 855-3916**
(817) 855-3918
**(817) 923-0009**
(901) 276-8196
**(901) 362-2222**
(901) 761-4743
**(902) 794-8198**
(904) 243-1257
**(904) 243-8565**
(913) 362-6398
**(913) 764-1520**
(913) 782-5115
**(915) 584-5393**
(916) 393-4459
**(918) 224-5347**

$2.50 ($3.00 In Canada)

# DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

## FOR USERS OF SMALL COMPUTER SYSTEMS

Number 54 April 1981 Volume 6, Issue 4

**Z-80 Floppy Disk Test**
*Ray Duncan*

**Polynomial Evaluation in BASIC**
*R.A. Hoffman*

**MINCE — Not Just Another Editor**
*Barry A. Dobyns*

**A SLIC Label Program**
*Robert Hallem*

**Important Features of the PCNET Protocol**
*David C. Harris*

**...and more inside!**



P.O. Box E
Menlo Park
CA 94025

# DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Volume 6, Issue 4
Number 54
April 1981

# Contents

## Articles

**Z-80 Floppy Disk Test** . . . . . 8
*by Ray Duncan*

**Polynomial Evaluation in BASIC** . . . . . 14
*by R. A. Hoffman*

**Algorithms in FORTH** . . . . . 16
*by S. B. Bassett*

**A SLIC Label Program** . . . . . 17
*by Robert Hallem*

**A Cassette I/O Routine for BASIC Arrays** . . . . . 18
*by Arne Rohde*

**Important Features of the PCNET Protocol** . . . . 47
*by David C. Harris*

**MINCE: Not Just Another Editor** . . . . . 48
*by Barry A. Dobyns*

© 1981 by People's Computer Company unless otherwise noted on specific articles. All rights reserved.

## Departments

Letters . . . . . 4
Of Interest . . . . . 42
Electric Phone Book . . . . . 44
Software Review . . . . . 48
Club News . . . . . 52

**Publisher**
Ann Merchberger
**Associate Publisher**
Michael Madaj
**Editor**
Marlin Ouverson
**Contributing Editors**
Dave Caulkins
Dave Cortesi
**Art Director**
Clifford West
**Artist**
Barbara Ruzgerian
**Typesetter**
Renny Wiggins
**Proofreader**
Nancy Heubach
**Marketing Director**
Craig S. Harper
**Advertising Director**
Michael Madaj
**Circulation Manager**
Peter Clark
**Circulation Assistant**
Leah Dansby

*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025.

Subscription Rates & Writer's Guidelines on page 15.

# Editorial

Whenever we get much response to a particular issue of *DDJ*, we feel good. It shows what we are doing is significant, and that our readers recognize that fact.

The response to recent articles on FORTH (*DDJ* #51 and #52) has been gratifying. There have been both positive and negative comments, all indicating much interest in the FORTH language. From what I hear about the efforts of "FORTHians" (FORTHites? FORTHists?) to organize themselves and to promote their language, we will be hearing more from them in the future.

The other day, Doc Dobb poked his head into my office.

"What's the FORTH Interest Group had to say about Hal Gordon's articles?" he asked.

I stammered a couple of things under my breath.

"Well, speak up!" His blue eyes flashed. Never can get away with much when Doc's around.

"The FIG folks don't seem real thrilled . . ." I understated.

"Figures," he snorted. "Everything's got to be just so to please them FORTH people. I suppose they're still complainin' about the garbage on that stack, too."

"Uh-huh," I admitted.

"Well, there's only one thing to do!" he cried gleefully. "Let them tell it the way they think it should be."

"We are printing a few of their letters to the editor –"

"Not good enough. Give 'em a whole issue."

In a twinkling the door slammed shut and I heard Doc's footsteps head down the hall in search of another sail to trim.

Anyone who knows Doc realizes that it's no use arguing with him even if you want to, which I don't. So, FORTH fans and foes, here is your chance to grab pens and processors, paper and postage. Later this year, *DDJ* will publish an issue dedicated to FORTH. We don't expect every reader or group to agree with everything we publish; but like Doc is fond of saying, if it's grist for the mill, we'll grind it.



Marlin Ouverson
Editor

# Letters

## Back and FORTH

To the Editor:

H. T. Gordon's "What FORTH Is?" article is a truly amazing document. It could only have been written by an expert on microprocessor instruction sets who has admittedly "never used an assembler." I get the feeling that we are hearing a weird echo of the point of view of those who argued (circa 1959) that payrolls should continue to be written in machine language.

Reasoning by analogy of FORTH command word to CPU opcode, Gordon observes that the information density of an opcode (bit by bit) is high, while the character by character information of a FORTH word is (sometimes) low, and concludes that something is going to waste. This is not so much a useful insight as it is an example of the danger of reasoning by analogy. FORTH words are not opcodes. In particular, the "English-semantic" content of FORTH command words (like DROP) is critical to what little readability FORTH has, and the information density of a FORTH word is of no use to FORTH, while the bit by bit information density of an opcode is of very much use to its CPU, and the opcode mnemonic makes little difference to assembler program readability.

Changing the FORTH word DROP into such gobbledygook as ?SXY would be questionable practice even if done for a good reason. Proposing such a thing for the purpose of rendering a change to the 9th place "down" in the FORTH variable stack is simply incomprehensible to me. What purpose could be served by turning the stack into a kind of bastard RAM? And of course the code servicing all this new complexity is to be built into the FORTH kernel, so that in short order bringing up FORTH on a new machine would be as much work as building a new COBOL compiler.

Admittedly FORTH's all encompassing stack discipline, extending even to requiring 0 or 1 on TOS to control branching, must seem terribly rigid and confining to one accustomed to working directly with a machine's instruction set, with many instructions and a complex status register dedicated to branching. Things are not, however, always what they seem. Some things buy simplicity and don't confine much.

Confronted with all the power and freedom (*and* the complexity) of an assembly language, a very good programmer responds by producing 20 lines of debugged code per day, an average one makes 10. Given an interpretive HLL with nice extensions, a mediocre programmer does around 40 loc/day. Given the goal of solving some real problem for a user with real economic constraints, one might be forgiven for sacrificing some power in return for some simplicity.

[Parenthetically, Gordon's explanation of FORTH, before he embarked on questionable extensions, was possibly the best I've ever read. . . .]

David Carew
2925 Sage St. No. 4
Colorado Springs, CO 80907

Dear *Dr. Dobb's*,

Thinking about "What FORTH Is?" (*DDJ* 51 by H. T. Gordon) I think that one of the reasons why FORTH programs are said to be "cryptic" is the stinginess of your typesetter (and *BYTE* ones too) in using blanks. Furthermore, it's considered good programming style among FORTH programmers to use indentation. So, FIBO may look like

```
: FIBO  ( --- N1 N2 )
  0 1
  11 1 DO CR DUP . SWAP OVER +
       LOOP ;
```

the comment in the "(" , ")" showing the effect of FIBO on the stack.

And thinking about "good" FORTH: I'd rather do it this way

```
: FIBO  ( --- )
  0 1
  10 0 DO CR DUP . SWAP OVER +
       LOOP
  DROP DROP ;
```

Sincerely,
Klaus Schleisiek
126 Sussex St.
Jersey City, NJ 07302

## ZDM Note

Dear Mr. Ouverson,

I am writing to thank both *DDJ* and your reviewer, Mr. Gene Head, for the positive review of our software package, ZDM, which was published in your February 1981 issue.

I would also like to point out that the "very small bug" referred to in the review was corrected before any packages were released for sale. Our customers, therefore, need not be concerned.

Thank you again for devoting space to this review.

Sincerely,
R. F. Doolittle
RD SOFTWARE
1290 Monument St.
Pacific Palisades, CA 90272

## Atmospheric Perturbation?

Dear Sir,

I am delighted to learn from his article in *Dr. Dobb's Journal* that Mr. John B. Palmer derives "pleasure and satisfaction" when he relaxes of an evening hidden away with his Sinclair ZX80 computer. I mean that seriously, as I am tired of reading stupidly derogatory articles about an excellent, inexpensive, dedicated mini-machine.

I am sorry, however, to read again (and again and again) about certain alleged deficiencies which do not, in fact, detract from the excellent qualities and capabilities of the ZX80. To build a complete computer for less than $200 is quite an engineering feat in itself.

In the first place I should like to dispel all past and future nonsense about the size of the keyboard. If Mr. Palmer saw a ZX80 before purchasing one, or if he saw any Sinclair Research advertising, it is obvious that the entire machine is 6-1/2" x 8-1/2" overall, keyboard included. If that is too small for Mr. Palmer, then he'd best stay with his Apple II and not leap in where angels fear to tread. One can hardly expect to find a 6" x 16" keyboard on a machine dimensioned as above, not to mention a $50 keyboard in a $200 machine. So his grading of "2, weak" is really not germane to the subject.

As to Mr. Palmer's apparent lack of video quality, I have had no difficulties whatsoever in this respect, and I imagine Mr. Palmer's problem is due entirely to a "lowcost" third-rate "black-and-white" television set. Low-cost TV's are apt to be of the "hot chassis" variety and this could easily account for spurious radiation pickup from power lines. As to the power output of the modulator unit, I am familiar with this device; it outputs at least one volt peak-to-peak, which is more than adequate to drive anything except possibly a 1902 crystal set. As to altering the bandwidth of the output to operate within the constraints of a low quality TV, as a former engineer in Silicon Valley, Mr. Palmer should know that the minimum bandwidth requirement is determined by the line dot width and

number of lines displayed and thus is unalterable. In the case of the ZX80 this figure is deliberately minimal at 32 characters by 24 rows, being 768 characters per screen image, times 5 dots as the horizontal component of the matrix, times 7 dots as the vertical component. These characteristics should not demand a TV receiver bandwidth in excess of 3.5 MHz (the figure is explicitly calculable), which is usual for a B/W receiver of any quality at all. So, let us lay off abusing the ZX80 on the grounds of home-generated video faults, take it down to the local TV shop and try it on a respectable receiver before we start grading Video Display "1, Very poor." Quite ridiculous!

Integer BASIC means just what it says, so for the most part the redoubtable hobbyist had better just lay off of maths and stick to the fine art of writing pure logic programs where only simple additions and subtractions will suffice (will have to suffice). There is simply no way to obtain a remainder in any variable in Integer BASIC, as there is no subroutine in the ROM designed to write even single precision notation, and no room for it. I am confused by the allusions to the ZX80 graphics capability, which is "primitive" on one page, and "good" on the next, and worthless in any event in all machines (e.g., TRS-80, PET) which are not designed to write a full-screen dot matrix.

Speaking of the ZX80 ROM, the integer BASIC is only about 2K bytes long, and that is quite sufficient for a very powerful little interpreter (some may recall the excellent TSC 2.2K Micro BASIC). At least 1.5K of the ZX80 ROM is devoted to monitor functions, character generation, video and cassette control, editing utility, to mention a few. A full 8K BASIC chip at about $40 should be available shortly, and this will undoubtedly contain both single and double precision arithmetic routines. And for those with big ideas about RAM, there will be an 8K plug-in for the card edge connector, and this will not require any sculptural ability with a sharp knife.

As to the limitations of the cassette tape handling utilities in the ZX80 ROM, they work, and that is all Mr. Palmer needs to know. A new magazine called SYNCH [*sic*] is available now, devoted to ZX80 users and their machines, and I am sure that if Mr. Palmer wants to swap cassette tapes he will find, through SYNCH, a good number of willing pen-pals with whom to tape-swap. With regard to Kansas City Standard cassette tape format, this was not instigated by "many users got together in Kansas City." Mr. Wayne Green of *MICROCOMPUTING*, the president of SWTPC, and several other manufacturers and industrial people agreed to meet and establish a standard cassette format. Unquestionably KCS is a good format, but it does have certain serious limitations with regard to data transfer rates. Two or three companies since that famous meeting have managed to develop enhancement cards which are capable of decoding fractional wave-shapes, and I have one on another machine which will transfer data reliably at 3,600 bps and is fairly reliable at 7,200 bps, the latter being somewhat error prone.

To sum up, I repeat, I am happy that Mr. Palmer enjoys his little ZX80, but I think his grousing and grumbling about what it is not, what it has not, and what it was never intended to be or to have, is simply unnecessary "atmospheric perturbation." You take it the way it is and you enjoy it, or you just find some innocent youngster and sell it or give it to him. And the power of this little machine should never be underestimated. I thoroughly disagree with the statement that "After all, the real reasons anyone would buy a low-cost personal computer should not include anything very serious." There is a very great deal to be learned from the Sinclair ZX80, which is why I bought one (bringing my total to four, LP/6800, TRS-80 and Vector Graphic), and I expect to learn a great deal from the ZX80. I also think that it will offer Mr. Palmer a very great and beneficial learning opportunity, as a benefit added to the "pleasure and satisfaction" he now assures us he has with his little ZX80.

Most sincerely yours,
Langdon Proctor
707 South Orange Grove Ave. – F
Pasadena, CA 91105

■■

Engineers, scientists, systems programmers, and business owners... All these read *Dr. Dobb's Journal*. Reach them with cost-effective advertising. (415) 323-3111.

Since 1976, the most innovative magazine for the technically-sophisticated microcomputer user.

# Dr. Dobb's Journal

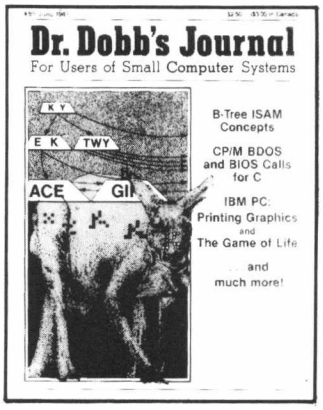
## For Users of Small Computer Systems




1976



1983




Have you been looking for a magazine that will provide you with serious articles on programming? *Dr. Dobb's Journal* can give you the technical depth you need to understand the latest software advances. We cover every microprocessor with articles on C, Forth, Unix, CP/M and much more. Our reputation for editorial quality is well known. Subscribe now, and discover what industry insiders have enjoyed since 1976.

Coming articles will cover:

- new language development
- 8- and 16-bit technology
- operating systems
- compilers, editors
- assemblers/cross-assemblers
- clever algorithms

**$25**

one year subscription
12 issues
saves $5 off the newsstand price

**YES!** Enter my subscription to *Dr. Dobb's Journal* and bill me later.

name

address

city state zip

**Mail to: DDJ, P.O. Box E., Menlo Park, CA 94025**

# Z~80 Floppy

The Z-80 Floppy Disk Test is a general purpose utility for testing or initializing a diskette. It requires a Z-80 microprocessor, at least 32 kbytes of random access memory, two flexible disk drives, and the Digital Research CP/M 2.x operating system. Program mode and area of diskette to be tested are controlled from the keyboard. Errors may be audited on the console or the line printer.

All console I/O is done using the standard CP/M function calls. For maximum test speed, access to protected system areas, and to avoid error interception by the CP/M operating system, disk I/O is performed directly by calls to the BIOS drivers. The test will not execute correctly under MP/M – don't even try it!

In the accompanying listing, the program is configured to test a soft-sectored, single-density disk with 77 tracks, 26 sectors per track, and 128 byte sectors. If you are using a different type of disk drive, the number of tracks and/or sectors per track may be specified during the initialization dialogue, or the program may be reassembled to modify the default configuration.

The program does not exhaustively check out the disk drive, and any errors detected by the program must be interpreted with care. Intermittent errors occurring throughout a broad range of disk addresses probably indicate a problem with the disk drive unit. Errors indicated at consistent addresses within adjacent tracks may result from a worn or damaged flexible disk. Error-free execution of the test certainly indicates an intact and usable diskette, and also is fairly good evidence of proper operation of your disk drive. Note that the program is rather insensitive to drive problems related to head movement, since the diskette is addressed in a smooth sequential manner.

## Using The Test Program

Put your system disk in drive A, boot up the CP/M operating system, and type:

```
A>dtst
```

The test program will load into memory and print the identifying message:

```
Z-80 Disk Test version 1.0
Copyright 1980 Laboratory Microsystems
```

If there is inadequate memory in the transient program area, or if the operating system is not CP/M version 2.0 or greater, the program will immediately print a warning message and exit. Otherwise, the test proceeds to the initialization dialogue. The test mode is set via a series of questions presented to the system console. Answers calling for an alphabetic character may be upper or lower case; a carriage return is not required. Answers calling for a number must be two digits, or one digit followed by a carriage return.

| QUERY | RESPONSES | EXPLANATION |
|---|---|---|
| Itemize errors? | Y or N | If answered Y, each error detected is audited on the selected device. If answered N, only the total error count is printed at the end of each pass. |
| Console or Printer? | C or P | If itemizing errors, select the device on which the itemized error listing will be printed. |


## by Ray Duncan

*Ray Duncan, Laboratory Microsystems, 4147 Beethoven Street, Los Angeles, CA 90066.*

# Disk Test

| QUERY | RESPONSES | EXPLANATION |
|---|---|---|
| Lock on read or write? | N, R, or W | N = no locks. R = lock on read mode (no data is written to the disk – useful for detecting any damaged areas on the media). W = lock on write (all data on the disk is destroyed – this may be used to erase a diskette by also locking on pattern). |
| Restore original data? | Y or N | Y = restore existing data on diskette after testing each sector. N = do not restore existing data. Test runs much faster when restore mode is not used. |
| Lock on data pattern? | Y or N | Y = user wishes to specify specific data to be used as the test pattern. N = random data pattern is generated by the test program. |
| Enter data pattern | hex 00-FF | If you answered Y to Lock On Pattern, enter the eight-bit data pattern to be used during disk testing. If you answered N, this question is bypassed. |
| Drive to be tested | B, C, or D | Specify disk drive to be used for diskette testing. In order to prevent accidental destruction of the system disk, the use of drive A is not allowed. |
| Test all tracks? | Y or N | Y = test all tracks (00-76) in sequence. N = user wishes to specify range of tracks (may be used to test only part of a diskette, or if disk drive is other than standard 8-inch single density). |
| First track to test | decimal 00-76 | If you answered N to Test All Tracks, specify here the number of the first track to test. |
| Last track to test | decimal 00-76 | If you answered N to Test All Tracks, specify here the number of the last track to test. Must be the same or greater than the number of the first track to test. |
| Test all sectors? | Y or N | Y = test all sectors (1-26) in sequence. N = user wishes to specify range of sectors (may be used to test only part of each track, or if disk drive is other than standard 8-inch single density). |
| First sector to test | decimal 01-26 | If you answered N to Test All Sectors, specify here the number of the first sector to test on each track. |

| QUERY | RESPONSES | EXPLANATION |
|---|---|---|
| Last sector to test | decimal 01-26 | If you answered N to Test All Sectors, specify here the number of the last sector to test on each track. Must be the same or greater than the number of the first sector to test. |
| How many test passes? | decimal 01-99 | Enter the number of test passes to be made on the diskette. At least one pass is always made even if you enter zero. |

After entry of the number of test passes, the program immediately begins executing. The program may be interrupted by pushing any key, however response may be delayed for several seconds depending on the speed of the processor and disk drive. Do not stop the test by resetting the computer system if you are using restore mode, or unpredictable amounts of data may be lost.

## Error Listing

Errors detected by the program may be itemized on the system console or the line printer. The pass, drive name, track, and sector where the error occurred are listed. All numbers are in decimal. The error type is designated at the right. In the explanations below, "hardware error" signifies an error status code returned from the low-level disk driver, and usually means a CRC fault or seek incomplete.

| ERROR TYPE | EXPLANATION |
|---|---|
| read error – original data | If running in restore mode, hardware read error noted as the original data was being loaded into memory. |
| write error – test data | Hardware error noted as the test data pattern was being written on the diskette. |
| read error – test data | Hardware error noted as the test data pattern was being read back from the diskette. |
| compare error – test data | Difference was found between the data pattern that was written to the diskette and the data that was read back. |
| write error – restore phase | Hardware error noted as the original data was being written back to the diskette. |
| read error – restore phase | Hardware error noted as the restored original data was being read back for verification. |
| data cannot be restored | Compare error found between the original data rewritten to the diskette and the same sectors read back for verification. |

## General Logic of Test Program

1. Check CP/M version and amount of user memory available. If they do not meet program requirements, exit with error message.
2. Configure test mode via interview with console operator.
3. Initialize current drive, track, and sector assignments.

4. If not restore mode, go to <5>, else read existing data on diskette for this track into memory and save it.

5. If this pass is locked on read, go to <6>, else generate a data pattern and write all selected sectors for this track.

6. If this pass is locked on write, go to <9>, else read all selected sectors for this track.

7. If locked on read or write, go to <9>, else compare the test data pattern that was written to the data that was read back from the disk.

8. If not restore mode, go to <9>, else write the original data back to the diskette for this track and verify it.

9. Advance to next track, if current track not greater than last track to test then go to <4>.

10. End of test pass, print pass number and total error count on selected devices.

11. If current pass not equal to last pass, go to <3>.

12. If user wishes to continue testing, go to <2>, else exit to CP/M.

## Major Program Subroutines

| | |
|---|---|
| RDBUF | Read up to one track from the disk into buffer specified by caller. Audit any hardware errors on the selected device, using descriptive message supplied by caller. |
| WTBUF | Write up to one track from the user-specified buffer to the disk. Audit any hardware errors on the specified device, using descriptive message supplied by caller. |
| BUFPAT | Fill a buffer with a data pattern to be used while testing the currently selected track. The user may specify a fixed eight-bit pattern, or the subroutine will generate a semi-random pattern from a combination of the refresh register contents and the memory address. |
| BUFCMP | Compare two buffers specified by caller. If any differences are found, print an error message on the selected device. |
| PERR | Subroutine called by all other subroutines to print an error message on the selected device. Current drive, track, sector and pass number are formatted together with a variable message supplied by caller. |

## Comments on Program Listing

The equates in lines 31-40 define the default drive, track, and sector limits. If you are not using a soft-sectored, single-density, eight-inch diskette drive, test convenience may be maximized by modifying these references and reassembling the program.

The reference "$dig" in line 45 defines the maximum number of digits to be accepted during console entries of hexadecimal or decimal integers in the test configuration sequence. This reference must be changed to a larger number if the test is to be used on disk drives having more than 99 tracks.

Lines 1072-1075 define the size of the disk I/O buffers. In the accompanying listing, these have been made twice as large as necessary so that the test can also be used on double-density drives. If the program will only be used on single-density disk

drives, you can minimize memory requirements by changing the source program as follows:

```
1072    buff1    equ    1000h
1073    buff2    equ    $bpt+buff1
1074    buff3    equ    $bpt+buff2
1075    buffend  equ    $bpt+buff3
```

Note that the starting memory address of each disk buffer should be evenly divisible by the value of $bps (bytes per sector).

## Program Areas of Interest to Users

To use the program on other types of disk drives without editing the source file and running an assembly, it may become necessary to change the first two locations given below. This may be done by reading in the COM file with DDT, patching the desired location, and SAVEing a new COM file.

0119H (8 bits) Sector skew flag. If byte=0, skew is not used. If byte=1, remap sector numbers as given in the sector translation table. On Z-80 systems at 4 mHz, skew is not needed and this byte may be left zero. On systems with slower processors, test speed may be improved by use of skew.

0121H (8 bits) Contains the number of digits to be accepted by the numeric input routines during the initialization dialogue, the value is two in the distributed version. If test is to run on a drive with greater than 99 tracks, change the value of this byte to three.

0122H (16 bits) Contains the address of the sector translation table. In the distributed program, the table is built with a skew factor of three, and is 26 bytes long.

## Test Availability on Machine Readable Media

For those readers who do not wish to retype the source text, Laboratory Microsystems will make available the .ASM, .COM, and .DOC files on a standard eight-inch, soft-sectored, single-density diskette. Send check or money order for $25.00 to 4147 Beethoven Street, Los Angeles, California 90066.

## TEST SAMPLE

```
A>DTST


Laboratory Microsystems Disk Test version 1.0

Itemize errors?  (Y/N)          > Y
Use console or printer? (C/P)   > C
Lock on read or write? (N/R/W)  > N
Restore original data? (Y/N)    > N
Lock on data pattern? (Y/N)     > N
Drive to be tested (B-D)        > B
Confirm: test drive B ? (Y/N)   > Y
Test all tracks? (Y/N)          > Y
Test all sectors? (Y/N)         > Y
How many test passes?           > 1

Beginning disk test – push any key to abort program.
Warning: user data will not be restored.
```

| Pass | Drive | Track | Sector | Error type |
|---|---|---|---|---|
| 0001 | B | 0062 | 0003 | read error – test pattern |
| 0001 | B | 0062 | 0003 | compare error – test pattern |
| 0001 | B | 0065 | 0023 | read error – test pattern |
| 0001 | B | 0065 | 0023 | compare error – test pattern |
| 0001 | B | 0066 | 0007 | read error – test pattern |
| 0001 | B | 0066 | 0022 | read error – test pattern |
| 0001 | B | 0066 | 0023 | read error – test pattern |
| 0001 | B | 0066 | 0007 | compare error – test pattern |
| 0001 | B | 0066 | 0022 | compare error – test pattern |
| 0001 | B | 0066 | 0023 | compare error – test pattern |
| 0001 | B | 0067 | 0007 | read error – test pattern |
| 0001 | B | 0067 | 0007 | compare error – test pattern |
| 0001 | B | 0068 | 0014 | read error – test pattern |
| 0001 | B | 0068 | 0014 | compare error – test pattern |
| 0001 | B | 0069 | 0012 | read error – test pattern |
| 0001 | B | 0069 | 0012 | compare error – test pattern |

Pass 0001 complete, 0016 errors.

Continue or exit test? (C/E) > E

Goodbye

A>
A>

(LISTING ON PAGE 22) ■■


## YOU CAN HELP YOUR COMMUNITY LEARN ABOUT COMPUTERS

**SUPPORT**

ComputerTown™

*ComputerTown*, a computer literacy project sponsored by People's Computer Company and later funded by the National Science Foundation, began operations in 1979. The project's ongoing goal is to design, publish and disseminate materials for the development of community-based computer literacy projects around the world.

**You too can help!** Your tax-deductible donations of new and used computer hardware and software will enable us to bring "hands-on" experience to people everywhere. These items can mean the difference between a person just hearing about computer technology and actually being able to sit down and use it. A *ComputerTown* can be set up in a public library, museum, school, business or a number of other places. What's needed, along with the information, resources and experience we provide, is just a few machines and some software to get started.

Already there are more than eighty *ComputerTowns* in the U.S., Canada, England and other countries. Your generous donations can help this number grow.

For more information about *ComputerTown*, and for details on how to make contributions, contact:

Fritzi Lareau, c/o *ComputerTown*
1263 El Camino Real, P.O. Box E, Menlo Park, CA 94025
**(415) 323-3111**

# Polynomial Evaluation in BASIC

by R.A. Hoffman

The need to evaluate polynomials occurs fairly often in scientific programming. For example, in order to plot a polynomial, one has to evaluate it on a large number of points; finding the "zero's" of a polynomial – say, by the Newton-Raphson method – also entails evaluation of the polynomial many times. Because many difficult functions can be closely approximated by a polynomial (power series expansion), this mathematical form appears often.

In addition to the above, it is often necessary to evaluate the derivatives of a polynomial, which, of course, are polynomials themselves. In general, a polynomial can be expressed as follows:

(1)

$$p_n(x)=a_nx^n+a_{n-1}x^{n-1}+\ldots+a_1x+a_0$$

Where x is the independent variable and the a's are constant coefficients. The highest power of the independent variable is the degree of the polynomial. $p_n(x)$ is, therefore, a polynomial of degree n, with the independent variable x. A short form for expressing $p_n(x)$ is:

(2)

$$p_n(x)=\sum_{i=0}^{n} a_ix^i$$

Representation of $p_n(x)$ in a computer can be accomplished in several ways. The simplest (but not necessarily the most efficient) uses a vector of dimension (n+1). We do this by stating:

80 DIM A(N)

This reserves (n+1) locations, in which we store the (n+1) coefficients of our polynomial, such that $a_0$ occupies A(0), $a_1$ occupies A(1) and so on to $a_n$ sitting in A(N).

If we wish to find the value of $p_n(x)$ when x assumes a specific value (say, x=7), we say that we wish to evaluate the polynomial at the point x=7 and write:

(3)

$$p_n(7)=a_n7^n+a_{n-1}7^{n-1}+\ldots+a_17+a_0$$

*R. A. Hoffman, 9 Greenway Ct., Brookline, MA 02146.*

Since the a's are simply numbers, using the computer to carry out the evaluation is – at least in principle – simple enough: first, we calculate 7 to the power of n; we then multiply it by $a_n$. Next, we calculate 7 to the power of (n-1), multiply the result by $a_{n-1}$ and add this to the previous result. We continue in this way, until the polynomial is finally evaluated.

There are two disadvantages to this evaluation scheme. It is easy to show that it involves n*(n+1)/2 multiplications, plus n additions (i.e., 36 multiplications and 8 additions for an 8th degree polynomial). This is both inaccurate (due to the accumulation of truncation and round-off errors) and slow. Because the calculation usually has to be carried out many times, and often for some or all of the derivatives as well, this process proves very unsatisfactory.

The algorithm to be presented below is much more efficient. Evaluation of the polynomial can be accomplished with n multiplications and n additions (8 each for an 8th degree polynomial). Better still, the derivatives of the polynomial follow naturally. They can be produced at the same time, with just over n*(n+1)/2 multiplications and additions for all.

The next section will present the mathematical derivation of this deceptively simple algorithm. Readers who are interested in results only will miss nothing by skipping it and picking up the action at the following section.

## Mathematical Derivation

We start by dividing our polynomial by (x-X1). Please note that (x-X1) happens to be a polynomial of degree one, with the independent variable x and $a_1=1$, $a_0=X1$ (X1 is, of course, a constant).

(4)

$$\frac{p_n(x)}{x-X1} = q_{n-1}(x)+\frac{R_0}{x-X1}$$

The quotient $q_{n-1}(x)$ is a polynomial whose degree is one less than that of $p_n(x)$. $R_0$ is a constant remainder, whose value depends on the coefficients of $p_n(x)$, as well as on the value X1. Rearranging terms:

(5)

$$p_n(x)=(x-X1)q_{n-1}(x)+R_0$$

Obviously, $p_n(X1)=R_0$, because the rest of the right side of the equation goes to zero when we evaluate at X1 by setting x=X1. It seems that if a neat way of evaluating $R_0$ exists, then the same way may be used to evaluate $p_n(X1)$.

Diverting our attention to polynomial $q_{n-1}(x)$:

(6)

$$q_{n-1}(x)=b_nx^{n-1}+b_{n-1}x^{n-2}+\ldots+b_2x+b_1$$

We can find the coefficients by actually carrying out the division called for by equation (4). Doing this, we find:

(7)

$$b_n = a_n$$
$$b_{n-1} = b_n*X1+a_{n-1}$$
$$b_{n-2} = b_{n-1}*X1+a_{n-2}$$
$$\vdots$$

with the general term being: $b_i = b_{i+1}*X1+a_i$, (i=n-1, n-2, . . ., 1). Please note that $b_n$ is the coefficient for the $x^{n-1}$ term, $b_{n-1}$ is the coefficient for the $x^{n-2}$ term and so forth down to $b_1$, which is the constant term of polynomial $q_{n-1}(x)$. It turns out that $b_0$, which may be calculated by equation (7), but is not a coefficient of $q_{n-1}(x)$ at all, is equal to $R_0$. The reader is invited to carry out the long hand division (4) and thus verify this (the operation is not difficult – just messy).

Now, let us divide polynomial $q_{n-1}(x)$ by the polynomial (x-X1). By analogy with previous results:

(8)

$$\frac{q_{n-1}(x)}{x-X1} = q_{n-2}(x)+\frac{R_1}{x-X1}$$

(9)

$$q_{n-1}(x) = (x-X1)q_{n-2}(x)+R_1$$

Just like before, $q_{n-2}(x)$ may be expressed by:

(10)

$$q_{n-2}(x)=c_nx^{n-2}+c_{n-1}x^{n-3}+\ldots+c_3x+c_2$$

With the general expression for the coefficients being:

(11)

$$c_n = b_n$$
$$c_i = c_{i+1}X1+b_i \quad (i=n-1, n-2, \ldots, 2)$$

Plugging the result into the original expression for $p_n(x)$, we get:

(12)

$$p_n(x) = (x-X1)^2 q_{n-2}(x) + (x-X1)R_1 + R_0$$

Taking the derivative of $p_n(x)$:

(13)

$$p'_n(x) = 2(x-X1)q_{n-2}(x) + (x-X1)^2 q'_{n-2}(x) + R_1$$

So that $p'_n(x)$ evaluated at X1: $p'_n(X1) = R_1$. But $R_1 = c_1$, as calculated by (11) above.

In a completely analogous manner it is possible to show that:

(14)

$$p''_n(X1) = 2*R_2$$
$$p'''_n(X1) = 6*R_3$$
.
.
.

with the general expression being: $p_n^{(i)}(X1) = i! * R_i$. In words: the i'th derivative of polynomial $p_n(x)$, evaluated at point X1, is equal to i factorial, multiplied by the i'th remainder of $p_n(x)$ divided by (x-X1). This statement is of a very general nature, if we agree that the zeroth derivative of a function is the function itself and that 0!=1.

## The BASIC Routine

In contrast with the frightening nature of the mathematical derivation, the BASIC routine which carries out the calculation is deceptively trivial. We start with three parameters which the routine receives from the calling program:

N – is the degree of the polynomial to be evaluated.

V – is a vector of dimension N (that is, contains N+1 elements, starting with V(0) and ending with V(N)), which contains the coefficients $a_0, a_1, \ldots, a_n$ of the polynomial.

X1 – point of evaluation.

The routine returns, in vector V, a series of values such that the value in V(i) is the value of the i'th derivative of the polynomial, evaluated at X1. V(0), of course, will contain the value of the polynomial itself, again – evaluated at X1.

Here's how it is done: to calculate the "b" coefficients of polynomial $q_{n-1}(X1)$, we do the following loop:

```
1060 FOR J=N-1 TO 0 STEP -1
1070 V(J)=V(J+1)*X1+V(J)
1080 NEXT J
```

The idea is to calculate the "b" coefficients *in situ* – that is, replace the content of V (which is the "a" coefficients of $p_n(x)$), with a new content (which is the "b" coefficients of $q_{n-1}(x)$).

Location V(N) contains $a_n$. Since $b_n = a_n$, we leave it alone and start the loop at location V(N-1), repeating the calculation all the way down to location V(0).

We now have $b_0$ sitting in location V(0). But $b_0 = R_0$, and $R_0$ is one of the values we are after. Therefore, we would like to repeat the calculation for the "c" coefficients of polynomial $q_{n-2}(x)$, starting again at location V(N-1), but going this time down to location V(1), leaving V(0) alone.

We will now have $R_0$ in location V(0) and $R_1$ in location V(1). We would like to repeat the calculation, this time for the "d" coefficients of polynomial $q_{n-3}(x)$, going from V(N-1) down to V(2). . . .

The pattern is now clear. . . . The routine looks like this:

```
1050 FOR I=0 TO N-1
1060    FOR J=N-1 TO I STEP -1
1070    V(J)=V(J+1)*X1+V(J)
1080    NEXT J
1090 NEXT I
```

After execution of this loop, V will contain the $R_i$ series. To calculate the actual derivatives, we need to multiply each term by the appropriate factorial, so:

```
1120 F=1
1130    FOR I=2 TO N
1140    F=F*I
1150    V(I)=V(I)*F
1160    NEXT I
```

In this loop, F assumes the value of I factorial in each iteration, by virtue of line 1140.

That's it!

Listing 1 shows a short program which demonstrates the polynomial evaluation subroutine (lines 1000 to 1170). It requests from the user a polynomial and an evaluation point, evaluates the polynomial and prints out the results. A sample run is shown in Listing 2.

■■

(LISTING ON PAGE 40)


DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

**Subscription Rates:** $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. dollars, drawn on a U.S. bank.

**Writer's Guidelines:** All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuacript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

**Donating Subscribers**

**Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives five one-year subscriptions).

**Contributing Subscribers:** Eric P. Hanson, J. G. Zeratsky, Mervin E. Frank, Edward W. Carr, Steven Fisher, Thomas H. Grohne, Richard Washburn, Les Hancock, G. V. Elkins, C. A. Forsberg, Daniel R. Lunsford, J. D. Griffith, Michael Slade, Thomas M. Schaefges, General Electric, Louis L. Odette, Robert M. Connors, William J. McEown, Robert C. Luckey, Kim Leary, Robert N. Stabler, Dewitt S. Brown, John B. Palmer, E. Glion Curtis, Roger Hanson, Jack Miller, Burks A. Smith, Transdata Corporation.

**Retaining Subscribers:** Mike Faithful, Osamu Shimizu, Scott B. Guthery, Robert L. Edens, David Dick.

**Sustaining Subscribers:** Gordon Eubanks, Jr. and Robert C. Birss, Ph.D.

**Foreign Distributors**

**UK & Europe:** LP Enterprises, 8/11 Cambridge House, Cambridge Road, Barking, Essex, 1G11 8NT, England (Tel. 01-591-6511, TLX 892395). Hofacker-Verlag, Tegernseer Strasse 18, D-8150 Holzkirchen, West Germany. Computerland/Computer Store AB, Box 7134, Kungsgatan 19, S-10387 Stockholmn, Sweden.

**Asia & Australia:** Electronic Concepts Pty Ltd., 55 Clarence Street, Sydney, NSW2000, Australia. Computer Store, POB 31-261, 22B Milford Road, Milford, Auckland 9, New Zealand. ASCII Publishing Co., Ltd., Aoyama Building 5F, 5-16-1 Minami Aoyama, Minato-ku, Tokyo 107, Japan.

# Algorithms in FORTH

## by S.B. Bassett

*Samuel B. Bassett, 1010 Ashbury #4, San Francisco, CA 94117.*

While I have no particular claim to being an expert in FORTH programming, I have been using Cap'n Software's FORTH systems for a couple of years, and have gotten familiar with the operation of figFORTH in its various incarnations.

When I thought about Algorithms, the first thing that occurred to me was Kernighan and Plauger's *Software Tools*, since the programming techniques they present are very general, and the functions they perform are quite useful. I got out my copy, re-read it, and found they lend themselves to programming in FORTH very well.

The first functions that K & P present are <getc> and <putc>, which get and put, respectively, one character from and to standard I/O devices. They ignore the details of where the characters come from, what their form is, and all other "details."

FORTH always works very close to the hardware level of the machine that it runs on, so a little attention to some of the details is in order.

First, I am going to assume that any character we are dealing with is an ASCII code right-justified in an 8-bit byte. Standard FORTH systems, to my knowledge, all make this assumption. The high-order bit is assumed to be zero for character-to-character compares.

Second, I am going to assume that the FORTH system normally talks to the outside world through a CRT terminal – all input and output come and go through it, unless directed to some specific peripheral.

FORTH provides two "primitive" words which link the system to machine-code routines which do the actual work of I/O:

KEY This word takes the next character received from the standard input, and pushes it onto the top of the working stack (S0, in Dr. Gordon's terminology). The stack is 16 bits wide, so the character is put into the low-order byte.

EMIT Pops the low-order 7 bits from the top of the working stack, and puts it to the standard output device.

FORTH usually assumes that programs and data are stored on a disk drive, and provides a word to do disk I/O:

R/W Reads or writes one sector of data from or to the disk. R/W takes three parameters from the stack:

Top (TOS) = a Boolean flag, with 0 meaning "Write" and 1 meaning "Read."

2nd = the number of the block to be fetched/written.

3rd = the starting address of the text buffer into/out of which data is to be read/written.

I will specify parameters for FORTH words like this:

| 3rd addr | 2nd blk# | TOS f | Word R/W (or "---") |
|---|---|---|---|

where the value on the top of the working stack is closest to the right, and values deeper in the stack are successively further to the left.

Let's assume we have gotten a buffer full of text from the disk, using R/W, and that it is located at address B000 Hex, and we want to move it to B400. This is the job which K & P describe for <getc> and <putc>.

First: HEX

To put the FORTH system into Hexadecimal mode, so that all numbers are interpreted as Hex and not Decimal.

The second step is to define variables to hold the "from" and "to" addresses and a pointer variable:

```
VARIABLE INPUT.BUFFER
VARIABLE OUTPUT.BUFFER
VARIABLE PTR
```

(79-Standard FORTH, unlike earlier models, does not provide for setting the initial value of a variable. Earlier versions would have entered: B000 VARIABLE INPUT.BUFFER. Notice the length of the name – 79-Standard FORTH accepts names of up to 31 characters.)

VARIABLES return the address of the area where their values are stored when their names are called. The address is put on the top of the stack.

Next, of course, we need to initialize the variables:

```
B000 INPUT.BUFFER !
B400 OUTPUT.BUFFER !
0 PTR !
```

The word "!" (pronounced "store") stores the value on the TOS into the address which is in the second position on the stack. So here we have stored B000 in INPUT.BUFFER and B400 in OUTPUT.-BUFFER without having to worry much about where the value is located.

Next we will define GETC:

```
: GETC
   INPUT.BUFFER @
   PTR @ +
   C@
  ;
```

A colon (":") is the FORTH word which starts a definition of a new word. A semicolon ends the definition.

GETC is, of course, the name of the new word.

INPUT.BUFFER puts the address of the Input Buffer on the TOS.

@ (pronounced "fetch") is the FORTH word which takes an address off the top of the working stack, and replaces it with whatever was stored at that address – the actual address of the Input Buffer.

PTR puts the address of the Pointer in the top of the stack.

@ pops the address of the pointer off the working stack, and pushes the contents of that address back on the stack – the offset into the buffers.

+ (pronounced "plus") takes the two top numbers on the stack, adds them, and puts the result back on the stack.

(At this point, we have put the address of the Input Buffer on the top of the stack, then pushed the address of the pointer on top of it, replaced the pointer address with the pointer's contents, and added the contents of the pointer to the beginning address of the Input Buffer – in effect, doing an offset from the beginning of the buffer.)

C@ We "fetch" the low-order byte of the offset address in the Buffer, and push it on the top of the stack.

PUTC is defined very similarly:

```
: PUTC
   OUTPUT.BUFFER @
   PTR @ +
   C!
  ;
```

Notice that I have been very careful to leave spaces between words in the definitions – spaces are just as significant in FORTH as they are in English – the dictionary look-up routine uses spaces as delimiters, and will not dig PTR, @ and + out of "PTR@+". It will simply reject it as an error.

C! ("c-store") pops an address off the TOS, and stores the low-order byte of the number at 2nd in that address.

(Born-again FORTH programmers will now scream in chorus: "But there are

LOTS of better ways to do that!!!" This is indeed true, but I am deliberately making this algorithm follow K & P's Ratfor examples.)

Having now defined GETC and PUTC, it is time to write the COPY which appears on page eight of *Software Tools* in FORTH, but first we need to define EOF:

```
FF CONSTANT EOF
```

CONSTANT returns the value stored in it when it is invoked. Every time we type "EOF", the CONSTANT routine will push FF onto the TOS.

```
: COPY
   BEGIN
    GETC
    DUP EOF = 0=
    PTR @ 400 <
    AND
   WHILE
    PUTC
    1 PTR +!
   REPEAT
;
```

BEGIN . . . WHILE . . . REPEAT is the FORTH equivalent of the <while> construct in Ratfor. It works by testing a flag at the WHILE. If the flag is True (nonzero in most FORTHs), the code between WHILE and REPEAT is executed, and the program loops back to just after the BEGIN. If the flag is False (zero), then the program exits the loop, and executes whatever code is after the repeat.

In this case, we GETC to put a value on the top of the stack.

DUP duplicates whatever is on top of the stack, so that TOS and 2nd are now the same. We duplicate, so that we can test the value without losing it – this is a very common operation in FORTH.

EOF pushes the value "FF" on the stack.

= ("equals") pops the two top values off the stack, and tests them to see if they are equal, and pushes a boolean value back on the stack. If they are equal, it leaves a "1", and if non, a "0".

0= ("zero-equals" or "not") pushes zero on the stack, pops the two top numbers, compares them, and pushes "1" (True) if the 2nd number was 0, and pushes "0" (False) otherwise.

(The next line checks for buffer overflow – FORTH does not provide this automatically, and we must provide it for ourselves.)

PTR @ gets the current value stored

*(Continued on page 20)*

# A SLIC Label Program

## by Robert Hallem

*Robert Hallem, RTG Data Systems, 1003 Wilshire Blvd., Suite 202, Santa Monica, CA 90401.*

I was finishing a fancy mailing list program, with keyed-access files, CRT masks and so forth, when I began to wonder if this might not be overkill. All I really needed was a sequential file of names and addresses, plus some "selection codes" to categorize each entry. Then if I had a program to read the entire file and print labels for those entries with specified codes – that would do the job!

The accompanying listing shows this simple mailing list label printer. The name and address file is created with a standard text editor (CP/M's ED, in my case). Each entry consists of a name line (last, comma, first), one or two address lines, a city-and-state line, a zip-code line, and a selection code line. The selection code line must begin with a space, and it may contain up to ten code numbers separated by spaces. Entries are separated by one or more "empty" lines (containing return and line feed, under CP/M).

The use of a sequential file and an existing editor for file maintenance means you aren't stuck with a fixed record layout. For instance, if you must switch to 9-digit zip codes, you just edit the file to add the extra digits. You can also store additional information after the code line, say a phone number; additional lines are ignored by the program if they begin with a space. The main formatting restriction is that you must include the code line, since the form assumes the line before the code line is the zip code, and the line before that is the city and state.

Another advantage of this free-format sequential file is space-saving: short entries take up less space than long ones. With the usual fixed-length-record programs, all entries must have enough room to accommodate the longest name and address. And best of all for those without disk storage: even a TRS-80 Model I with cassette or stringy floppy can use this program.

The major disadvantage, clearly, is the lack of order within the file. Labels are printed in the order they are found in the file, not sorted alphabetically by name or numerically by zip. To find a particular name you must use your editor's "find" or "search" command. But if you don't need sorted output, this is a good way to go.

The program is written in SLIC, a structured language I wrote. SLIC's big advantage is that it has both an interpreter for modification and debugging, and a compiler for speed and compactness. The interpreter and compiler are largely (but not yet entirely) compatible. This program was debugged with the interpreter, then compiled (without changes) for speed. The compiled version uses less than 7.5K bytes of memory, including the 4K run-time system, plus some stack space.

SLIC syntax is quite similar to C and RATFOR, so translation shouldn't be too difficult. SLIC is fully documented in "Structured Programming in SLIC," available from RTG Data Systems for $10. However, I'll go over the major differences from C for those who may want to do the translation.

In SLIC, the symbol "@" precedes each function name, so they stand out clearly. Functions may normally alter their arguments in SLIC, as in FORTRAN and RATFOR. This eliminates much of the confusion caused by the extensive use of pointers in C. However, an explicit call-by-value is possible by surrounding the argument variable (in the call) with parentheses.

I/O uses built-in functions, and the "ask" and "type" statements (which are somewhat like INPUT and PRINT in BASIC). At any given moment there is a current input device and a current output device. The default values are 0 (the keyboard) for input and 1 (the video display) for output. They are changed with statements like "type #3;" to send output to the printer (unit 3), or "ask #6;" to read from the file open on unit 6. Function "@ask" reads a line and "@type" outputs a line.

SLIC's "for" loop has the condition to be tested in the third position, not the second as in C. This was not just to be difficult – it's a bit easier for an interpreter to scan. (Compilers don't care.) Also, the SLIC "do" loop is like FORTRAN's. C's "do-while" is SLIC's "repeat-until".

Of course, there are many other dif-

*(Continued on page 20)*

# A Cassette I/O Routine for BASIC Arrays

by Arne Rohde

*Arne Rohde, Pilevej 31, 7600 Struer, Denmark.*

One of the main disadvantages of using cassette tapes as the medium for storing programs and data is the speed with which data can be read and written. The TRS-80 Level II records data on tape at about 500 baud, and can therefore transfer data at about 60 bytes per second. For each block written to tape there is a preamble consisting of 256 bytes of binary zeroes, requiring about 4 seconds to write or read. Since data blocks in BASIC may be a maximum of 256 bytes in length, the preamble will always require at least half of the total data tape processing time. For very short data blocks, the proportion of time spent on the preamble may well be over 90%.

As an example, a two-dimensional array is to be stored on tape. The array consists of 150 x 5 elements, or 750 integer values altogether. Since integer values require two bytes each, the array will require 1500 bytes plus a few for the definition of the array. If 5 elements are written in each tape block, then 150 blocks will require about 600 seconds, or 10 minutes to write. The format for the data portion of each block is the same as it would be for a screen display, in other words expanded to decimal form with room for a sign and a trailing blank for each value, but with a comma as delimiter between the values. The 5 values will therefore require from 3 to 7 bytes each, plus 4 commas, giving between 19 and 39 bytes depending on the number of significant digits in each value. Assuming an average of 30 bytes per block, or 4500 data bytes in all, the data alone will require about 75 seconds to write. The preambles take almost 90% of the total tape time of 675 seconds, assuming no pauses are needed between the blocks. If the data could be written in its original form with the 1500 data bytes written as one large block, then only 4 seconds would be required for preamble and about 25 seconds for data, giving 29 seconds in all. The standard TRS-80 method takes more than 23 times as much time as our hypothetical method would take. Savings of this dimension can at least be considered significant!

When string variables are written to tape with the PRINT statement, the variables must not contain quotes, commas, or colons, since these will be treated as delimiters when the tape is read, nor may they contain leading spaces since these will be removed when they are read from tape. All except the quotes can be accepted if the variable is extended with a leading quote (ASCII value 34) before the string is written to tape. A trailing quote is not required, unless other data values follow the string in the same data block. The final value in a block will be terminated with a carriage return control character. The method can be illustrated as follows:

```
10 Q$=CHR$(34)
...
250 PRINT #-1,Q$;A$
```

where A$ is the string which may contain commas, colons, or leading spaces. The remaining problem of the quote could also be solved if it were possible to write the strings to tape without the need for delimiters between the variables.

Another problem with the normal tape I/O method is that no checksum or any other control character is written in the block, resulting in I/O errors being accepted without any possibility of checking the validity of the data. Since tape is a relatively unreliable medium, no I/O should be performed without any means for a validity check. I realize that BASIC programs are written without a checksum, but I also consider this to be a design flaw!

These factors which had long annoyed me and restricted the type of programs I could reasonably run on a tape-based system, motivated me to find a solution and write a routine which could cure most of the ills. I realize that one or two of the utility programs advertised in the microcomputer magazines state that they have array I/O routines, but living in Denmark naturally restricts one in the possibility of trying and buying software. I have never tried any of these utilities, and have no idea how they implement these functions. Also, none of the magazines I have read has contained a routine which could be used.

In order to construct a routine which will read or write arrays in BASIC, it is necessary to know how array data is stored in a program. Immediately following the source program in memory, BASIC stores information concerning all variables and arrays defined or referenced in the program. All simple variables are stored first, followed by array variables. There are three pointers in reserved memory which are useful for referencing this area. The first, at 40F9 and 40FA, points to the start of the variable storage area. The next, at 40FB and 40FC, points to the end of the simple variables and the beginning of the array area. The address at 40FD and 40FE is the end address of the array area. A consequence of this storage method is that arrays must be moved in memory when a new simple variable is defined or referenced.

Since we will be concerned only with arrays, we shall look at the method for storing arrays. Numeric and string arrays are stored differently, although the methods are very similar. The first byte of the array area defines the type of array variable – integer, single or double precision real, or string. This definition consists of the number of bytes needed for storing each element in the array. The values are 2 for integer, 3 for string, 4 for single precision, and 8 for double precision. The next two bytes are used for the array name, without any type qualifier (%, #, !, or $) since this is already implicit in the value of the first byte. The name is stored with the least-significant byte first, with zero if a one-character name is used. The next two bytes contain the length of the area required for storing the array, excluding the first 5 bytes of the definition area. The next byte is the number of array dimentions, from 1 to 255. For each dimension, two bytes are used to indicate the number of elements in that dimension. This value will be one greater than that used in the DIM statement. The following area contains 2, 3, 4, or 8 bytes for each of the elements in the array. For numeric arrays these bytes contain the actual values of each variable, for string arrays the first byte contains the length of the string and the next two bytes the address of the first byte of the string. The array definition for an array defined as DIM A%(2) would be as follows:

02 0041 0900 01 0300 0000 0000 0000

where 02 is the type (integer), 0041the name (A), 0900 the remaining length of the definition (9), 0300 the number of elements in the first dimension (3, since subscripts 0, 1, and 2 can be used), and the following zeroes are the values of each element on the array.

For string values, the address pointer can point to a position in the string area (defined with the CLEAR n statement) or to a point within the BASIC program. If

a string is assigned a value with a READ statement or assigned to a literal then the pointer will point to the original string definition in the program and the string will therefore not take up space in the string area. However if a string array is written to tape and then read into the program from tape, then every string in the array will be stored in the string area. The need for string storage may thus often be considerably greater when strings are read in from data tapes. This factor must be taken into account when tape storage is used for string arrays, or for other string variables.

The array I/O routine as implemented is written in machine language, and is resident in the top of memory. Calls to the routine are through the USR interface, with a single POKE required when changing from a read to a write and vice versa. Read and write functions are contained in the same program, requiring less than 300 bytes in all. It can handle all types of arrays, with any number of dimensions in each array.

Since the USR call only allows the passing of a single 2-byte parameter to the called routine, an extra POKE must be made to distinguish between a read and a write function. The parameter passed to the routine is the address of an element in the array to be read or written. Arrays to be read must therefore have been defined, either by DIM or by reference, before the read routine is called. The routine also returns a value to the calling program indicating whether the read or write was successful or not. The definition of the array to be read must be identical to the definition of the array written to tape as concerns variable type, number of dimensions, and number of elements in each dimension. Partial arrays cannot be read or written, and a separate block is used on tape for each array.

The routine is constructed so that the entry points for write and read are at 7F00 and 7F08 respectively for a 16K system. This gives the commands POKE 16526,0 and POKE 16526,8 before each write or read if the MSB of the USR address is left unchanged during execution. A typical sequence for writing an array to tape would be as follows:

```
220 PRINT "MOUNT TAPE WITH
    RECORD, HIT ANY KEY"
230 IF INKEY$="" THEN 230
240 POKE 16526,0:IF
    USR(VARPTR(array element))
    PRINT "ARRAY WRITE
    ERROR":STOP
```

where (array element) is a reference to any element in the array to be written, typically element zero. The value returned by the routine is -1 if an error is found, else zero. The IF statement can thus be coded as shown, recalling that -1 is the value for true and 0 the value for false. The only possible error for a write is that the parameter passed to the routine does not point to a byte within an array definition. The cassette used for writing will be the on-board cassette, although this could easily be changed if necessary.

The procedure for reading a numeric array is very similar to that for writing an array. The following could be used:

```
320 PRINT "MOUNT TAPE WITH
    PLAY, HIT ANY KEY"
330 IF INKEY$="" THEN 330
340 POKE 16526,8:IF
    USR(VARPTR(array element))
    PRINT "ARRAY READ
    ERROR":STOP
```

where the parameters used are as for write. The procedure for reading string arrays is more complicated, since the maximum space possible should be made available in the string area before the array is read. The read routine does not have time during the read process to initiate and wait for a string compress, and this should therefore be done by the BASIC program before the read routine is called. The maximum area is made available by assigning each element in the array to be read to a null string, and then calling the FRE function to force a string compress. The reassignment is only required if the array has been used previously. The initial process could be coded as follows:

```
300 FOR I%=0 TO AM%: A$(I%)=
    "":NEXT I%
310 IF FRE("")< minimum PRINT
    "INSUFFICIENT STRING
    AREA":STOP
```

for a single-dimension string array A$, where AM% is the DIM value used for the array. The lines following 310 are identical to those used for a numeric array read.

A number of error conditions are possible when reading arrays:

a) The value passed to the routine does not point to an array element.
b) The array definitions on tape and in memory are different. Only the names may differ.
c) A checksum error occurred during the read.
d) There was insufficient string storage area available.

Since the routine uses the Level II ROM routines for tape I/O, two asterisks will appear in the upper right corner of the screen when a block header has been found. If the tape read is successful, these asterisks will be cleared from the screen before returning to the BASIC program. Any data contained in these positions will be lost. No blinking of the asterisks will occur, even for very long arrays.

Since I did not have an editor/assembler at the time the routine was written, the routine was hand-coded and converted to machine code. It is entered into memory with a BASIC program containing the hexadecimal values of the bytes to be entered. The routine can easily be relocated, with only a few minor changes in the program. The entry program calculates the relative jump values and the addresses of the branch points used in the program. A similar concept has been used in a MACROPOKE monitor by Larry Suter (*80-Microcomputing*, April 1980), although the one shown here is easier to use when segments are to be entered in different portions of memory. The program will check whether sufficient storage has been reserved for machine-language code, and will reserve space if none has been reserved. It is therefore not necessary to answer the MEMORY SIZE message with any specific value.

A few comments on the format of the machine code DATA entries may be needed. Each DATA statement contains 3 string values, each of which may be null. The first is the hexadecimal address where the code on the line is to be entered. If null, the code will follow immediately after the previous line. The address should be entered for the first line of each segment. The second value is an optional one-character label which can be referenced in the code portion of the routine. All ASCII characters are valid labels, except those recognized by BASIC as delimiters (quote, comma, and colon). The third string is the actual hexadecimal code to be entered, with the following extension. Spaces may be used to separate bytes to improve legibility, as in the listing where a space is used to separate each of the instructions. Each byte must be written as two characters, leading zeroes may not be omitted. If the first

character is an R (for relative jump value) then the following character must be the label of the byte to be jumped to. If the first character is a W (for word address) then the following character is again a label. The next two characters must be hexadecimal, and are interpreted as a one-byte two's complement displacement from the label. WL00 would be the adress of the label L, WL05 the address + 5, and WLFB the address - 5. The DATA statements are read in two passes, the first to resolve the label values, and the second to POKE the values into memory. A listing of each label used is given in the first pass for control purposes. Error messages may be listed for various error conditions.

The assembler language routine may also need a few comments. The labels WRTARY and RDARY are entry points for array write and read respectively. Each of these calls the routine ARAYA which finds the start and end addresses of the array to be written or read. The cassette will also be switched on, ready for write or read commands. In the read routine, the first 6 bytes are read from tape; these bytes are the first 6 array definition bytes. All except the array name portion is compared with the definition of the target array definition. If these are equal, the remaining array bytes are read directly into the array area. Numeric arrays only need the checksum to be read after all the data bytes have been read. String arrays require a reference to the length of each string and must update the string address pointers to the new address where each string will be stored. BYPLN will bypass the dimension length codes in the array definition, and point to the definition of the first element. Address 40A0H contains the address of the first byte reserved for string storage (the lowest address) and 40D6H one less than the last byte used for previous strings. Strings are stored starting at the highest available address, so the pointer in 40D6H must be reduced for each string stored. If the string length was zero, then no bytes are read from tape for that string, otherwise up to 255 characters will be read for each string, until all strings in the array have been read. The checksum is then read and checked, and the two asterisks cleared from the screen.

The array write routine writes the first 6 definition bytes followed by a pause to allow time during input for the comparison. The remaining part of the array area is then written. For string arrays this is followed by the string data for each string with non-zero length. A pause is left before each string to allow for address calculation on input. Finally the checksum is calculated and written, and the cassette switched off before returning to the calling program.

The routine ARAYA is used to step through the array definitions until the required array has been found. The routine called by RST 18H is used several times to compare the values in DE and HL, but unfortunately the value in HL is not saved by this routine.

The program should be easy to relocate to another address if required, since the BASIC program takes care of all the instructions with direct address references. The values which will have to be changed are the ones which check for and set the memory size, and the two address values in the DATA statements for the two code segments. The USR call address will automatically be changed, as will all the absolute JMP and CALL instructions. The routine is not intended for use with DOS systems, although it could be modified to cater for this. The usual 408EH address vector for USR has been moved, so DEFUSR could be used instead, or the two addresses could be set in the BASIC routine. Also, interrupts should be disabled during the tape I/O process.

The enclosed listings show the BASIC program used to enter the routine into memory, and a listing of the program in Z-80 assembler format. Unfortunately the TRS-80 assembler does not allow labels to be defined on separate lines, so they have been defined with an EQU $ statement. I prefer labels on separate lines for easing insertions and deletions at a later stage.

Hopefully, these routines can make cassette tapes a realistic medium for storing data in BASIC programs.

■■

## Algorithms in FORTH

*(Continued from page 17)*

in the pointer.

400 < compares that value to 400 and leaves a True if it was 3FF or less.

AND pops the two top numbers off the stack, does a bit-by-bit Logical AND on them, and pushes the result back on the stack.

(At this point, the Top Of the Stack has a boolean value in it – if the character returned by GETC was not equal to EOF, and the contents of PTR are less than 400 Hex, it is a "1", or True. Otherwise it is "0", False.)

WHILE tests the top of the stack, and if it finds a "1", it goes to PUTC, below. If WHILE finds a "0", the program ends, and whatever is after REPEAT is executed.

PUTC takes the value returned by GETC, and puts it in OUTPUT.BUFFER at the offset pointed to by PTR.

1 PTR +! adds 1 to the contents of PTR – it increments the pointer, so that it points to the next character in the buffer.

The program now loops back to GETC to fetch another byte for testing.

If nothing follows REPEAT when the test at WHILE branches out of the copy program, the FORTH system returns control to the terminal, prints out:

OK

and waits for further instructions.

Next issue, I will show how to code K&P's Counting programs from pages 12-18 of *Software Tools*.

■■

## A SLIC Label Program

*(Continued from page 17)*

ferences between SLIC and C. I've just mentioned the ones I feel are most likely to confuse the C programmer reading a SLIC program.

For those who appreciate the advantages of a structured language interpreter, SLIC is available in several forms: A DISK SLIC interpreter and SLIC compiler for CP/M systems. TAPE SLIC for the TRS-80 (Model I, Level II, 16K) is on cassette. That's the interpreter only, no compiler. Write to me for information. The TAPE SLIC interpreter is also available on stringy floppy wafer, directly from Exatron, 181 Commercial St., Sunnyvale, CA 95051.

As for that fancy mailing list program with keyed-access files, I finished it anyway so I could test the keyed-access routines. Works pretty well, too, but it really gobbles up disk space with those fixed-length records!

**(LISTING ON PAGE 36)** ■■

Since 1976, the most innovative magazine for the technically-sophisticated microcomputer user.

# Dr. Dobb's Journal

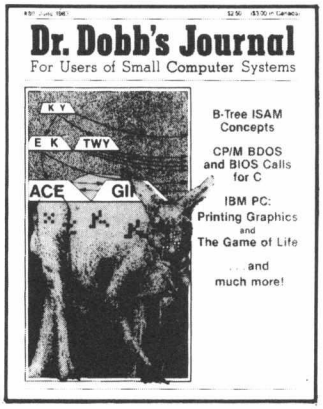
## For Users of Small Computer Systems




1976



1983




Have you been looking for a magazine that will provide you with serious articles on programming? *Dr. Dobb's Journal* can give you the technical depth you need to understand the latest software advances. We cover every microprocessor with articles on C, Forth, Unix, CP/M and much more. Our reputation for editorial quality is well known. Subscribe now, and discover what industry insiders have enjoyed since 1976.

Coming articles will cover:

- new language development
- 8- and 16-bit technology
- operating systems
- compilers, editors
- assemblers/cross-assemblers
- clever algorithms

**$25**

one year subscription
12 issues
saves $5 off the newsstand price

**YES!** Enter my subscription to *Dr. Dobb's Journal* and bill me later.

name

address

city state zip

**Mail to: DDJ, P.O. Box E., Menlo Park, CA 94025**

# Z-80 Floppy Disk

```
                        0002 ; Floppy Disk Test for Z-80 CP/M Systems
                        0003 ;
                        0004 ; version 1.0  16 December 1980
                        0005 ;
                        0006 ; The purchaser may freely create any number
                        0007 ; of copies of this program on magnetic media
                        0008 ; as necessary to support his computer system(s).
                        0009 ; Further resale of this program is prohibited.
                        0010 ;
                        0011 ; Copyright (c) 1980 by
                        0012 ; Ray Duncan
                        0013 ; Laboratory Microsystems
                        0014 ; 4147 Beethoven Street
                        0015 ; Los Angeles, CA 90066
                        0016 ;
0000                    0017          org     100h
                        0018 ;
      (0005)            0019 cpm      equ     5         ;references to
      (0000)            0020 wboot    equ     0         ;operating system
                        0021 ;
                        0022 ;                          references to ASCII char
      (000D)            0023 cr       equ     0dh
      (000A)            0024 lf       equ     0ah
      (000C)            0025 ff       equ     0ch
      (0009)            0026 tab      equ     09h
                        0027 ;
                        0028 ;                          parameters for disk
                        0029 ;                          (supplied as single
                        0030 ;                          density, soft sectored)
      (0001)            0031 $drvf    equ     1         ;first drive to allow
                        0032                            ;testing (0=A, 1=B, etc)
      (0003)            0033 $drvl    equ     3         ;last drive to allow
                        0034                            ;testing
      (0000)            0035 $trkf    equ     0         ;first track
      (004C)            0036 $trkl    equ     76        ;last track
      (0001)            0037 $secf    equ     1         ;first sector
      (001A)            0038 $secl    equ     26        ;last sector
      (0080)            0039 $bps     equ     128       ;bytes per sector
      (0D00)            0040 $bpt     equ     $bps*$secl ;bytes per track
                        0041 ;
                        0042                            ;number of digits to
                        0043                            ;accept in track and
                        0044                            ;sector assignments
      (0002)            0045 $dig     equ     2         ;(should be set larger
                        0046                            ; for devices having
                        0047                            ; track or sector
                        0048                            ; numbers >99 )
                        0049 ;
                        0050 ;                          program identification
      (0001)            0051 $ver     equ     1         ;version
      (0000)            0052 $rev     equ     0         ;revision
                        0053 ;
0100  C32401            0054          jp      dtst      ;enter from CP/M

                        0056 ; global variables for use by all routines
                        0057 ;
0103  0000              0058 pass     dw      0         ;current pass number
                        0115                            ;test run
                        0116                            ;
0121  02                0117 digits   db      $dig      ;maximum number of digits
                        0118                            ;to be accepted during
                        0119                            ;decimal or hexadecimal
                        0120                            ;numeric input.
                        0121                            ;
0122  6905              0122 xtran    dw      sectrb    ;address of sector
                        0123                            ;translation table

                        0125 ;
                        0126 ; disk test --- main control
                        0127 ;
      (0124)            0128 dtst     equ     $         ;entry from CP/M
0124  118305            0129          ld      de,dtsta  ;print program title
0127  0E09              0130          ld      c,9
0129  CD0500            0131          call    cpm
012C  2A0600            0132          ld      hl,(cpm+1)
012F  11005E            0133          ld      de,buffend
0132  B7                0134          or      a,a       ;make sure enough user
0133  ED52              0135          sbc     hl,de     ;memory to execute test
0135  300B              0136          jr      nc,dtst01
0137  11B907            0137          ld      de,dtsts  ;not enough memory,
013A  0E09              0138          ld      c,9       ;print warning and exit.
013C  CD0500            0139          call    cpm
013F  C30000            0140          jp      wboot
0142  0E0C              0141 dtst01   ld      c,12      ;check CP/M version
0144  CD0500            0142          call    cpm
0147  7D                0143          ld      a,l       ;make sure 2.x
0148  E6F0              0144          and     0f0h
014A  FE20              0145          cp      20h
014C  280B              0146          jr      z,dtst02
014E  11A008            0147          ld      de,dtstz  ;not CP/M 2.x, print
0151  0E09              0148          ld      c,9       ;error message and exit
0153  CD0500            0149          call    cpm
0156  C30000            0150          jp      wboot
0159  AF                0151 dtst02   xor     a,a       ;initialize variables
015A  321B01            0152          ld      (bypass),a
015D  321901            0153          ld      (skew),a
0160  321A01            0154          ld      (list),a
0163  321B01            0155          ld      (lockio),a
0166  321C01            0156          ld      (restor),a
0169  321D01            0157          ld      (lockpt),a
016C  320301            0158          ld      (pass),a
016F  320401            0159          ld      (pass+1),a
0172  320501            0160          ld      (errors),a
0175  320601            0161          ld      (errors+1),a
                        0162                            ;now set up test
                        0163                            ;configuration
0178  11CC05            0164          ld      de,dtstb
017B  CD8309            0165          call    getyn     ;itemize errors?
017E  FE79              0166          cp      'y'
0180  2807              0167          jr      z,dtst03  ;yes
0182  3E01              0168          ld      a,1       ;no
0184  321B01            0169          ld      (bypass),a
0187  1816              0170          jr      dtst04    ;skip query for output
```

```
0105  0000   errors  dw    0     ;error count for pass
             ;
                                ;the following variables
                                ;are used by RDBUF and
                                ;WTBUF to address the
                                ;disk, and by PERR to
                                ;display failing disk
                                ;addresses ---
0107  00     drv     db    0     ;drive to test
0108  0000   trk     dw    0     ;current track
010A  0000   sec     dw    0     ;current sector
010C  0000   buffer  dw    0     ;current memory address
010E  0000   iolen   dw    0     ;bytes last transferred
             ;
                                ;the following variables
                                ;define the area to be
                                ;tested on the selected
                                ;disk drive ---
0110  0000   trkf    dw    0     ;first track to test
0112  0000   trkl    dw    0     ;last track to test
0114  0000   secf    dw    0     ;first sector to test
0116  0000   secl    dw    0     ;last sector to test
             ;
                                ;the following variables
                                ;define the test mode ---
                                ;
0118  00     bypass  db    0     ;0=do not bypass error
                                ;itemization, 1=bypass
                                ;error itemization,print
                                ;total errors per pass
                                ;only.
                                ;
0119  00     skew    db    0     ;0=no sector skew
                                ;1=use sector skew for
                                ;  increased test speed
                                ;
011A  00     list    db    0     ;0=print errors on
                                ;terminal, 1=print errors
                                ;on list device.
                                ;
011B  00     lockio  db    0     ;0=no lock
                                ;1=lock on read
                                ;2=lock on write
                                ;
011C  00     restor  db    0     ;0=do not restore original
                                ;data, 1=restore original
                                ;data on diskette
                                ;
011D  00     lockpt  db    0     ;0=use variable test
                                ;data pattern, 1=lock on
                                ;user supplied data pattern
                                ;
011E  00     pattrn  db    0     ;contains user supplied
                                ;8 bit data pattern
                                ;
011F  0000   passl   dw    0     ;last pass to do on this
```

(Continued on top of page 22)

```
                                ;device, since errors
                                ;will not be listed
0189  11E205 dtst03  ld    de,dtstc;audit errors on console
018C  CDAE09         call  getl    ;or line printer?
018F  FE63           cp    'c'
0191  280C           jr    z,dtst04;c=use console
0193  FE70           cp    'p'
0195  C4560A         call  nz,query
0198  20EF           jr    nz,dtst03;no match,try again
019A  3E01           ld    a,1     ;p=use line printer
019C  321A01         ld    (list),a

019F  110306 dtst04  ld    de,dtstd;lock on read or write?
01A2  CDAE09         call  getl
01A5  FE6E           cp    'n'     ;n=no locks
01A7  2819           jr    z,dtst06
01A9  FE72           cp    'r'     ;r=lock on read
01AB  2007           jr    nz,dtst05
01AD  3E01           ld    a,1
01AF  321B01         ld    (lockio),a
01B2  1835           jr    dtst12  ;bypass querys about
                                ;restore mode and
                                ;data pattern: since
                                ;we are locked in read
                                ;mode, they are
                                ;irrelevant.
                                ;
01B4  FE77   dtst05  cp    'w'     ;w=lock on write
01B6  C4560A         call  nz,query
01B9  20E4           jr    nz,dtst04 ;no match,try again
01BB  3E02           ld    a,2
01BD  321B01         ld    (lockio),a
01C0  180F           jr    dtst08  ;bypass restore question,
                                ;since we are locked in
                                ;write mode.
                                ;
01C2  112506 dtst06  ld    de,dtste;restore user data?
01C5  CD8309         call  getyn
01C8  FE79           cp    'y'     ;y=restore
01CA  2005           jr    nz,dtst08
01CC  3E01           ld    a,1     ;n=do not restore
01CE  321C01         ld    (restor),a
01D1  113F06 dtst08  ld    de,dtstf;lock on data pattern?
01D4  CD8309         call  getyn
01D7  FE6E           cp    'n'
01D9  280E           jr    z,dtst12;n=use variable pattern
01DB  3E01           ld    a,1     ;y=lock on pattern
01DD  321D01         ld    (lockpt),a ;supplied by operator
01E0  115806         ld    de,dtstg;accept data pattern
01E3  CD0B0A         call  geth    ;from keyboard
01E6  321E01         ld    (pattrn),a
01E9  117806 dtst12  ld    de,dtsth;select drive to be
01EC  CDAE09         call  getl    ;used for test
01EF  D661           sub   'a'     ;convert to logical #
01F1  FE01           cp    $drvf   ;make sure its legal
01F3  DC560A         call  c,query
```

(Continued on next page)

# Z-80 Floppy Disk (Continued)

```
01F6  38F1          0226            jr      c,dtst12 ;too small,try again
01F8  FE04          0227            cp      $drvl+1
01FA  D4560A        0228            call    nc,query
01FD  30EA          0229            jr      nc,dtst12 ;too large,try again
01FF  320701        0230            ld      (drv),a ;save drive assignment
0202  C641          0231            add     'A'     ;also format for output
0204  32AA06        0232            ld      (dtstil),a
0207  119406        0233            ld      de,dtsti;confirm selected drive?
020A  CD8309        0234            call    getyn
020D  FE6E          0235            cp      'n'
020F  28D8          0236            jr      z,dtst12;not confirmed,try again
                    0237                    ;
                    0238                    ;initialize track
                    0239                    ;limits
0211  210000        0240            ld      hl,$trkf
0214  221001        0241            ld      (trkf),hl
0217  214C00        0242            ld      hl,$trkl
021A  221201        0243            ld      (trkl),hl
021D  11AF06  dtst15 0244           ld      de,dtstj ;test all tracks?
0220  CD8309        0245            call    getyn
0223  FE79          0246            cp      'y'     ;y=use all of them.
0225  281E          0247            jr      z,dtst20;n=user wants to specify
                    0248                    ;   range of tracks
0227  11C606  dtst17 0249           ld      de,dtstk;enter first track
022A  CDC509        0250            call    getn    ;to test
022D  221001        0251            ld      (trkf),hl ;save it
0230  11E206        0252            ld      de,dtstl;enter last track
0233  CDC509        0253            call    getn    ;to test
0236  221201        0254            ld      (trkl),hl ;save it
0239  ED5B1001      0255            ld      de,(trkf);make sure first
023D  B7            0256            or      a,a     ;track<=last track
023E  ED52          0257            sbc     hl,de
0240  DC560A        0258            call    c,query ;wrong,start over
0243  38E2          0259            jr      c,dtst17
                    0260 dtst20             ;initialize sector
                    0261                    ;limits
0245  210100        0262            ld      hl,$secf
0248  221401        0263            ld      (secf),hl
024B  211A00        0264            ld      hl,$secl
024E  221601        0265            ld      (secl),hl
0251  11FD06  dtst22 0266           ld      de,dtstm;use all sectors
0254  CD8309        0267            call    getyn   ;of each track?
0257  FE79          0268            cp      'y'
0259  281E          0269            jr      z,dtst26;y=use all sectors
                    0270                    ;n=user wants to specify
                    0271                    ;range of sectors
025B  111507  dtst24 0272           ld      de,dtstn;enter first sector
025E  CDC509        0273            call    getn    ;to test.
0261  221401        0274            ld      (secf),hl ;save it.
0264  113007        0275            ld      de,dtsto;enter last sector
0267  CDC509        0276            call    getn    ;to test.
026A  221601        0277            ld      (secl),hl ;save it.
026D  ED5B1401      0278            ld      de,(secf);make sure first
0271  B7            0279            or      a,a     ;sector<=last sector
0272  ED52          0280            sbc     hl,de
0274  DC560A        0281            call    c,query
0277  38E2          0282            jr      c,dtst24;error,start over

02E8  200C          0338            jr      nz,dtst70 ;yes,jump
                    0339                    ;no, compare test data
02EA  210010        0340            ld      hl,buff1;written to data read
02ED  11002A        0341            ld      de,buff2;back from disk. If
02F0  010E09        0342            ld      bc,merr4;difference found,
02F3  CD1F04        0343            call    bufcmp  ;print error message
02F6  3A1C01  dtst70 0344           ld      a,(restor)
02F9  B7            0345            or      a,a     ;using restore mode?
02FA  281E          0346            jr      z,dtst80;no, jump
                    0347                    ;yes,write back user's
                    0348                    ;data
02FC  210044        0349            ld      hl,buff3
02FF  114C09        0350            ld      de,merr6
0302  CDB503        0351            call    wtbuf
0305  210010        0352            ld      hl,buff1;verify that

0308  116809        0353            ld      de,merr7;it was rewritten ok
030B  CD8703        0354            call    rdbuf
030E  210010        0355            ld      hl,buff1
0311  110044        0356            ld      de,buff3
0314  012B09        0357            ld      bc,merr5;check restored data
0317  CD1F04        0358            call    bufcmp
                    0359                    ;if difference found,
                    0360                    ;print 'data cannot
                    0361                    ;be restored'
                    0362                    ;
                    0363 dtst80             ;advance current track
031A  ED5B0801      0364            ld      de,(trk)
031E  13            0365            inc     de
031F  ED530801      0366            ld      (trk),de
0323  2A1201        0367            ld      hl,(trkl)
0326  B7            0368            or      a,a     ;done with all tracks?
0327  ED52          0369            sbc     hl,de
0329  D2A102        0370            jp      nc,dtst42 ;no,process another
                    0371                    ;
                    0372 dtst90             ;end of pass
032C  ED4B0301      0373            ld      bc,(pass)
0330  03            0374            inc     bc      ;count passes
0331  ED430301      0375            ld      (pass),bc
0335  219B07        0376            ld      hl,dtstr1
0338  CD7A0A        0377            call    conv    ;convert pass #
033B  ED4B0501      0378            ld      bc,(errors)
033F  21AA07        0379            ld      hl,dtstr2
0342  CD7A0A        0380            call    conv    ;convert error count
0345  119307        0381            ld      de,dtstr;print pass and errors
0348  0E09          0382            ld      c,9     ;on console
034A  CD0500        0383            call    cpm
034D  3A1A01        0384            ld      a,(list);also using printer?
0350  B7            0385            or      a,a
0351  2806          0386            jr      z,dtst92;no, jump
                    0387                    ;yes,also send pass
                    0388                    ;and error count to
                    0389                    ;list device
0353  219307        0390            ld      hl,dtstr
0356  CDED04        0391            call    perr9
                    0392 dtst92             ;reset error count
```

```
                        0283                    ;
                        0284                    ;all variables set up
                        0285                    ;now --- how many
0279  114B07            0286 dtst26  ld      de,dtstp;test passes should be
027C  CDC509            0287         call    getn    ;made?
027F  221F01            0288         ld      (passl),hl ;save # of passes
                        0289                    ;
                        0290                    ;print advisory message
0282  11DB07            0291         ld      de,dtstt;as test begins
0285  0E09              0292         ld      c,9
0287  CD0500            0293         call    cpm
028A  111508            0294         ld      de,dtstu;remind user whether he
028D  3A1C01            0295         ld      a,(restor) ;is using restore

0290  B7                0296         or      a,a     ;mode
0291  2803              0297         jr      z,dtst32
0293  114008            0298         ld      de,dtstv
0296  0E09              0299 dtst32  ld      c,9
0298  CD0500            0300         call    cpm
                        0301                    ;
                        0302 dtst40             ;begin a pass
029B  2A1001            0303         ld      hl,(trkf)
029E  220801            0304         ld      (trk),hl;initialize current track
                        0305                    ;
                        0306 dtst42             ;process next track
02A1  0E06              0307         ld      c,6     ;check for interruption
02A3  1EFF              0308         ld      e,0ffh
02A5  CD0500            0309         call    cpm     ;from console
02A8  B7                0310         or      a,a
02A9  C26D03            0311         jp      nz,dtst94 ;break detected,quit
02AC  3A1C01            0312         ld      a,(restor)
02AF  B7                0313         or      a,a     ;is this restore mode?
02B0  2809              0314         jr      z,dtst45;no, jump
02B2  210044            0315         ld      hl,buff3;yes, save current
02B5  11BE08            0316         ld      de,merr1;disk contents
02B8  CD8703            0317         call    rdbuf
02BB  3A1B01            0318 dtst45  ld      a,(lockio)
02BE  FE01              0319         cp      a,1     ;is this lock on read?
02C0  2812              0320         jr      z,dtst47;yes, jump
02C2  210010            0321         ld      hl,buff1;set up test pattern
02C5  11000D            0322         ld      de,$bpt
02C8  CD0204            0323         call    bufpat
02CB  210010            0324         ld      hl,buff1;write test pattern
02CE  11D908            0325         ld      de,merr2
02D1  CDB503            0326         call    wtbuf
02D4  3A1B01            0327 dtst47  ld      a,(lockio)
02D7  FE02              0328         cp      a,2     ;is this lock on write?
02D9  281B              0329         jr      z,dtst70;yes, jump
02DB  21002A            0330         ld      hl,buff2;read back test pattern
                        0331                    ;(or just read existing
                        0332                    ; data if locked on read)
02DE  11F408            0333         ld      de,merr3
02E1  CD8703            0334         call    rdbuf
02E4  3A1B01            0335 dtst50  ld      a,(lockio)
02E7  B7                0336         or      a,a     ;is this lock on
                        0337                    ;read or write?
```

*(Continued on top of page 24)*

```
0359  AF                0393         xor     a,a
035A  320501            0394         ld      (errors),a
035D  320601            0395         ld      (errors+1),a
0360  2A0301            0396         ld      hl,(pass)
0363  ED5B1F01          0397         ld      de,(passl)
0367  B7                0398         or      a,a     ;are enough passes done?
0368  ED52              0399         sbc     hl,de
036A  DA9B02            0400         jp      c,dtst40;not yet,loop
                        0401 dtst94             ;done with all passes
036D  115E08            0402         ld      de,dtstw;ask whether to exit
0370  CDAE09            0403         call    getl    ;or to continue test
0373  FE63              0404         cp      'c'     ;c=continue
0375  CA2401            0405         jp      z,dtst
0378  FE65              0406         cp      'e'     ;e=exit
037A  20F1              0407         jr      nz,dtst94;if no match,try again
037C  117D08            0408         ld      de,dtstx;print goodbye
037F  0E09              0409         ld      c,9

0381  CD0500            0410         call    cpm     ;and return control
0384  C30000            0411         jp      wboot   ;to CP/M

                        0414 ;
                        0415 ; routines to read and write up to one track
                        0416 ;
                        0417 rdbuf              ;read current track from
                        0418                    ;secf to secl
                        0419                    ;
                        0420                    ;call hl=buffer base addr
                        0421                    ;     de=error msg addr
0387  ED53B303          0422         ld      (rdbufa),de ;save message address
038B  220C01            0423         ld      (buffer),hl ;save buffer address
038E  210000            0424         ld      hl,0    ;initialize transfer byte
0391  220E01            0425         ld      (iolen),hl ;count
0394  CD1D05            0426         call    seldsk  ;select disk
0397  2A1401            0427         ld      hl,(secf)
039A  220A01            0428         ld      (sec),hl;initialize current sector
039D  CD4D05            0429 rdbuf1  call    setio   ;set up track,sector,memory
03A0  CD5705            0430         call    read    ;now request transfer
03A3  B7                0431         or      a,a     ;was i/o successful?
03A4  2807              0432         jr      z,rdbuf2;no error, jump
03A6  ED5BB303          0433         ld      de,(rdbufa)
03AA  CD6A04            0434         call    perr    ;i/o error, audit it
03AD  CDE103            0435 rdbuf2  call    rwadv   ;advance sector address
03B0  30EB              0436         jr      nc,rdbuf1;not done,read another
03B2  C9                0437         ret             ;back to caller
03B3  0000              0438 rdbufa  dw      0       ;address of error message
                        0439 ;
                        0440 wtbuf              ;write current track
                        0441                    ;from secf to secl
                        0442                    ;
                        0443                    ;call de=error msg addr
                        0444                    ;     hl=buffer base addr
                        0445                    ;
03B5  ED53B303          0446         ld      (wtbufa),de ;save message addr
03B9  220C01            0447         ld      (buffer),hl ;save memory addr
03BC  210000            0448         ld      hl,0    ;initialize transfer
03BF  220E01            0449         ld      (iolen),hl ;byte count
```

*(Continued on next page)*

# Z-80 Floppy Disk (Continued)

```
03C2  CD1D05          0450            call    seldsk  ;select disk drive
03C5  2A1401          0451            ld      hl,(secf)
03C8  220A01          0452            ld      (sec),hl;initialize current sector
03CB  CD4D05          0453 wtbuf1     call    setio   ;set track,sector,memory
03CE  CD5C05          0454            call    write   ;request disk write
03D1  B7              0455            or      a,a     ;any i/o errors?
03D2  2807            0456            jr      z,wtbuf2;no, jump
03D4  ED5BB303        0457            ld      de,(wtbufa)
03D8  CD6A04          0458            call    perr    ;error, audit it
03DB  CDE103          0459 wtbuf2     call    rwadv   ;advance sector address
03DE  30EB            0460            jr      nc,wtbuf1;not done,write another
03E0  C9              0461            ret             ;back to caller
      (03B3)          0462 wtbufa     equ     rdbufa  ;save address of error
                      0463                            ;message
                      0464                            ;
                      0465 rwadv                      ;advance sector and
                      0466                            ;memory addresses
                      0467                            ;
03E1  118000          0468            ld      de,$bps ; de <- bytes per sector
03E4  2A0C01          0469            ld      hl,(buffer)
03E7  19              0470            add     hl,de   ;update buffer address
03E8  220C01          0471            ld      (buffer),hl
03EB  2A0E01          0472            ld      hl,(iolen)
03EE  19              0473            add     hl,de   ;count bytes transferred
03EF  220E01          0474            ld      (iolen),hl
03F2  ED5B0A01        0475            ld      de,(sec)
03F6  13              0476            inc     de      ;advance current sector
03F7  ED530A01        0477            ld      (sec),de
03FB  2A1601          0478            ld      hl,(secl)
03FE  B7              0479            or      a,a     ;done with all sectors?
03FF  ED52            0480            sbc     hl,de   ;exit with carry set if
                      0481                            ;done
0401  C9              0482            ret
                      0484 ;
                      0485 ; set up buffer with test pattern
                      0486 ;
                      0487 bufpat                     ;call hl=address of base
                      0488                            ;         of buffer
                      0489                            ;      de=byte length of
                      0490                            ;         area to set up
0402  3A1D01          0491            ld      a,(lockpt)
0405  B7              0492            or      a,a     ;are we locked on user
                      0493                            ;specified data pattern?
0406  200C            0494            jr      nz,bufpa2;yes, jump
0408  ED5F            0495 bufpa1     ld      a,r     ;read refresh register
040A  AC              0496            xor     a,h
040B  85              0497            add     a,l
                      0498                            ;make data a function of
                      0499                            ;memory address
040C  77              0500            ld      (hl),a  ;and store it
040D  23              0501            inc     hl      ;advance buffer address
040E  1B              0502            dec     de      ;count bytes stored
040F  7A              0503            ld      a,d     ;done yet?
0410  B3              0504            or      a,e
0411  20F5            0505            jr      nz,bufpa1 ;no,loop
0413  C9              0506            ret
                      0507                            ;user specified pattern
```

```
0457  D1              0565            pop     de
0458  C1              0566            pop     bc
                      0567 bufcm4                     ;advance memory address
                      0568                            ;out of this sector wher
                      0569                            ;an error was found.
0459  23              0570            inc     hl      ;bump buffer addresses
045A  13              0571            inc     de
045B  0B              0572            dec     bc      ;done with all data area?
045C  78              0573            ld      a,b
045D  B1              0574            or      a,c
045E  C8              0575            ret     z       ;yes,exit compare routine
045F  7D              0576            ld      a,l     ;check if on new sector
0460  E67F            0577            and     $bps-1  ;boundary
0462  28C6            0578            jr      z,bufcm1;found it, go compare
                      0579                            ;more data
0464  18F3            0580            jr      bufcm4  ;keep advancing until
                      0581                            ;sector boundary.
                      0582                            ;
0466  0000            0583 bufcma     dw      0       ;address of error message
0468  0000            0584 bufcmb     dw      0       ;base buffer address
                      0586 ;
                      0587 perr                       ;error printing routine,
                      0588                            ;prints pass,drive,track,
                      0589                            ;sector, and message
                      0590                            ;specified by caller on
                      0591                            ;console or list device.
                      0592                            ;
                      0593                            ; call with de=address
                      0594                            ;    of message giving
                      0595                            ;    type of error
                      0596                            ;
046A  3A1B01          0597            ld      a,(bypass)
046D  B7              0598            or      a,a     ;is error itemization
                      0599                            ;bypass flag set?
046E  205A            0600            jr      nz,perr2;yes,skip printing
                      0601                            ;and go count errors
0470  ED53FC04        0602            ld      (perra),de ;save message addr.
0474  ED4B0301        0603            ld      bc,(pass)
0478  03              0604            inc     bc
0479  210005          0605            ld      hl,perrc;convert current pass
047C  CD7A0A          0606            call    conv
047F  3A0701          0607            ld      a,(drv) ;form drive name
0482  C641            0608            add     'A'
0484  320805          0609            ld      (perrd),a
0487  ED4B0801        0610            ld      bc,(trk);convert current track
048B  210E05          0611            ld      hl,perre
048E  CD7A0A          0612            call    conv
0491  ED4B0A01        0613            ld      bc,(sec);convert current sector
0495  3A1901          0614            ld      a,(skew);is skew in effect?
0498  B7              0615            or      a,a
0499  2803            0616            jr      z,perr0 ;no
049B  CD6105          0617            call    sectran ;yes, translate sector
049E  211505          0618 perr0      ld      hl,perrf
04A1  CD7A0A          0619            call    conv
04A4  3A1A01          0620            ld      a,(list);should output be on
04A7  B7              0621            or      a,a     ;console or printer?
04A8  2028            0622            jr      nz,perr3;jump,use printer
```

```
0414  3A1E01        0508 bufpa2  ld      a,(pattrn)
0417  77            0509         ld      (hl),a   ;store one byte
0418  23            0510         inc     hl       ;advance buffer address
0419  1B            0511         dec     de       ;count bytes stored
041A  7A            0512         ld      a,d      ;done yet?
041B  B3            0513         or      a,e
041C  20F6          0514         jr      nz,bufpa2 ;not done,loop
041E  C9            0515         ret              ;exit
                    0516                          ;
                    0518 ;
                    0519 ;
                    0520 ; compare specified buffer and print error
                    0521 ; message if difference found
                    0522 ;
                    0523 bufcmp                   ;compare buffers
                    0524                          ;
                    0525                          ;call bc=address of
                    0526                          ;         error message
                    0527                          ;      de=address 1st buffer
                    0528                          ;      hl=address 2nd buffer
                    0529                          ;
041F  ED436604      0530         ld      (bufcma),bc ;save msg address
0423  226804        0531         ld      (bufcmb),hl ;save base of buffer
0426  ED4B0E01      0532         ld      bc,(iolen)  ;length to compare
042A  1A            0533 bufcm1  ld      a,(de)  ;fetch byte from 1st buffer
042B  BE            0534         cp      a,(hl)  ;compare it to 2nd buffer
042C  2008          0535         jr      nz,bufcm3 ;difference found,jump
042E  23            0536 bufcm2  inc     hl       ;advance buffer addresses
042F  13            0537         inc     de
0430  0B            0538         dec     bc       ;count bytes
0431  78            0539         ld      a,b      ;done yet?
0432  B1            0540         or      a,c
0433  20F5          0541         jr      nz,bufcm1 ;no,loop
0435  C9            0542         ret              ;back to caller
                    0543                          ;
                    0544 bufcm3                   ;difference found, print
                    0545                          ;error audit trail
0436  C5            0546         push    bc       ;first save registers
0437  D5            0547         push    de
0438  E5            0548         push    hl
0439  ED5B6804      0549         ld      de,(bufcmb)
043D  B7            0550         or      a,a
043E  ED52          0551         sbc     hl,de    ;find a buffer offset
0440  E5            0552         push    hl       ;now divide by bytes per
0441  C1            0553         pop     bc       ;sector to find relative
0442  118000        0554         ld      de,$bps  ;sector number
0445  CDA10A        0555         call    div
0448  2A1401        0556         ld      hl,(secf)
044B  09            0557         add     hl,bc    ;add relative sector to
                    0558                          ;first sector to find
                    0559                          ;actual address for use
                    0560                          ;by PERR
044C  220A01        0561         ld      (sec),hl
044F  ED5B6604      0562         ld      de,(bufcma)
0453  CD6A04        0563         call    perr     ;now audit error
0456  E1            0564         pop     hl       ;restore registers
```

*(Continued on top of page 26)*

```
                    0623                          ;fall thru,use console
04AA  2A0501        0624         ld      hl,(errors)
04AD  7C            0625         ld      a,h      ;is this first error?
04AE  B5            0626         or      a,l
04AF  2008          0627         jr      nz,perr1;no, jump
04B1  116707        0628         ld      de,dtstq;print title for errors
04B4  0E09          0629         ld      c,9
04B6  CD0500        0630         call    cpm
04B9  11FE04        0631 perr1   ld      de,perrb;print disk address
04BC  0E09          0632         ld      c,9
04BE  CD0500        0633         call    cpm
04C1  ED5BFC04      0634         ld      de,(perra)
04C5  0E09          0635         ld      c,9      ;print error type
04C7  CD0500        0636         call    cpm
                    0637                          ;
                    0638 perr2                    ;count errors
04CA  2A0501        0639         ld      hl,(errors)
04CD  23            0640         inc     hl
04CE  220501        0641         ld      (errors),hl
04D1  C9            0642         ret              ;back to caller
                    0643                          ;
                    0644 perr3                    ;errors to printer
04D2  2A0501        0645         ld      hl,(errors)
04D5  7C            0646         ld      a,h      ;is this 1st error to
04D6  B5            0647         or      a,l      ;be printed this pass?
04D7  2006          0648         jr      nz,perr4;no, jump
04D9  216707        0649         ld      hl,dtstq;yes,print title
04DC  CDED04        0650         call    perr9
04DF  21FE04        0651 perr4   ld      hl,perrb;print disk address
04E2  CDED04        0652         call    perr9
04E5  2AFC04        0653         ld      hl,(perra)
04E8  CDED04        0654         call    perr9    ;print error type
04EB  18DD          0655         jr      perr2    ;go count errors
                    0656                          ;
                    0657 perr9                    ;send a string
                    0658                          ;terminated by '$'
                    0659                          ;to list device
04ED  7E            0660         ld      a,(hl)   ;fetch next char
04EE  FE24          0661         cp      a,'$'    ;is it terminator?
04F0  C8            0662         ret     z        ;yes,exit
04F1  E5            0663         push    hl       ;save string addr.
04F2  5F            0664         ld      e,a      ;send this character
04F3  0E05          0665         ld      c,5
04F5  CD0500        0666         call    cpm
04F8  E1            0667         pop     hl       ;restore string addr
04F9  23            0668         inc     hl       ;and increment it
04FA  18F1          0669         jr      perr9    ;check next char.
                    0670                          ;
04FC  0000          0671 perra   dw      0        ;addr of message
                    0672                          ;describing error type
04FE  0D0A          0673 perrb   db      cr,lf
0500  6E6E6E6E      0674 perrc   db      'nnnn    ' ;pass #
      20202020
0508  6E202020      0675 perrd   db      'n       ' ;drive
      2020
050E  6E6E6E6E      0676 perre   db      'nnnn    ' ;track
```

*(Continued on next page)*

# Z-80 Floppy Disk (Continued)

```
                202020
0515  6E6E6E6E  0677 perrf   db      'nnnn   $' ;sector
                20202024
                0678 ;
                0680 ;
                0681 ;
                0682 ; disk interface to CP/M BIOS
                0683 ;
051D  3A0701    0684 seldsk  ld      a,(drv) ;select disk drive
0520  4F        0685         ld      c,a
0521  111800    0686         ld      de,24
                0687                         ;this routine links
                0688                         ;to the desired routine
                0689                         ;through the standard
                0690                         ;CP/M BIOS jump table
0524  2A0100    0691 jpbios  ld      hl,(wboot+1)
0527  19        0692         add     hl,de
0528  E9        0693         jp      (hl)
                0694 ;
0529  ED4B0801  0695 settrk  ld      bc,(trk);select track
052D  111B00    0696         ld      de,27
0530  18F2      0697         jr      jpbios
                0698 ;
0532  ED4B0A01  0699 setsec  ld      bc,(sec);select sector
0536  111E00    0700         ld      de,30
0539  3A1901    0701         ld      a,(skew);use sector skew?
053C  B7        0702         or      a,a
053D  28E5      0703         jr      z,jpbios;no
053F  CD6105    0704         call    sectran ;translate sector addr.
0542  18E0      0705         jr      jpbios
                0706 ;
0544  ED4B0C01  0707 setdma  ld      bc,(buffer) ;set memory addr.
0548  112100    0708         ld      de,33
054B  18D7      0709         jr      jpbios
                0710 ;
054D  CD2905    0711 setio   call    settrk  ;set up track,sector,
0550  CD3205    0712         call    setsec  ;and memory address
0553  CD4405    0713         call    setdma  ;for subsequent read
0556  C9        0714         ret             ;or write
                0715 ;
                0716 read                    ;read one disk sector
0557  112400    0717         ld      de,36
055A  18C8      0718         jr      jpbios
                0719 ;
                0720 write                   ;write one disk sector
055C  112700    0721         ld      de,39
055F  18C3      0722         jr      jpbios
                0723 ;
                0724 sectran                 ;translate logical to
                0725                         ;physical sector number
                0726                         ;
                0727                         ;call bc=logical sector
                0728                         ;return bc=physical sector
0561  E5        0729         push    hl
0562  216805    0730         ld      hl,sectrb-1
0565  09        0731         add     hl,bc
0566  4E        0732         ld      c,(hl)
```

```
0612  6F722077  0758         db      'or write? (N/R/W) $'
      72697465
      3F20284E
      2F522F57
      292024
0625  0D0A5265  0759 dtste   db      cr,lf,'Restore '
      73746F72
      6520
062F  6F726967  0760         db      'original data? $'
      696E616C
      20646174
      613F2024
063F  0D0A4C6F  0761 dtstf   db      cr,lf,'Lock on '
      636B206F
      6E20
0649  64617461  0762         db      'data pattern? $'
      20706174
      7465726E
      3F2024
0658  0D0A456E  0763 dtstg   db      cr,lf,'Enter data '
      74657220
      64617461
      20
0665  70617474  0764         db      'pattern, hex 00-FF$'
      65726E2C
      20686578
      2030302D
      464624
0678  0D0A4472  0765 dtsth   db      cr,lf,'Drive '
      69766520
0680  746F2062  0766         db      'to be tested '
      65207465
      73746564
      20
068D  28422D    0767         db      '(',$drvf+'A','-'
0690  44292024  0768         db      $drvl+'A',') $'
0694  0D0A436F  0769 dtsti   db      cr,lf,'Confirm: test drive '
      6E666972
      6D3A2074
      65737420
      64726976
      6520
06AA  58203F20  0770 dtstil  db      'X ? $'
      24
06AF  0D0A5465  0771 dtstj   db      cr,lf,'Test all '
      73742061
      6C6C20
06BA  74726163  0772         db      'tracks?    $'
      6B733F20
      20202024
06C6  0D0A4669  0773 dtstk   db      cr,lf,'First '
      72737420
06CE  74726163  0774         db      'track to test      $'
      6B20746F
      20746573
      74202020
      20202024
```

```
0567  E1              0733           pop     hl
0568  C9              0734           ret
0569  0104070A        0735 sectrb    db      1,4,7,10,13      ;table built
      0D
056E  10131619        0736           db      16,19,22,25,2    ;with skew
      02
0573  05080B0E        0737           db      5,8,11,14,17,20  ;factor =3
      1114
0579  171A0306        0738           db      23,26,3,6,9,12
      090C
057F  0F121518        0739           db      15,18,21,24
                      0741 ;
                      0742 ; messages for test initialization and
                      0743 ; error printing
                      0744 ;
0583  0D0A0A          0745 dtsta     db      cr,lf,lf
0586  5A2D3830        0746           db      'Z-80 Floppy Disk '
      20466C6F
      70707920
      4469736B
      20
0597  54657374        0747           db      'Test version '
      20766572
      73696F6E
      20
05A4  312E            0748           db      $ver+'0','.'
05A6  300D0A          0749           db      $rev+'0',cr,lf
05A9  28632920        0750           db      '(c) 1980 Laboratory '
      31393830
      204C6162
      6F726174
      6F727920
05BD  4D696372        0751           db      'Microsystems',cr,lf,'$'
      6F737973
      74656D73
      0D0A24
05CC  0D0A4974        0752 dtstb     db      cr,lf,'Itemize '
      656D697A
      6520
05D6  6572726F        0753           db      'errors?    $'
      72733F20
      20202024
05E2  0D0A5573        0754 dtstc     db      cr,lf,'Use '
      6520
05E8  636F6E73        0755           db      'console or printer'
      6F6C6520
      6F722070
      72696E74
      6572
05FA  3F202843        0756           db      '? (C/P) $'
      2F502920
      24
0603  0D0A4C6F        0757 dtstd     db      cr,lf,'Lock on read '
      636B206F
      6E207265
      616420
```

*(Continued on top of page 28)*

```
06E2  0D0A4C61        0775 dtstl     db      cr,lf,'Last '
      737420
06E9  74726163        0776           db      'track to test      $'
      6B20746F
      20746573
      74202020
      20202024
06FD  0D0A5465        0777 dtstm     db      cr,lf,'Test all '
      73742061
      6C6C20
0708  73656374        0778           db      'sectors?    $'
      6F72733F
      20202020
      24
0715  0D0A4669        0779 dtstn     db      cr,lf,'First '
      72737420
071D  73656374        0780           db      'sector to test    $'
      6F722074
      6F207465
      73742020
      202024
0730  0D0A4C61        0781 dtsto     db      cr,lf,'Last '
      737420
0737  73656374        0782           db      'sector to test     $'
      6F722074
      6F207465
      73742020
      20202024
074B  0D0A486F        0783 dtstp     db      cr,lf,'How many '
      77206D61
      6E7920
0756  74657374        0784           db      'test passes?    $'
      20706173
      7365733F
      20202020
      24
0767  0D0A0A50        0785 dtstq     db      cr,lf,lf,'Pass  '
      61737320
      20
0770  44726976        0786           db      'Drive  Track  '
      65202054
      7261636B
      2020
077E  53656374        0787           db      'Sector  Error-type'
      6F722020
      4572726F
      722D7479
      7065
0790  0D0A24          0788           db      cr,lf,'$'
0793  0D0A0A50        0789 dtstr     db      cr,lf,lf,'Pass '
      61737320
079B  6E6E6E6E        0790 dtstr1    db      'nnnn complete, '
      20636F6D
      706C6574
      652C20
07AA  6E6E6E6E        0791 dtstr2    db      'nnnn errors.'
```

*(Continued on next page)*

# Z-80 Floppy Disk (Continued)

```
                20657272
                6F72732E
07B6 0D0A24     0792            db      cr,lf,'$'
07B9 0D0A4E6F   0793 dtsts      db      cr,lf,'Not enough '
                7420656E
                6F756768
                20
07C6 6D656D6F   0794            db      'memory to execute. '
                72792074
                6F206578
                65637574
                652E
07D8 0D0A24     0795            db      cr,lf,'$'
07DB 0D0A0A42   0796 dtstt      db      cr,lf,lf,'Beginning '
                6567696E
                6E696E67
                20
07E8 6469736B   0797            db      'disk test - push '
                20746573
                74202D20
                70757368
                20
07F9 616E7920   0798            db      'any key to abort '
                6B657920
                746F2061
                626F7274
                20
080A 70726F67   0799            db      'program. ',cr,lf,'$'
                72616D2E
                0D0A24
0815 5761726E   0800 dtstu      db      'Warning: user '
                696E673A
                20757365
                7220
0823 64617461   0801            db      'data will not be '
                2077696C
                6C206E6F
                74206265
                20
0834 72657374   0802            db      'restored. ',cr,lf,'$'
                6F726564
                2E0D0A24
0840 55736572   0803 dtstv      db      'User data will be '
                20646174
                61207769
                6C6C2062
                6520
0852 72657374   0804            db      'restored. ',cr,lf,'$'
                6F726564
                2E0D0A24
085E 0D0A436F   0805 dtstw      db      cr,lf,'Continue or '
                6E74696E
                7565206F
                7220
086C 65786974   0806            db      'exit test? (C/E)$'
                20746573
                743F2028
                20
0940 62652072   0820            db      'be restored$'
                6573746F
                72656424
094C 77726974   0821 merr6      db      'write error - restore phase$'
                65206572
                726F7220
                2D207265
                73746F72
                65207068
                61736524
0968 72656164   0822 merr7      db      'read error - restore phase$'
                20657272
                6F72202D
                20726573
                746F7265
                20706861
                736524

                0823 ;
                0825 ;
                0826 ;
                0827 ; utility and console input routines
                0828 ;
                0829 getyn                      ;get y or n response
                0830                            ;from operator.
                0831                            ;
                0832                            ;call de=address of cue
                0833                            ;return acc=y or n
0983 D5         0834            push    de      ;save cue address
0984 0E09       0835            ld      c,9     ;print cue message
0986 CD0500     0836            call    cpm
0989 11A409     0837            ld      de,getyna
098C 0E09       0838            ld      c,9     ;print possible answers
098E CD0500     0839            call    cpm
0991 CD640A     0840            call    getchar ;get a character
                0841                            ;from console
0994 F620       0842            or      a,20h   ;fold to lower case
0996 D1         0843            pop     de      ;restore cue address
                0844                            ;in case needed again
0997 FE79       0845            cp      'y'     ;make sure response
                0846                            ;is ok
0999 C8         0847            ret     z       ;exit if y
099A FE6E       0848            cp      'n'
099C C8         0849            ret     z       ;exit if n
099D D5         0850            push    de
099E CD560A     0851            call    query   ;print question mark if
09A1 D1         0852            pop     de      ;not y or n, try again
09A2 18DF       0853            jr      getyn   ;
09A4 28592F4E   0854 getyna     db      '(Y/N) ',tab,'> $'
                2920093E
                2024

                0855 ;
                0856 ;
                0857 getl                       ;get any letter response
                0858                            ;from operator.
                0859                            ;
                0860                            ;call de=address of cue
```

```
                432F4529
                24
087D  0D0A0A    0807 dtstx   db      cr,lf,lf
0880  476F6F64    0808       db      'Goodbye. ',cr,lf,'$'
                6279652E
                0D0A24
088B  0D0A5573  0809 dtsty   db      cr,lf,'Use sector '
                65207365
                63746F72
                20
0898  736B6577  0810         db      'skew?  $'
                3F202024
08A0  0D0A4E65  0811 dtstz   db      cr,lf,'Need CP/M 2.x '
                65642043
                502F4D20
                322E7820
08B0  746F2065    0812       db      'to execute. ',cr,lf,'$'
                78656375
                74652E0D
                0A24
                0813 ;
                0814 ;
08BE  72656164  0815 merr1   db      'read error - original data$'
                20657272
                6F72202D
                206F7269
                67696E61
                6C206461
                746124
08D9  77726974  0816 merr2   db      'write error - test pattern$'
                65206572
                726F7220
                2D207465
                73742070
                61747465
                726E24
08F4  72656164  0817 merr3   db      'read error - test pattern$'
                20657272
                6F72202D
                20746573
                74207061
                74746572
                6E24
090E  636F6D70  0818 merr4   db      'compare error - test pattern$'
                61726520
                6572726F
                72202D20
                74657374
                20706174
                7465726E
                24
092B  6F726967  0819 merr5   db      'original data cannot '
                696E616C
                20646174
                61206361
                6E6E6F74
```

*(Continued on top of page 30)*

```
                0861                    ;return acc=ASCII char.
09AE  0E09      0862         ld     c,9     ;print cue message
09B0  CD0500    0863         call   cpm
09B3  11C109    0864         ld     de,getla;tab and print
09B6  0E09      0865         ld     c,9     ;cue mark
09B8  CD0500    0866         call   cpm
09BB  CD640A    0867         call   getchar ;read console
09BE  F620      0868         or     a,20h   ;fold to lower case
09C0  C9        0869         ret
09C1  093E2024  0870 getla   db     tab,'> $'
                0871 ;
                0872 ;
                0873 getn               ;get a decimal number
                0874                    ;from the console.
                0875                    ;
                0876                    ;call de=address of cue
                0877                    ;return hl=number
09C5  D5        0878         push   de      ;save cue message address
                0879                    ;in case needed later
09C6  0E09      0880         ld     c,9
09C8  CD0500    0881         call   cpm     ;print cue message
09CB  11050A    0882         ld     de,getna;print tab and cue mark
09CE  0E09      0883         ld     c,9
09D0  CD0500    0884         call   cpm
09D3  210000    0885         ld     hl,0    ;initialize forming
                0886                    ;answer
09D6  3A2101    0887         ld     a,(digits)
09D9  47        0888         ld     b,a     ;total characters allowed
                0889                    ;to be input
09DA  E5        0890 getn1   push   hl      ;save answer
09DB  C5        0891         push   bc      ;save char. count
09DC  CD640A    0892         call   getchar ;read console
09DF  C1        0893         pop    bc      ;restore char. count
09E0  E1        0894         pop    hl      ;restore forming answer
09E1  FE0D      0895         cp     cr      ;is this return?
09E3  281E      0896         jr     z,getn9 ;yes,exit with answer
09E5  FE30      0897         cp     '0'     ;is this legal char.?
09E7  3814      0898         jr     c,getn3 ;no, jump
09E9  FE3A      0899         cp     '9'+1   ;is this legal char.?
09EB  3010      0900         jr     nc,getn3;no, jump
09ED  E60F      0901         and    0fh     ;isolate bottom 4 bits
                0902                    ;previous data * 10
09EF  E5        0903         push   hl
09F0  D1        0904         pop    de
09F1  29        0905         add    hl,hl   ;(*2)
09F2  29        0906         add    hl,hl   ;(*4)
09F3  19        0907         add    hl,de   ;(*5)
09F4  29        0908         add    hl,hl   ;(*10)
09F5  5F        0909         ld     e,a     ;now add in this digit
09F6  1600      0910         ld     d,0
09F8  19        0911         add    hl,de
09F9  10DF      0912         djnz   getn1   ;count characters accepted
09FB  1806      0913         jr     getn9   ;enough accepted,exit
                0914 getn3              ;illegal character detected.
09FD  CD560A    0915         call   query   ;print question mark and
0A00  D1        0916         pop    de      ;restart input
```

*(Continued on next page)*

# Z-80 Floppy Disk (Continued)

```
0A01  18C2          0917            jr      getn
                    0918 getn9                      ;input complete,clean
                    0919                            ;stack and exit with
                    0920                            ;answer in (hl)
0A03  D1            0921            pop     de
0A04  C9            0922            ret
0A05  093E2024      0923 getna      db      tab,'> $'
0A09  3F24          0924 getnb      db      '?$'
                    0925 ;
                    0926 ;
                    0927 geth                       ;get $dig hex digits
                    0928                            ;from keyboard
                    0929                            ;
                    0930                            ;call de=addr of cue
                    0931                            ;return acc=lower 8 bits
                    0932                            ;       of entered number,
                    0933                            ;       hl=entire 16 bit no.
                    0934                            ;
0A0B  D5            0935            push    de      ;save cue address
                    0936                            ;in case needed again
0A0C  0E09          0937            ld      c,9
0A0E  CD0500        0938            call    cpm     ;print cue message
0A11  11520A        0939            ld      de,getha;print tab and cue mark
0A14  0E09          0940            ld      c,9
0A16  CD0500        0941            call    cpm
0A19  210000        0942            ld      hl,0    ;initialize forming
                    0943                            ;answer
0A1C  3A2101        0944            ld      a,(digits)
0A1F  47            0945            ld      b,a     ;max digits to accept
0A20  C5            0946 geth1      push    bc      ;save registers
0A21  E5            0947            push    hl
0A22  CD640A        0948            call    getchar ;read console
0A25  E1            0949            pop     hl
0A26  C1            0950            pop     bc      ;restore registers
0A27  FE0D          0951            cp      cr      ;if carriage return exit
0A29  281E          0952            jr      z,geth25
0A2B  FE30          0953            cp      '0'     ;make sure its legal
0A2D  381D          0954            jr      c,geth3 ;no, jump
0A2F  FE3A          0955            cp      '9'+1   ;if alpha fold to
0A31  3802          0956            jr      c,geth15;lower case
0A33  F620          0957            or      20h
0A35  FE67          0958 geth15     cp      'f'+1   ;make sure its legal
0A37  3013          0959            jr      nc,geth3;no, jump
0A39  FE61          0960            cp      'a'     ;check if alpha
0A3B  3802          0961            jr      c,geth2 ;jump if 0-9
0A3D  C609          0962            add     9       ;add correction
0A3F  E60F          0963 geth2      and     0fh
0A41  29            0964            add     hl,hl   ;previous data *16
0A42  29            0965            add     hl,hl   ;(left shift 4 bits)
0A43  29            0966            add     hl,hl
0A44  29            0967            add     hl,hl
0A45  85            0968            add     a,l     ;add this char. to
0A46  6F            0969            ld      l,a     ;forming result
0A47  10D7          0970            djnz    geth1   ;keep reading console
0A49  D1            0971 geth25     pop     de      ;clean up stack
0A4A  7D            0972            ld      a,l     ;put lower 8 bits
                    0973                            ;of answer in acc.
```

```
0A99  79            1030 conv9      ld      a,c     ;turn quotient into
0A9A  C630          1031            add     '0'     ;ASCII character
0A9C  77            1032            ld      (hl),a  ;and store it
0A9D  23            1033            inc     hl      ;bump output pointer
0A9E  D5            1034            push    de      ;bc <- remainder
0A9F  C1            1035            pop     bc
0AA0  C9            1036            ret
                    1037 ;
                    1038 ;
                    1039 div                        ;single precision divide
                    1040                            ;call bc=numerator
                    1041                            ;       de=divisor
                    1042                            ;return bc=quotient
                    1043                            ;       de=remainder
0AA1  E5            1044            push    hl
0AA2  210000        1045            ld      hl,0
0AA5  B7            1046            or      a,a
0AA6  ED52          1047            sbc     hl,de
0AA8  EB            1048            ex      de,hl
0AA9  210000        1049            ld      hl,0
0AAC  3E11          1050            ld      a,17
0AAE  E5            1051 div0       push    hl
0AAF  19            1052            add     hl,de
0AB0  3001          1053            jr      nc,div1
0AB2  E3            1054            ex      (sp),hl
0AB3  E1            1055 div1       pop     hl
0AB4  F5            1056            push    af
0AB5  CB11          1057            rl      c
0AB7  CB10          1058            rl      b
0AB9  CB15          1059            rl      l
0ABB  CB14          1060            rl      h
0ABD  F1            1061            pop     af
0ABE  3D            1062            dec     a
0ABF  20ED          1063            jr      nz,div0
0AC1  B7            1064            or      a,a
0AC2  CB1C          1065            rr      h
0AC4  CB1D          1066            rr      l
0AC6  EB            1067            ex      de,hl
0AC7  E1            1068            pop     hl
0AC8  C9            1069            ret
                    1070 ;
                    1071 ;
      (1000)        1072 buff1      equ     1000h   ;disk buffers
      (2A00)        1073 buff2      equ     $bpt*2+buff1
      (4400)        1074 buff3      equ     $bpt*2+buff2
      (5E00)        1075 buffend    equ     $bpt*2+buff3
                    1076 ;
                    1077 ;
0AC9  (0100)        1078            end     100h

Errors              0
```

```
                        0974                            ;(in case exit by
                        0975                            ; carriage return)
0A4B  C9                0976            ret
0A4C  CD560A            0977  geth3     call    query   ;print question mark
0A4F  D1                0978            pop     de      ;then restart input
0A50  18B9              0979            jr      geth
0A52  093E2024          0980  getha     db      tab,'> $'
                        0981  ;
                        0982  ;
0A56  F5                0983  query     push    af      ;save flags
0A57  0E09              0984            ld      c,9     ;print question mark
0A59  11610A            0985            ld      de,querya
0A5C  CD0500            0986            call    cpm
0A5F  F1                0987            pop     af      ;restore flags
0A60  C9                0988            ret
0A61  203F24            0989  querya    db      ' ?$'
                        0990  ;
                        0991  ;
                        0992  getchar                   ;get 1 character from
                        0993                            ;console via raw input
                        0994                            ;mode.  do not echo a
                        0995                            ;carriage return.
0A64  1EFF              0996            ld      e,0ffh
0A66  0E06              0997            ld      c,6
0A68  CD0500            0998            call    cpm     ;read console
0A6B  B7                0999            or      a,a     ;anything there?
0A6C  28F6              1000            jr      z,getchar ;no, try again
0A6E  FE0D              1001            cp      cr      ;is it a carriage return?
0A70  C8                1002            ret     z       ;yes
0A71  F5                1003            push    af      ;no, echo it
0A72  5F                1004            ld      e,a
0A73  0E06              1005            ld      c,6
0A75  CD0500            1006            call    cpm
0A78  F1                1007            pop     af      ;restore acc. and exit
0A79  C9                1008            ret
                        1009  ;
                        1010  ;
                        1011  conv                      ;convert binary to
                        1012                            ;decimal ascii
                        1013                            ;
                        1014                            ;call bc=binary data, in
                        1015                            ;        range 0000-9999.
                        1016                            ;     hl=first byte addr
                        1017                            ;        to store output
                        1018                            ;
0A7A  11E803            1019            ld      de,1000
0A7D  CDA10A            1020            call    div
0A80  CD990A            1021            call    conv9   ;thousands digit
0A83  116400            1022            ld      de,100
0A86  CDA10A            1023            call    div
0A89  CD990A            1024            call    conv9   ;hundreds digit
0A8C  110A00            1025            ld      de,10
0A8F  CDA10A            1026            call    div
0A92  CD990A            1027            call    conv9   ;tens digit
0A95  CD990A            1028            call    conv9   ;units
0A98  C9                1029            ret             ;back to caller
```

*(Continued on top of page 32)*

```
$BPS    0080    $BPT    0D00    $DIG    0002    $DRVF   0001    $DRVL   0003
$REV    0000    $SECF   0001    $SECL   001A    $TRKF   0000    $TRKL   004C
$VER    0001    BUFCM1  042A    BUFCM2  042E    BUFCM3  0436    BUFCM4  0459
BUFCMA  0466    BUFCMB  0468    BUFCMP  041F    BUFF1   1000    BUFF2   2A00
BUFF3   4400    BUFFEN  5E00    BUFFER  010C    BUFPA1  0408    BUFPA2  0414
BUFPAT  0402    BYPASS  0118    CONV    0A7A    CONV9   0A99    CPM     0005
CR      000D    DIGITS  0121    DIV     0AA1    DIV0    0AAE    DIV1    0AB3
DRV     0107    DTST    0124    DTST01  0142    DTST02  0159    DTST03  0189
DTST04  019F    DTST05  01B4    DTST06  01C2    DTST08  01D1    DTST12  01E9
DTST15  021D    DTST17  0227    DTST20  0245    DTST22  0251    DTST24  025B
DTST26  0279    DTST32  0296    DTST40  029B    DTST42  02A1    DTST45  02BB
DTST47  02D4    DTST50  02E4    DTST70  02F6    DTST80  031A    DTST90  032C
DTST92  0359    DTST94  036D    DTSTA   0583    DTSTB   05CC    DTSTC   05E2
DTSTD   0603    DTSTE   0625    DTSTF   063F    DTSTG   0658    DTSTH   0678
DTSTI   0694    DTSTI1  06AA    DTSTJ   06AF    DTSTK   06C6    DTSTL   06E2
DTSTM   06FD    DTSTN   0715    DTSTO   0730    DTSTP   074B    DTSTQ   0767
DTSTR   0793    DTSTR1  079B    DTSTR2  07AA    DTSTS   07B9    DTSTT   07DB
DTSTU   0815    DTSTV   0840    DTSTW   085E    DTSTX   087D    DTSTY   088B
DTSTZ   08A0    ERRORS  0105    FF      000C    GETCHA  0A64    GETH    0A0B
GETH1   0A20    GETH15  0A35    GETH2   0A3F    GETH25  0A49    GETH3   0A4C
GETHA   0A52    GETL    09AE    GETLA   09C1    GETN    09C5    GETN1   09DA
GETN3   09FD    GETN9   0A03    GETNA   0A05    GETNB   0A09    GETYN   0983
GETYNA  09A4    IOLEN   010E    JPBIOS  0524    LF      000A    LIST    011A
LOCKIO  011B    LOCKPT  011D    MERR1   08BE    MERR2   08D9    MERR3   08F4
MERR4   090E    MERR5   092B    MERR6   094C    MERR7   0968    PASS    0103
PASSL   011F    PATTRN  011E    PERR    046A    PERR0   049E    PERR1   04B9
PERR2   04CA    PERR3   04D2    PERR4   04DF    PERR9   04ED    PERRA   04FC
PERRB   04FE    PERRC   0500    PERRD   0508    PERRE   050E    PERRF   0515
QUERY   0A56    QUERYA  0A61    RDBUF   0387    RDBUF1  039D    RDBUF2  03AD
RDBUFA  03B3    READ    0557    RESTOR  011C    RWADV   03E1    SEC     010A
SECF    0114    SECL    0116    SECTRA  0561    SECTRB  0569    SELDSK  051D
SETDMA  0544    SETIO   054D    SETSEC  0532    SETTRK  0529    SKEW    0119
TAB     0009    TRK     0108    TRKF    0110    TRKL    0112    WBOOT   0000
WRITE   055C    WTBUF   03B5    WTBUF1  03CB    WTBUF2  03DB    WTBUFA  03B3
XTRAN   0122
```

**end**

# A Cassette I/O Routine

## Listing 1

```
10 POKE 16553,255:IF PEEK (16562)*256+PEEK (16561) > 32479 POKE 1
6562,120:POKE 16561,223:CLEAR:RUN
20 CLEAR 200:DEFINT H,I,V:DEFSTR C,D:DIM L(95)
30 READ C:IF C="END" THEN PRINT:RESTORE:GOTO 80 ELSE I=1:READ C1
:IF C="" THEN 50 ELSE A=0
40 V=0:GOSUB 300:A=A*16+V:IF I<= LEN(C) THEN 40
50 READ C:I=1:IF C1="" THEN 70 ELSE V=ASC(C1)-32:IF L(V)<>0 PRIN
T "DUPL LABEL ";C1:STOP
60 L(V)=A:PRINT C1;"=";:FOR I=4 TO 1 STEP -1:GOSUB 200:NEXT:PRIN
T " ",:I=1
70 IF I>LEN(C) THEN 30 ELSE D=MID$(C,I,1):I=I+1:IF D=" " THEN 70
 ELSE I=I+1:A=A+1:GOTO 70
80 PRINT "*";:READ C:IF C="END" THEN CLEAR 50:END ELSE READ C1:I
=1:IF C="" THEN READ C:GOTO 100 ELSE A=0
90 V=0:GOSUB 300:A=16*A+V:IF I<=LEN(C) THEN 90 ELSE READ C:I=1
100 IF I>LEN(C) THEN 80 ELSE V=0:GOSUB 300:IF D="R" OR D="W" THE
N 120 ELSE IF D=" " THEN 100
110 GOSUB 300:GOSUB 250:GOTO 100
120 V=ASC(MID$(C,I,1))-32:I=I+1:IF L(V)=0 PRINT "UNDEF LABEL":ST
OP
130 IF D="W" THEN 150 ELSE V=L(V)-A-1:IF V<-128 OR V>127 PRINT "
RANGE ERROR":STOP
140 V=V AND 255:GOSUB 250:GOTO 100
150 L=L(V):V=0:GOSUB 300:GOSUB 300:IF V>127 THEN V=256-V
160 L=L+V:V=(L-32768) AND 255:GOSUB 250
170 V=INT(L/256):GOSUB 250:GOTO 100
200 L=A/16^(I-1):IF L>32767 THEN L=L-32768
210 L=L AND 15:PRINT CHR$(L+48-7*(L>9));:RETURN
250 IF A>32767 POKEA-65536,V ELSE POKE A,V
260 A=A+1:RETURN
300 D=MID$(C,I,1):V=V*16+ASC(D)-48+7*(D>"9"):I=I+1:RETURN
400 DATA 408E,,WA00
410 DATA 7F00,A,CDWZ00 1A F5 C3WP00
420 DATA ,B,CDWZ00 CD9602 010006 1A F5 E5 21W*00
430 DATA ,C,CDWX00 2B 78 FE05 28RD FE04 28RD 1A BE 20RM
440 DATA ,D,13 10RC E1 D5 EB
450 DATA ,E,CDWX00 E5 DF E1 20RE E1 F1 FE03 20RJ 2B 46 23
460 DATA ,F,23 23 10RF
470 DATA ,G,46 23 D5 E5 2AA040 3AD640 90 5F 3AD740 DE00 57 DF EB
 30RM 22D640 23 EB E1 73 23 72 23 E5 EB 78 B7 28RI
480 DATA ,H,CDWX00 10RH
490 DATA ,I,E1 D1 E5 DF E1 20RG
500 DATA ,J,CD3502 81 20RO 212020 223E3C
510 DATA ,K,210000
520 DATA ,L,CDF801 C39A0A
530 DATA ,M,E1
540 DATA ,N,E1
550 DATA ,O,21FFFF 18RL
560 DATA ,P,CD8702 EB 010006
570 DATA ,Q,CDWY00 10RQ 10FE E5
580 DATA ,R,CDWY00 E5 DF E1 20RR E1 F1 FE03 20RW 2B 46 23
590 DATA ,S,23 23 10RS
600 DATA ,T,10FE 46 23 7E 23 E5 66 6F 78 B7 28RV
610 DATA ,U,CDWY00 10RU
620 DATA ,V,E1 23 E5 DF E1 20RT
630 DATA ,W,79 ED44 CD6402 18RK
640 DATA ,X,CD3502 77 23 81 4F C9
650 DATA ,Y,7E CD6402 23 81 4F C9
660 DATA ,Z,CD7F0A 44 4D EB 2AFB40 DF 30RN 2AFD40 DF 38RN 2AFB40
 C3W100
670 DATA 7FE0,1,E5 23 23 23 7F 23 85 5F 7C 8E 57 13 60 69 DF EB
D1 30RI AF CD1202 C9
680 DATA ,*,00
690 DATA END
```

**end**

```
7F22 2804      00670          JR      Z,NOCOMP    ;YES, NO COMPARE
7F24 1A        00680          LD      A,(DE)      ;ARRAY DATA
7F25 BE        00690          CP      (HL)        ;CHECK IF SAME AS TAPE
7F26 2062      00700          JR      NZ,ERSTP2   ;NO, ERROR RETURN
7F28           00710 NOCOMP   EQU     $
7F28 13        00720          INC     DE          ;NEXT ARRAY BYTE
7F29 10EC      00730          DJNZ    RDNXH       ;REPEAT IF MORE HEADER
7F2B E1        00740          POP     HL          ;ARRAY END ADDRESS
7F2C D5        00750          PUSH    DE          ;ARRAY START + 6
7F2D EB        00760          EX      DE,HL       ;START TO HL, END TO DE
7F2E           00770 RDLP     EQU     $
7F2E CDD67F    00780          CALL    RDSUM       ;NOW READ ARRAY DATA
7F31 E5        00790          PUSH    HL          ;READ AND SUM
7F32 DF        00800          RST     CPDEHL      ;STORE NEXT ADDR
7F33 E1        00810          POP     HL          ;CHECK FOR ARRAY END
7F34 20F8      00820          JR      NZ,RDLP     ;RESTORE NEXT ADDR
7F36 E1        00830          POP     HL          ;NOT END, REPEAT READ
7F37 F1        00840          POP     AF          ;ARRAY DATA START
7F38 FE03      00850          CP      CHRARY      ;VAR TYPE IN A
7F3A 2039      00860          JR      NZ,RDCSUM   ;CHECK CHARACTER ARRAY
7F3C 2B        00870          DEC     HL          ;NO, READ CHECKSUM
7F3D 46        00880          LD      B,(HL)      ;ADDR OF # OF DIM
7F3E 23        00890          INC     HL          ;# DIM TO B
7F3F           00900 BYPLN    EQU     $           ;BACK TO START
7F3F 23        00910          INC     HL          ;BYPASS DIM LENGTH CODES
7F40 23        00920          INC     HL          ;2 BYTES FOR EACH DIM
7F41 10FC      00930          DJNZ    BYPLN       ;REPEAT FOR EACH DIM
7F43           00940 NXTENT   EQU     $           ;NEXT ELEMENT ENTRY
7F43 46        00950          LD      B,(HL)      ;STRING LENGTH
7F44 23        00960          INC     HL          ;OLD STRING ADDR LSB
7F45 D5        00970          PUSH    DE          ;STORE ARRAY END
7F46 E5        00980          PUSH    HL          ;STORE ACTUAL ADDR
7F47 2AA040    00990          LD      HL,(CHREND) ;END OF STRING AREA
7F4A 3AD640    01000          LD      A,(CHRNXT)  ;LSB OF NEXT AVAIL POS
7F4D 90        01010          SUB     B           ;SUBTRACT LENGTH
7F4E 5F        01020          LD      E,A         ;STORE RESULT
7F4F 3AD740    01030          LD      A,(CHRNXT+1) ;MSB OF NEXT POS
7F52 DE00      01040          SBC     A,0         ;ADJUST FOR BORROW
7F54 57        01050          LD      D,A         ;AND STORE
7F55 DF        01060          RST     CPDEHL      ;CHECK IF ROOM FOR STRING
7F56 EB        01070          EX      DE,HL       ;NEW NEXT TO HL
7F57 3031      01080          JR      NC,ERSTP2   ;NO, ERROR RETURN
7F59 22D640    01090          LD      (CHRNXT),HL ;STORE NEW NEXT
7F5C 23        01100          INC     HL          ;ADDR FIRST DATA BYTE
7F5D EB        01110          EX      DE,HL       ;ADDR TO DE
7F5E E1        01120          POP     HL          ;ARRAY ENTRY ADDR
7F5F 73        01130          LD      (HL),E      ;LSB OF NEW ADDR
7F60 23        01140          INC     HL          ;TO ADDR MSB
7F61 72        01150          LD      (HL),D      ;STORE MSB
7F62 23        01160          INC     HL          ;TO NEXT ELEM IN ARRAY
7F63 E5        01170          PUSH    HL          ;STORE ADDR
7F64 EB        01180          EX      DE,HL       ;STRING ADDR TO HL
7F65 78        01190          LD      A,B         ;CHECK STRING LENGTH
7F66 B7        01200          OR      A
7F67 2805      01210          JR      Z,FNXT      ;ZERO, NO TAPE READ
7F69           01220 RDCHLP   EQU     $           ;READ STRING CHARACTER
7F6C CDD67F    01230          CALL    RDSUM       ;READ AND ADD TO CKSUM
7F6C 10FB      01240          DJNZ    RDCHLP      ;REPEAT FOR EACH CHAR
7F6E           01250 FNXT     EQU     $           ;FIND NEXT ELEMENT
7F6E E1        01260          POP     HL          ;NEXT ELEMENT
7F6F D1        01270          POP     DE          ;ARRAY END
7F70 E5        01280          PUSH    HL
7F71 DF        01290          RST     CPDEHL      ;CHECK IF ARRAY END
```

# Listing 2

```
                00100 ;
                00110 ; ARRAY I/O ROUTINE FOR TRS-80 LEVEL II TAPE SYSTEM
                00120 ; PROGRAMMED BY ARNE ROHDE, STRUER, DENMARK, MAY 1980
                00130 ;
                00140 ; THE ROUTINE CAN BE USED FOR WRITING BASIC ARRAYS TO TAPE
                00150 ; AND FOR READING THE ARRAYS INTO MEMORY FROM TAPE.
                00160 ; ALL VALID ARRAYS CAN BE READ AND WRITTEN, BUT WHEN READING
                00170 ; FROM TAPE THE ARRAY MUST BE DEFINED WITH THE SAME DIMENSIO N
NS
                00180 ; AND ELEMENTS AS WHEN IT WAS WRITTEN.
                00190 ; NOTE THAT STRING ARRAYS MAY REQUIRE MORE STRING AREA WHEN
                00200 ; READ THAN WHEN THEY ARE WRITTEN.  THE STRING AREA SHOULD
                00210 ; BE COMPACTED BEFORE READING STRING ARRAYS.
                00220 ;
                00230 ; DEFINITIONS
0003            00240 CHRARY  EQU     3               ;VAR TYPE CHARACTER
0A7F            00250 GVARPT  EQU     0A7FH           ;GET USR PARAMETER VALUE
0A9A            00260 RETBAS  EQU     0A9AH           ;RETURN TO BASIC WITH VALUE
0212            00270 TPON    EQU     212H            ;SWITCH TAPE MOTOR ON
0296            00280 RDLDR   EQU     296H            ;FIND TAPE LEADER
0235            00290 RDBYTE  EQU     235H            ;READ BYTE FROM TAPE
0287            00300 WRTLDR  EQU     287H            ;WRITE TAPE LEADER
02c4            00310 WRBYTE  EQU     2c4H            ;WRITE BYTE TO TAPE
01F8            00320 TPOFF   EQU     1F8H            ;SWITCH TAPE OFF
0018            00330 CPDEHL  EQU     18H             ;RST FOR CMP DE,HL
40A0            00350 CHREND  EQU     40A0H           ;END OF STRING AREA
40D6            00360 CHRNXT  EQU     40D6H           ;NEXT AVAIL STRING BYTE
40FB            00370 ARYST   EQU     40FBH           ;ARRAY AREA START
40FD            00380 ARYEND  EQU     40FDH           ;ARRAY AREA END
3C3E            00390 ASTER   EQU     3C3EH           ;SCREEN ASTERISK POS
                00400 ;
408E            00410         ORG     408EH           ;USR ADDRESS POINTER
408E 007F       00420         DEFW    WRTARY          ;ENTRY FOR WRITE
                00430 ;
                00440 ;START ADDRESS OF WRITE - FORCED TO LSB=00
7F00            00450         ORG     7F00H           ;FOR 16K SYSTEM
7F00            00460 WRTARY  EQU     $               ;ENTRY POINT FOR WRITE
7F00 CDE67F     00470         CALL    ARAYA           ;GET ARRAY START AND END
7F03 1A         00480         LD      A,(DE)          ;ARRAY TYPE
7F04 F5         00490         PUSH    AF              ;STORE IT
7F05 C3917F     00500         JP      WRTAP           ;CONTINUE WRITE
                00510 ;READ ENTRY - LSB FORCED TO 08
7F08            00520 RDARY   EQU     $               ;ENTRY POINT FOR READ
7F08 CDE67F     00530         CALL    ARAYA           ;GET ARRAY START AND END ADR
S
7F0B CD9602     00540         CALL    RDLDR           ;READ TAPE LEADER

7F0E 010006     00550         LD      BC,0600H        ;# BYTES + CHKSUM CLEAR
7F11 1A         00560         LD      A,(DE)          ;VARIABLE TYPE
7F12 F5         00570         PUSH    AF              ;STORE IT
7F13 E5         00580         PUSH    HL              ;STORE ARRAY END
7F14 21F87F     00590         LD      HL,HDRST        ;TEMP STORE AREA
7F17            00600 RDNXH   EQU     $               ;READ NEXT HEADER BYTE
7F17 CDD67F     00610         CALL    RDSUM           ;READ AND ADD TO CKSUM
7F1A 2B         00620         DEC     HL              ;BACK TO BYTE READ
7F1B 78         00630         LD      A,B             ;BYTE NUMBER
7F1C FE05       00640         CP      5               ;PART OF NAME
7F1E 2808       00650         JR      Z,NOCOMP        ;YES, NO COMPARE
7F20 FE04       00660         CP      4               ;STILL NAME
```

*(Continued on top of page 34)*

```
7F72 E1         01300         POP     HL
7F73 20CE       01310         JR      NZ,NXTENT       ;NOT END, REPEAT
7F75            01320 RDCSUM  EQU     $               ;READ CHECKSUM
7F75 CD3502     01330         CALL    RDBYTE          ;READ BYTE
7F78 81         01340         ADD     A,C             ;ADD TO CALC SUM
7F79 2011       01350         JR      NZ,ERRSTP       ;NON-ZERO, ERROR
7F7B 212020     01360         LD      HL,2020H        ;TWO SPACES
7F7E 223E3C     01370         LD      (ASTER),HL      ;CLEAR ASTERISKS ON SCREEN
7F81            01380 RETOK   EQU     $               ;RETURN ON OK
7F81 210000     01390         LD      HL,0            ;SET OK INDICATOR
7F84            01400 STPRET  EQU     $               ;STOP TAPE AND RETURN
7F84 CDF801     01410         CALL    TPOFF           ;SWITCH OFF
7F87 C39A0A     01420         JP      RETBAS          ;RETURN WITH HL
7F8A            01430 ERSTP2  EQU     $               ;ERROR, 2 POPS
7F8A E1         01440         POP     HL
7F8B            01450 ERSTP1  EQU     $               ;ERROR, 1 POP
7F8B E1         01460         POP     HL

7F8C            01470 ERRSTP  EQU     $               ;ERROR RETURN
7F8C 21FFFF     01480         LD      HL,0FFFFH       ;ERROR INDICATION
7F8F 18F3       01490         JR      STPRET          ;STOP TAPE AND RETURN
                01500 ;
                01510 ; ROUTINE TO WRITE ARRAY TO TAPE
7F91            01520 WRTAP   EQU     $
7F91 CD8702     01530         CALL    WRTLDR          ;WRITE TAPE LEADER
7F94 EB         01540         EX      DE,HL           ;START TO HL, END TO DE
7F95 010006     01550         LD      BC,0600H        ;# BYTES + CHECKSUM CLEAR
7F98            01560 WRTPRE  EQU     $               ;WRITE PREAMBLE
7F98 CDDE7F     01570         CALL    WCHSUM          ;WRITE CHAR + CAL CKSUM
7F9B 10FB       01580         DJNZ    WRTPRE          ;REPEAT FOR 6 BYTES
7F9D 10FE       01590         DJNZ    $               ;PAUSE, TIME FOR READ
7F9F E5         01600         PUSH    HL              ;NEXT BYTE ADDR
7FA0            01610 NXDBY   EQU     $               ;NEXT DATA BYTE
7FA0 CDDE7F     01620         CALL    WCHSUM          ;WRITE CHAR, ADD TO CKSUM
7FA3 E5         01630         PUSH    HL              ;TEMP STORE
7FA4 DF         01640         RST     CPDEHL          ;CHECK FOR END
7FA5 E1         01650         POP     HL              ;RESTORE
7FA6 20F8       01660         JR      NZ,NXDBY        ;NOT END, REPEAT
7FA8 E1         01670         POP     HL              ;DATA START
7FA9 F1         01680         POP     AF              ;ARRAY TYPE
7FAA FE03       01690         CP      CHRARY          ;CHECK CHAR ARRAY
7FAC 2020       01700         JR      NZ,WRTCSM       ;NO, WRITE CHECKSUM
7FAE 2B         01710         DEC     HL              ;GET # OF DIMENSIONS
7FAF 46         01720         LD      B,(HL)          ;# OF DIM
7FB0 23         01730         INC     HL              ;BACK TO START
7FB1            01740 BYPSLN  EQU     $               ;BYPASS LENGTH CODES
7FB1 23         01750         INC     HL              ;2 BYTES PER DIM
7FB2 23         01760         INC     HL
7FB3 10FC       01770         DJNZ    BYPSLN          ;REPEAT FOR EACH DIM
7FB5            01780 NXTENT  EQU     $               ;NEXT ENTRY UPDATE
7FB5 10FE       01790         DJNZ    $               ;PAUSE FOR EACH STRING
7FB7 46         01800         LD      B,(HL)          ;STRING LENGTH
7FB8 23         01810         INC     HL
7FB9 7E         01820         LD      A,(HL)          ;LSB OF STRING ADDR
7FBA 23         01830         INC     HL
7FBB E5         01840         PUSH    HL              ;STORE ADDR
7FBC 66         01850         LD      H,(HL)          ;MSB OF STRING ADDR
7FBD 6F         01860         LD      L,A             ;ADDR TO HL
7FBE 78         01870         LD      A,B             ;STRING LENGTH
7FBF B7         01880         OR      A               ;CHECK FOR ZERO
7FC0 2805       01890         JR      Z,NOWCH         ;ZERO, NO WRITE
7FC2            01900 WRNXCH  EQU     $               ;WRITE NEXT CHAR
7FC2 CDDE7F     01910         CALL    WCHSUM          ;WRITE BYTE, ADD TO CKSUM
7FC5 10FB       01920         DJNZ    WRNXCH          ;REPEAT FOR EACH BYTE
```

*(Continued on next page)*

# A Cassette I/O Routine (Continued)

```
7FC7            01930 NOWCH   EQU     $
7FC7 E1         01940         POP     HL              ;STORED ADDR
7FC8 23         01950         INC     HL              ;ADDR OF NEXT
7FC9 E5         01960         PUSH    HL              ;TEMP STORE
7FCA DF         01970         RST     CPDEHL          ;CHECK FOR END
7FCB E1         01980         POP     HL              ;RESTORE ADDR
7FCC 20E7       01990         JR      NZ,NXTENU       ;NOT END, REPEAT
7FCE            02000 WRTCSM  EQU     $               ;WRITE CHECKSUM
7FCE 79         02010         LD      A,C             ;CALCULATED SUM
7FCF ED44       02020         NEG                     ;2'S COMPLEMENT
7FD1 CD6402     02030         CALL    WRBYTE          ;WRITE SUM
7FD4 18AB       02040         JR      RETOK           ;RETURN TO BASIC
                02050 ;
                02060 ; READ CHARACTER AND ADD TO CHECKSUM
7FD6            02070 RDSUM   EQU     $
7FD6 CD3502     02080         CALL    RDBYTE          ;READ BYTE
7FD9 77         02090         LD      (HL),A          ;STORE VALUE
7FDA 23         02100         INC     HL              ;READY FOR NEXT
7FDB 81         02110         ADD     A,C             ;ADD TO CHECKSUM
7FDC 4F         02120         LD      C,A             ;RESTORE SUM
7FDD C9         02130         RET
                02140 ;
                02150 ; WRITE BYTE AND ADD TO CHECKSUM
7FDE            02160 WCHSUM  EQU     $
7FDE 7E         02170         LD      A,(HL)          ;CHAR TO WRITE
7FDF CD6402     02180         CALL    WRBYTE          ;WRITE TO TAPE
7FE2 23         02190         INC     HL              ;ADDR OF NEXT
7FE3 81         02200         ADD     A,C             ;ADD TO CHECKSUM
7FE4 4F         02210         LD      C,A             ;RESTORE SUM
7FE5 C9         02220         RET
                02230 ;
                02240 ; GET ARRAY START AND END ADDRESSES
7FE6            02250 ARAYA   EQU     $
7FE6 CD7F0A     02260         CALL    GVARPT          ;GET VARPTR VALUE
7FE9 44         02270         LD      B,H             ;VARPTR TO BC
7FEA 4D         02280         LD      C,L
7FEB EB         02290         EX      DE,HL           ;VARPTR TO DE
7FEC 2AFB40     02300         LD      HL,(ARYST)      ;START OF ARRAY AREA
7FEF DF         02310         RST     CPDEHL          ;CHECK IF VARPTR LOWER
7FF0 3099       02320         JR      NC,ERSTPI       ;YES, ERROR
7FF2 2AFD40     02330         LD      HL,(ARYEND)     ;END OF ARRAY AREA
7FF5 DF         02340         RST     CPDEHL          ;CHECK VARPTR HIGHER
7FF6 3893       02350         JR      C,ERSTPI        ;YES, ERROR
7FF8 2AFB40     02360         LD      HL,(ARYST)      ;ARRAY START
7FFB C3E07E     02370         JP      NXTV            ;CONTINUE PROCESS
                02380 ;
                02390 ; ARAYA CONTINUED - ORG BACK BEFORE ENTRY ADDRESSES
                02400 ;       ADDRESS FOR 16 K SYSTEM
7EE0            02410         ORG     7EE0H           ;CHANGE OR DELETE IF CODE MO
VED
7EE0            02420 NXTV    EQU     $               ;NEXT ARRAY
7EE0 E5         02430         PUSH    HL              ;STORE ADDRESS
7EE1 23         02440         INC     HL              ;ADDR OF LENGTH
7EE2 23         02450         INC     HL
7EE3 23         02460         INC     HL
7EE4 7E         02470         LD      A,(HL)          ;LSB OF LENGTH
7EE5 23         02480         INC     HL
7EE6 85         02490         ADD     A,L             ;ADD ADDR TO LSB
7EE7 5F         02500         LD      E,A             ;RESULT TO E
7EE8 7C         02510         LD      A,H             ;MSB OF ADDR
7EE9 8E         02520         ADC     A,(HL)          ;ADD MSB OF LEN
7EEA 57         02530         LD      D,A             ;END ADDR -1 IN DE
7EEB 13         02540         INC     DE              ;END ADDR
```

# A SLIC Label Program

```
!PLABEL.SLI     PRINT MAILING LABELS    9/17/80
!Written by:
!       Bob Hallem
!       RTG Data Systems
!       1003 Wilshire Blvd., Suite 202
!       Santa Monica, CA  90401
!Each entry in name-and-address file has this format:
!       last_name, first_name
!       address-1
!       address-2               [optional]
!       city, state
!       zip
!        c0,c1,...,c9           [Note leading space]
!       [Additional lines beginning with space (ignored)]
!       [1 or more empty lines]
!The labels are printed in this format:
!       first_name last_name
!       address-1
!       address-2               [if used]
!       city, state
!                       zip
!Routine @tab positions printer to 1st column of label.
!Currently assumes one label per line ("one-up").
!Routine @select determines which are sent to output file.



long @atol;     !declare built-in function returning long integer

@global{
        int READ,WRITE,ERR,EOF,OK;
        int YES,NO;
        int out;        !output unit for labels
        !for CP/M:
        char fcb1(36),buf1(128),fcb2(36),buf2(128);
        !data for one label
        char name(31),first(31),addr1(31),addr2(31),city(31);
        char zip(81),code(81);
        int person;     !YES if individual's name
        int LBL;        !lines between labels (top-to-top)
}

@g{
        set READ=0,WRITE=1,ERR=-2,EOF=-1,OK=1;
        set YES=1,NO=0;
        set LBL=6;
}



@main{
        int status;
        int val(10);    !code values
```

```
7EEC 60        02550          LD      H,B             ;VARPTR TO HL
7EED 69        02560          LD      L,C
7EEE DF        02570          RST     CPDEHL          ;COMPARE END AND VARPTR
7EEF EB        02580          EX      DE,HL           ;END TO HL
7EF0 D1        02590          POP     DE              ;START TO DE
7EF1 30ED      02600          JR      NC,NXTV         ;NOT IN RANGE, REPEAT
7EF3 AF        02610          XOR     A               ;ON-BOARD CASSETTE ADDR
7EF4 CD1202    02620          CALL    TPON            ;SWITCH ON
7EF7 C9        02630          RET
0001           02640 HDRST    DEFS                    ;TEMP STORE ARRAY READ
0000           02650          END
00000 TOTAL ERRORS



ARAYA  7EE6 02250   00470 00530
ARYEND 40FD 00380   02330
ARYST  40FB 00370   02300 02360
ASTER  3C3E 00390   01370
BYPLN  7F3F 00900   00930
BYPSLN 7FB1 01740   01770
CHRARY 0003 00240   00850 01690
CHREND 40A0 00350   00990
CHRNXT 40D6 00360   01000 01030 01090
CPDEHL 0018 00330   00800 01060 01290 01640 01970 02310 02340
                    02570
ERRSTP 7F8C 01470   01350
ERSTP1 7F8B 01450   02320 02350
ERSTP2 7F8A 01430   00700 01080
FNXT   7F6E 01250   01210
GVARPT 0A7F 00250   02260
HDRST  7EF8 02640   00590
NOCOMP 7F28 00710   00650 00670
NOWCH  7FC7 01930   01890
NXDBY  7FA0 01610   01660
NXTENT 7F43 00940   01310
NXTENU 7FB5 01780   01990
NXTV   7EE0 02420   02370 02600
RDARY  7F08 00520
RDBYTE 0235 00290   01330 02080
RDCHLP 7F09 01220   01240
RDCSUM 7F75 01320   00860
RDLDR  0296 00280   00540
RDLP   7F2E 00770   00820
RDNXH  7F17 00600   00730
RDSUM  7FD6 02070   00610 00780 01230
RETBAS 0A9A 00260   01420
RETOK  7F81 01380   02040
STPRET 7F84 01400   01490
TPOFF  01F8 00320   01410
TPON   0212 00270   02620
WCHSUM 7FDE 02160   01570 01620 01910
WRBYTE 02D4 00310   02030 02180
WRNXCH 7FC2 01900   01920
WRTAP  7F91 01520   00500
WRTARY 7F00 00460   00420
WRTCSM 7FCE 02000   01700
WRTLDR 0287 00300   01530
WRTPRE 7F98 01560   01580
```

**end**

```
        @g;             !initialize global variables
        @clear;         !clear the screen
        type "PRINT MAILING LABELS   V1.0";
        @files;
        ask #6;         !input from file on unit 6
        type #out,"\r"; !return without line feed



                        !output to video, printer, or file
        while( @ask(name)#EOF ){        !read line from input
                !check for blank line ('\e' is end-of-string)
                ! or leading space
                if( name(0)='\e' | name(0)=' ' )
                        next;           !and ignore
                @doname;                !adjust if "last,first"
                set status=@doaddr;     !read the rest
                if(status=EOF){
                        type #1,"\nUNEXPECTED END-OF-FILE";
                        @exit;
                }
                if(status=ERR){
                        type #1,"\nNO CODE LINE";
                        @exit;
                }
                @docode(val);           !interpret code line
                !The following call puts name & address and
                !codes on screen so we have something to watch.
                @show(val);
                if( @select(val)=YES ){
                        type #out;      !select output device
                        @label;         !print a label
                }
        }
        if( @close(out)<0 )
                type #1,"\nClose error.";
}



@files{
        char name(20);
        type "\n\nEnter name-and-address filename: ";
        if( @ask(name)<0 ){
                type "\n[EXIT]";
                @exit;
        }
        if( @open(name,6,READ,fcb1,buf1)=ERR ){
                type "\nCannot open "; @type(name);
                @exit;
        }
        type "\nChoose name for label file",
         "\n (VD: is video, PR: is printer): ";
        ask #0;         !restore keyboard as input device
```

*(Continued on next page)*

# A SLIC Label Program (Continued)

```
        if( @ask(name)<0 ){
                type "\n[EXIT]";
                @exit;
        }
        if( name(0)='P' & name(1)='R' & name(2)=':' )
                set out=3;          ! unit 3 is printer
        else if( name(0)='V' & name(1)='D' & name(2)=':' )
                set out=1;          ! unit 1 is video display
        else
                set out=7;          !open a disk file on unit 7
        if( @open(name,out,WRITE,fcb2,buf2)=ERR ){



                type "\nCannot open "; @type(name);
                @exit;
        }
}

!Process name line, putting first & last names in separate
!strings if a comma is found.
@doname{
        int posn;       !position of comma in name
        int j,k;
        set posn=@index(name,',');      !look for comma
        if(posn=ERR)                    !no comma
                set person=NO;  !not a person
        else{
                set person=YES; !a person
                set name(posn)='\e';    !terminate last name
                for(set k=0,j=1+posn; set k=k+1,j=j+1; name(j)#'\e')
                        set first(k)=name(j);
                set first(k)='\e';      !terminate string
                @ltrim(first);  !remove leading blanks
        }
}



!Read 1 or 2 address lines, city & state, zip, and code line
@doaddr{
        if( @ask(addr1)=EOF )   !address line
                return(EOF);
        if( @ask(addr2)=EOF )   !address (or city-state)
                return(EOF);
        if( @ask(city)=EOF )    !city-state (or zip)
                return(EOF);
        if( @ask(zip)=EOF )     !zip (or code line)
                return(EOF);
        if( zip(0)=' '){        !code line?
                @scopy(zip,code);       !yes, no "addr2" line
                @scopy(city,zip);
                @scopy(addr2,city);
                set addr2="";           !null string
        }
```

```
!Print the label.  If we had more than one label per line,
!we could save them up here, then print an entire line.
!But be sure to take out the extra call to @label from @show.
!We also could (should!) check the length of each line.
@label{
        int lines;      ! # of lines printed
        @tab;           !move to start of label
        if(person=YES){
                @type(first);
                @space(1);
                @type(name);    !last name only
        }
        else
                @type(name);
        @nl;                    !print newline
        @tab; @type(addr1); @nl;
        if( addr2='\e' )        !no 2nd line
                set lines=4;
        else{
                @tab; @type(addr2); @nl;
                set lines=5;
        }
        @tab; @type(city); @nl;
        @tab; @space(12); @type(zip); @nl;
        while(lines<LBL){       !go to top of next label
                @nl;
                set lines=lines+1;
        }
}
```

```
!-----------------------------------
! Utility Functions
!-----------------------------------

!find "c" in string "s"
char s(); int c;
@index(s,c){
        int i;
        for(set i=0; set i=i+1; s(i)#'\e')
                if( s(i)=c )
                        return(i);
        return(ERR);
}

!remove leading blanks
char s();
@ltrim(s){
```

```
        else{                       !haven't read code line yet
                if( @ask(code)=EOF )
                        return(EOF);
                if( code(0)#' ' )       !this is required
                        return(ERR);
        }
}

!Extract up to 10 code values from code line.
int val();      !declare argument to be integer array
@docode(val){
        int i,k;
        set i=0;        !index into string
        do(k,0,9)
                set val(k)=@atov(code,i);  !ASCII to variable
}




!Display name & address, and codes
int val();
@show(val){
        int k;
        if(out=1)       !output device is VD:
                return;
        type #1,"CODES:";
        do(k,0,9)
                type %F2,val(k);
        @nl;
        @label;
}




!Decide whether to print this name and address
!Returns YES to print, NO to skip
int val();
@select(val){
        int i;
        long zipval;    !"long" is 32- or 48-bit integer
        set i=0, zipval=@atol(zip,i);
        !we could use "zipval" to select a range of zips
        !we'll just select out some codes for now
        if( val(0)>1 & val(1)=0 )
                return(NO);
        else
                return(YES);
}
```

*(Continued on top of page 38)*

```
        int i,j,k;
        for(set k=0; set k=k+1; s(k)=' ')
                ;
        !s(k) is non-blank
        if(k=0)
                return;
        for(set i=0,j=k; set i=i+1,j=j+1; s(j)#'\e')
                set s(i)=s(j);
        set s(i)='\e';
}

!copy string s1 to string s2
char s1(),s2();
@scopy(s1,s2){
        int i;
        for( set i=0; set i=i+1; s1(i)#'\e' )
                set s2(i)=s1(i);
        set s2(i)='\e';
        return(i);
}

int k;
@space(k){
        int i;
        do(i,1,k)
                type " ";
}

!print newline
!neater than having 'type "\n";' all over the place
@nl{
        type "\n";
}

!move over to start of label
@tab{
        @space(22);
}
```

**end**

# Polynomial Evaluation in Basic

## Listing 1

```
 10 REM
 20 PRINT : PRINT : PRINT
 30 PRINT"       POLYNOMIAL EVALUATION"
 40 PRINT
 50 REM **** Get parameters from user ********************************
 60 REM
 70 INPUT"ENTER ORDER OF POLYNOMIAL;     N=",N
 80 DIM A(N),V(N)
 90 PRINT
100 PRINT"ENTER COEFFICIENTS:"
110  FOR I=N TO 0 STEP-1
120  PRINT"  FOR X^";I;
130  INPUT" = ",A(I)
140  NEXT I
150 PRINT
160 INPUT"ENTER POINT OF EVALUATION;    X1=",X1
170 PRINT : PRINT
180 REM **** Copy vector A into vector V ****************************
190 REM
200  FOR I=0 TO N
210  V(I)=A(I)
220  NEXT I
230 GOSUB 1000
240 REM **** Printout results and branch back ***********************
250 REM
260 PRINT"POLYNOMIAL   = ";V(0)
270  FOR I=1 TO N
280  PRINT"DERIVATIVE ";I;" = ";V(I)
290  NEXT I
300 PRINT
310 INPUT"FOR A NEW X1 ENTER 'X'; FOR ANOTHER POLYNOMIAL HIT RETURN ",A$
320 IF A$="X"THEN 150
330 GOTO 40
340 STOP
1000 REM **** SUBROUTINE - evaluate polynomial and derivatives ******
1010 REM ************************************************************
1020 REM
1030 REM **** Calculate the R(i) series into vector V ***************
1040 REM
1050  FOR I=0 TO N-1
1060   FOR J=N-1 TO I STEP-1
1070   V(J)=V(J+1)*X1+V(J)
1080   NEXT J
1090  NEXT I
```

```
DERIVATIVE 3 = 1353
DERIVATIVE 4 = 996
DERIVATIVE 5 = 360

FOR A NEW X1 ENTER 'X'; FOR ANOTHER POLYNOMIAL HIT RETURN X


ENTER POINT OF EVALUATION;    X1=2.4


POLYNOMIAL   = 348.94112
DERIVATIVE 1 = 689.288
DERIVATIVE 2 = 1077.12
DERIVATIVE 3 = 1255.2
DERIVATIVE 4 = 960
DERIVATIVE 5 = 360


FOR A NEW X1 ENTER 'X'; FOR ANOTHER POLYNOMIAL HIT RETURN X


ENTER POINT OF EVALUATION;    X1=2.6


POLYNOMIAL   = 510.07968
DERIVATIVE 1 = 931.12
DERIVATIVE 2 = 1347.84
DERIVATIVE 3 = 1454.4
DERIVATIVE 4 = 1032
DERIVATIVE 5 = 360

FOR A NEW X1 ENTER 'X'; FOR ANOTHER POLYNOMIAL HIT RETURN


ENTER ORDER OF POLYNOMIAL;     N=4

ENTER COEFFICIENTS:
  FOR X^4 = 3
```

```
1100 REM **** Multiply each term by i! ********************************
1110 REM
1120 F=1
1130  FOR I=2 TO N
1140  F=F*I
1150  V(I)=V(I)*F
1160  NEXT I
1170 RETURN
```

**end**

# Listing 2

```
>>RUN


       POLYNOMIAL EVALUATION

ENTER ORDER OF POLYNOMIAL;      N=5

ENTER COEFFICIENTS:
  FOR X^5 = 3
  FOR X^4 = 4
  FOR X^3 = -2
  FOR X^2 = 0
  FOR X^1 = 5
  FOR X^0 = -7

ENTER POINT OF EVALUATION;     X1=2.5


POLYNOMIAL    = 423.46875
DERIVATIVE 1 = 803.4375
DERIVATIVE 2 = 1207.5
```

*(Continued on top of page 40)*

```
  FOR X^3 = 2
  FOR X^2 = -1
  FOR X^1 = 2
  FOR X^0 = -5


ENTER POINT OF EVALUATION;     X1=2


POLYNOMIAL    = 59
DERIVATIVE 1 = 118
DERIVATIVE 2 = 166
DERIVATIVE 3 = 156
DERIVATIVE 4 = 72

FOR A NEW X1 ENTER 'X'; FOR ANOTHER POLYNOMIAL HIT RETURN X

ENTER POINT OF EVALUATION;     X1=1.999


POLYNOMIAL    = 58.882082974003
DERIVATIVE 1 = 117.834077988
DERIVATIVE 2 = 165.844036
DERIVATIVE 3 = 155.928
DERIVATIVE 4 = 72

FOR A NEW X1 ENTER 'X'; FOR ANOTHER POLYNOMIAL HIT RETURN

ENTER ORDER OF POLYNOMIAL;      N=
♦♦♦60 ESCAPE♦♦♦

>>
```

**end**

# Of Interest

by Dave Cortesi

## Phillip Woellhof Spins Off

Mr. Woellhof, formerly Director of New Products for Lifeboat Associates, has apparently contracted a case of Spinoff-itis, an occupational hazard more common in Silicon Valley than in Connecticut where he has founded **Westico**, a new software distribution house. Westico aims to distinguish itself through overnight delivery and technical support that is "fast, accurate, and courteous." And more power to him, say we.

## Your Whole Summer is Booked Already

. . . or it would be if you attended every interesting conference we've been told about. If you hurry, you might be able to catch the **Southwest Computer Show** at the Dallas Market Center in Dallas, April 9-12. $50 million worth of hardware will be on display, plus the usual robot and computer art sideshows.

Readers in Michigan might like the more modest Annual Computer Show presented by the **Saginaw Valley Chapter of DPMA**, to be held May 7-8 in Midland. East Coast TRS-80 users may prefer the First Annual **TRS-80 Microcomputer Show**, May 22-24 at the Statler Exposition Center in New York City. The show, which isn't sponsored by Radio Shack or Tandy, will feature wares related to all the Tandy computer products.

If you are into graphics, plan to make the annual conference of the **National Computer Graphics Association**, June 14-18 at the Baltimore Convention Center. The last day will be a theme day for Personal Computer Graphics, and a number of rooms in the exhibition area will be set aside for personal computers running graphics applications.

Dealers and distributors in the computer trade have been invited to the **Computer Industry Trade Expo**, sponsored by the Microcomputer Industry Trade Association (MITA) and scheduled for June 24-26 in the Atlantic City NJ Convention Center. Dealers and users both are wanted at the **National Small Computer Show**, August 26-29 in the New York Coliseum.

## Cleverest New Product This Month

That's how an announcement from **Data Node, Inc.** struck us. They've built a board that installs in the common DEC VT100 terminal, converting it from a mere terminal into a personal computer with networking abilities. The board costs $1850, contains a Z-80 and 64K of RAM, and comes with networking software so that it can communicate with a PDP-11. The erstwhile terminal can run either CP/M or MP/M.

## An S-100 Do-All Board

**Digicomp Research**, like Godbout and others, have designed an S-100 board that handles all the miscellaneous I/O functions needed by an MP/M system. Theirs contains four serial ports with independent baud rates, 24 bits of parallel I/O, five timers, an interrupt controller and a clock-calendar with battery backup. Evaluation units should be available now, but the writer of the release neglected to mention the price.

## An IEEE-488 Bus Driver

The instrumentation bus protocol brought to the micro community's attention by the Commodore PET's use of it has acquired another supporter. **Ziatech Corp.** have a board designed for use in Multibus-based machines that can drive an IEEE-488 bus at as much as 250 Kbytes per second. The high data rate persuades the company to claim that "the GPIB has replaced RS232 as a local computer-to-computer connection," which qualifies as the month's best example of wishful thinking. The board costs $1200 if you only want one.

## A Sideways L.E.D.

Tiny PC board hardware items fascinate us. The latest is a subminiature LED indicator light that points at ninety degrees to its mounting base, like a beady-eyed little periscope. **Industrial Devices, Inc.** make it, call it their Series 5322, and charge 45 cents each, provided you buy a bunch of them.

## A TRS-80 In Orbit?

**Parasitic Engineering**, makers of add-on drives for TRS-80s, want to sell their Data Separator separately. The unit includes an FD1770 controller chip, plugs into the TRS-80 with no modifications needed, and costs $250. The writer of the release, clearly not an engineer, is most impressed with the phase locked loop circuit used in the device. It's "used in satellite communications, where errors can't be tolerated," we're told. And in acoustic modems, and broadcast FM receivers, and every disk controller but Tandy's . . .

## Useful Reference Cards

We are indebted to **Micro Logic Corp.** for sending us a free copy of their solid-plastic, 8" x 11", 8080 cpu reference card. It shows absolutely anything the hobbyist would need to know in a hurry, from pinouts to instruction timings, and has found a permanent home under the writer's terminal. It even makes neat sounds if you flap it back and forth, as in the old record *Tie Me Kangaroo Down, Sport*. They make one for the 6502 as well. Either is $4.95 plus $1 handling per order.

## Cromemco Announces C

**Cromemco** brags that it is "the only microcomputer manufacturer who currently supplies the C language." That's true, as far as it goes; the others let software vendors supply it. Anyway, their new C compiler runs on the CROMIX multi-user system, produces object modules that can be linked with FORTRAN or COBOL modules, and costs $595. The price includes "extensive documentation." It will be a genuine first if that isn't a reprint of the Kernighan and Ritchie book.

## A New Version of OASIS

An announcement from **Phase One Systems** comes just in time to remind us that Digital Research doesn't have a monopoly on small operating systems. OASIS is available at level 5.5; it has a new spooler and a new backup utility. The spooler supports 26 different file queues and allows for forms set-up and multiple copies. The utility will save only the changed files, and can do data compression. *DDJ*'s editor would like to see some submissions related to OASIS, just for variety.

## An Improved FORTH for Alpha Micro

AM-FORTH is an implementation of that language for use on Alpha Micro computers under the AMOS operating system (hah! there's another one!). **Sierra Computer Co.**, who publish it, have adopted the refreshing custom of designating release levels with letters. Release B of AM-FORTH is improved in that the basic dictionary is re-entrant; a single copy can be shared by all the system's users at once. New licenses are $150; users of version A can have it for $120.

## HUMBUG? Bah!

HUMBUG is said to have been a popular ROM monitor for SS-50 systems. **Star-Kits** have now produced HUMBUG-09, derived from the original but written for the 6809 CPU on the SS-50 bus. The package includes two ROMs, a manual, and your choice of a source listing or the source code itself on diskette. The price is $75, which sounds like a bargain to us. (My editor wonders where all the 68xx experimenters have gone, and why he only gets articles on the Z-80 anymore?)

## An Intriguing Report Writer

A product announcement from **Friends Software** piques our interest. Access/80 is claimed to be a sophisticated report writer and data reorganizing tool for CP/M. It will scan any file, selecting records, sorting them, tabulating and doing sub-totals, and formatting a report to your specifications. The intriguing part is that it claims to be "the microcomputer version of a system running on IBM and Univac compuers," but they don't give the name of the original model. The implementation sounds interesting, too: "Access/80 is built on top of its own virtual machine, 'S380,' and virtual operating system 'VOS' . . . modeled on the operating system used on an IBM 370." A report writer in a virtual IBM operating system in a virtual machine in a CP/M system is something we'd like to see. Or perhaps not.

## Artificial Intelligence Faces The Music

The **MIT Press** publishes a quarterly, *Computer Music Journal*. They'd like you to know that a special two-part issue is just out, in which are surveyed all the ways that Artificial Intelligence research can complement computer music research. Sounds like a must for anyone serious about either topic.

## An OEM Wishbook

This has been a good month for freebies. We are in receipt of a copy of **Sentry Publishing Company**'s *Dataguide*, subtilted "the master catalog and directory of OEM computer products," which pretty well describes it. Each of its 14 product sections is headed by an article analyzing that marketplace, all of which contain bullish forecasts of market growth. The rest of the editorial matter consists of listings of manufacturers; these fill the space between hundreds of display ads. A year's subscription (two issues) is $50. Write them on letterhead paper; who knows, you might get a free copy too.

## An Exhaustive Directory

**Datapro Research Corp.** have been publishing detailed, authoritative descriptions of computer hardware for years. Now they've turned their attention to personal machines and have produced their *Directory of Small Computers*. It contains 200-odd system descriptions, each thorough, all in the same format for easy comparison, plus 900 company profiles and several vendor directories. The two compendious tomes with 12 monthly supplements are not cheap; they'll set you (or, more likely, your employer) back $280. Would that PCC would get one of *these* as a freebie!

## A Software Author's Hopebook

*The 1981 Software Writer's Market* can be had from **Kern Publications**. It lists the names and addresses of "more than 1800 firms who will market and distribute programs for independents." Not every entry will be a winner, but the book's bulk is an indication of the extraordinary breadth and vigor of the independent software market. At $45 a copy it isn't priced for the casual reader, but if you're really in that trade you'll probably find it priceless.

## Free Observations of Industry

The formidable Wayne Green runs **1001001, Inc.** (get it?), which publishes *Microcomputing Industry*, a monthly that analyzes and comments on matters of interest to, well, people in the industry. Subscriptions are "free to qualified subscribers" and (presumably) unattainable to those who aren't. Swipe some letterhead stationery from that friend of yours whose consultancy went broke and write for it; it's an excellent value at the price.

## USC Nerves Itself for the Future

The press release pile yields up a lot of piffle and tosh, which we routinely round-file. University P.R. departments seem to turn out more than their share. These diligent scribes interview a professor, write up his opinions in what they believe to be layman's langauge, and send the result out in hopes of getting the old school a little free ink.

Word has filtered through to USC that chips are all the rage, and they imposed upon Professor John Nelson of the E. E. department to give them a few hot quotes. The awful result is not to be laid at Prof. Nelson's door; the man sounds as if he knows his onions. But someone has surrounded the Professor's words with wide-eyed baby talk, turning it into a prime example of what Ted Nelson calls Cybercrud. Try the taste of this lead paragraph:

> Tomorrow's "superchips" – postage-stamp-size computers that will outperform the massive electronic brains that are currently as big as bank vaults – may enable us to talk to our home appliances and free us from ever again having to commute to work.

Gosh! This sort of Tom Swift journalism with its nattering of "massive electronic brains" is harmless, if irritating. The writer's real sin is his use of anthropomorphic terms:

> And, because superchips will be endowed with "nerve" cells by the millions, Nelson believes superchips can become the humanoid brains necessary for tomorrow's military and consumer robots to engage in "signal processing" – the technical term for sensory skills resembling human speech and sight.

Never mind the technical errors; what's wicked about this kind of gee-whizzery is its deliberate perpetuation of the worst of the public's myths about computers. It goes on and on:

> The original integrated circuits of microelectronics, in the early 1960s, were chips of silicon crystal impregnated with less than 10 metallic, transisitor-like components. Those first models were analogous to single brain cells. Today, with thousands of components, the chips can be compared to specialized brain lobes . . .

No doubt there are small newspaper around the country that will pick up on this turkey as a welcome space-filler. Thus is the cause of computer literacy set back.

*(Continued on page 44, column 1)*

# Electric Phone Book

## Of Interest

*(Continued from page 43)*

## Contact Points

**1001001, Inc.**, Pine Street, Peterborough NH 03458.

**Computer Industry Trade Expo**, 100 Charlotte Place, Englewood Cliffs NJ 07632, (201) 569-8542.

**Cromemco**, 280 Bernardo Ave., Mountain View CA 94043, (415) 964-7600.

**Data Node, Inc.**, 432 Toyama Drive, Sunnyvale CA 94086, (408) 744-0561.

**Datapro Research Corp.**, 1805 Underwood Blvd., Delran NJ 08075, (609) 764-0100.

**Friends Software**, Tioga Building Suite 400, 2020 Milvia St., POB 527, Berkeley CA 94701, (415) 540-7282.

**Industrial Devices, Inc.**, 7 Hudson Avenue, Edgewater NJ 07020, (201) 224-4700.

**Kern Publications**, 190 Duck Hill Road, POB 1029, Duxbury MA 02332, (617) 934-0445.

**Micro Logic Corp.**, Dept. CDJ, POB 174, Hackensack NJ 07602, (201) 342-6518.

**MIT Press**, 28 Carleton Street, Cambridge MA 02142, (617) 253-2889.

**National Computer Graphics Association**, 2033 M Street NW, Suite 330, Washington DC 20036, (202) 466-5895.

**National Small Computer Show**, 110 Charlotte Place, Englewood Cliffs NJ 07632, (201) 569-8542.

**Parasitic Engineering**, 1101 Ninth Avenue, Oakland CA 94606, (415) 839-2636.

**Phase One Systems, Inc.**, 7700 Edgewater Dr. Suite 710, Oakland CA 94621, (415) 562-8085.

**Saginaw Valley Chapter of DPMA** Annual Computer Show at 2914 W. Midland Road, Midland, Michigan; pre-show contact is Don Seidel, (517) 790-4220.

**Sentry Publishing Company**, 5 Kane Industrial Drive, Hudson MA 01749.

**Sierra Computer Company**, 617 Mark NE, Albuquerque NM 87112, (505) 296-8085.

**Star Kits**, POB 209, Mt. Kisco NY 10549.

**TRS-80 Microcomputer Show**: Exhibitors contact Kengore Corporation, 3001 Route 26, Franklin Park NJ 08823, (201) 297-6918.

**Westico**, 25 Van Zant St., Norwalk CT 06855, (203) 853-6880.

**Ziatech Corporation**, 2410 Broad Street, San Luis Obispo CA 93401, (805) 541-0488.

■■

A computerized bulletin board works just like an ordinary bulletin board system except that instead of paper and thumbtacks it uses a terminal, a computer, and the dial-up telephone network.

The list below was developed from several sources including the Peripheral People in Mercer Island, Washington, and the People's Message System in Santee, California. It is being maintained by People's Computer Company's PCNET project, our effort to bring computers and telecommunications into the hands of everyone. While this is the most complete listing we have as of this writing, we would appreciate additions and corrections. Send them to PCNET, PCC, P.O. Box E, Menlo Park, CA 94025.

All the bulletin board systems listed here can be accessed by telephone using a 300-baud ASCII terminal and a Bell 103 modem. Most use carriage-return as a speed recognition character, after which they are self-teaching. All are free to anyone who calls, unlike Arpanet, which is restricted, and The Source and MicroNet, which cost money. The list has been sorted by area code; consult your local telephone directory for geographical correspondence. (Also printed in *Recreational Computing*, March/April 1981).

(201) 283-2724
(201) 457-0893
(201) 688-7117
(201) 753-1225
(201) 753-8152
(201) 835-7228
(201) 843-4563
(201) 874-6833
(201) 891-7441
(201) 968-1074

(202) 337-4694
(202) 635-5730

(203) 348-6353
(203) 357-1920
(203) 746-4644

(205) 945-1489

(206) 244-5438
(206) 246-8983
(206) 482-5134
(206) 482-5590
(206) 524-0203
(206) 546-6239
(206) 723-3282
(206) 937-0444

(209) 638-6392

(212) 245-4363
(212) 448-6576
(212) 787-5520
(212) 997-2186

(213) 276-4276
(213) 316-5706
(213) 329-3715
(213) 340-0135
(213) 346-1849
(213) 349-5728
(213) 360-6332
(213) 394-1505
(213) 395-1592
(213) 396-3905
(213) 424-3506
(213) 428-4718
(213) 459-3177
(213) 459-6400
(213) 566-8035
(213) 631-3186
(213) 657-8803
(213) 673-2206
(213) 675-8803
(213) 709-5423

(213) 787-4004
(213) 795-3788
(213) 799-1632
(213) 799-6514
(213) 826-0325
(213) 828-3400
(213) 843-5390

(214) 288-4859
(214) 634-2668
(214) 634-2775
(214) 641-8759

(216) 644-1965
(216) 745-7855

(301) 344-9156

(303) 759-2625
(303) 789-0936

(305) 261-3639
(305) 566-0805
(305) 689-3234
(305) 772-4444
(305) 821-7401
(305) 989-9647

(309) 688-0470
(309) 694-6531

(312) 255-6489
(312) 337-6631
(312) 359-9450
(312) 420-7995
(312) 545-8086
(312) 622-9609
(312) 729-2403
(312) 782-8180
(312) 941-9009
(312) 964-7768

(313) 288-0335
(313) 357-1422
(313) 465-9531
(313) 477-4471
(313) 484-0732
(313) 569-2063
(313) 588-7054

(314) 838-7784

(316) 746-2078

(319) 353-6528
(319) 557-9618

(404) 394-4220
(404) 733-3461
(404) 790-8614
(404) 793-1045
(404) 939-1520
(404) 939-8429
(404) 953-0723

(405) 353-2554
(405) 528-8009

(408) 241-1956
(408) 296-5799
(408) 263-0248
(408) 263-9650

(414) 241-5406
(414) 241-8364
(414) 282-8118

(415) 348-2139
(415) 348-2396
(415) 493-7691
(415) 527-0400
(415) 647-9524
(415) 661-0705
(415) 683-4703
(415) 792-8406
(415) 851-3453
(415) 948-1474

(417) 862-7852

(419) 865-1584

(502) 245-8288
(502) 896-9624

(503) 646-5510

(512) 657-0779

(513) 671-2753
(513) 874-2283

(515) 279-8863

(516) 938-9043

(523) 223-3672

(602) 866-0258
(602) 956-5612
(602) 957-4428
(602) 957-9282

(604) 687-2640

(607) 754-5571
(607) 797-6416

(609) 983-5970

(612) 561-6311
(612) 929-8966

(614) 272-2759
(614) 649-7097

(615) 254-9193

(617) 354-4682
(617) 388-5125
(617) 431-1699
(617) 649-7097
(617) 692-3973
(617) 864-3819
(617) 897-0346
(617) 963-8310

(702) 826-7234
(702) 873-9491

(703) 281-2125
(703) 281-2222
(703) 379-0303
(703) 620-4990
(703) 734-1387
(703) 750-0930
(703) 893-9474
(703) 978-7561

(707) 448-9055

(713) 233-7943
(713) 693-8080
(713) 977-7019

(714) 449-5689
(714) 463-0461
(714) 495-6458
(714) 526-3687
(714) 537-7913
(714) 565-0961
(714) 571-5550
(714) 582-9557
(714) 730-1206
(714) 739-0711
(714) 751-1422
(714) 772-8868
(714) 898-1984
(714) 952-2110

(714) 962-7979
(714) 963-7222

(801) 375-7000
(801) 466-1737
(801) 753-6800

(802) 748-9089
(802) 879-4981

(803) 270-5372
(803) 270-5392
(803) 279-5392
(803) 771-0922
(803) 772-1592

(804) 340-5246

(805) 484-9904
(805) 527-9321
(805) 682-7876
(805) 964-4115

(806) 355-5610

(813) 223-7688

(816) 523-9121
(816) 531-1050
(816) 861-7040
(816) 931-3135

(817) 855-3916
(817) 855-3918
(817) 923-0009

(901) 276-8196
(901) 362-2222
(901) 761-4743

(902) 794-8198

(904) 243-1257
(904) 243-8565

(913) 362-6398
(913) 764-1520
(913) 782-5115

(915) 584-5393

(916) 393-4459

(918) 224-5347

■■

# Important Features of The PCNET Protocol

PCNET means Personal Computer NET. The PCNET Project is a group of volunteer computer professionals who are working together to create tools that will accelerate the spread of computer-aided communication. Currently we are concentrating upon three tasks:

(a) arranging the publication of the "Electric Phone Book," a list of Bulletin Board Systems, which appears frequently in *Dr. Dobb's Journal* and *Recreational Computing*,

(b) describing and encouraging the use of the PAN protocol for simple electronic messages, and

(c) creating the PCNET protocol, a powerful set of conventions for computers communicating through the telephone system. This remains our most important and significant activity, and is the topic of this article. We will be publishing more details in the future, particularly in *Dr. Dobb's Journal*.

The PCNET Project undertook an unusually large set of design goals, so that the PCNET protocol can serve as the base for a wide variety of communication tasks. We have designed from the start to make this protocol useful to personal computers, and have included the ability for personal computers to communicate with other emerging networks. This article describes some goals of our design, and our current strategy for implementation.

Since computers have limited amounts of storage for incoming messages, messages are broken into packets that carry portions of the message between computers. The receiving computer will put the contents of the packets back together, yielding the original message.

Like all serious protocols for use in an environment where electrical noise introduces errors, the PCNET protocol has a method for sending a checksum with the data, so that errors may be detected. Through the use of counters that indicate which packets are wanted next, and labels telling which have arrived, the protocol will signal when any erroneous packets must be retransmitted.


## by David C. Harris

*David C. Harris, c/o PCNET, 1263 El Camino Real, P.O. Box E, Menlo Park, CA 94025.*


The PCNET protocol attempts to isolate logically separable activites in separate "layers" of the protocol. This technique, widely used and respected in data communications, makes it easier to modify parts of the protocol without damaging other parts at the same time. When optical fibers, satellites, broadcast radio, or cable television become available for computer communication, the PCNET protocol should be easy to adapt to these opportunities.

A user's program can treat the PCNET protocol as an input or output device, issuing "PREAD" and "PWRITE" commands. The PCNET protocol will transmit or receive until error-free information has been communicated, and the return to the user's program. Although the initial versions of the PCNET protocol will be oriented to file and mail transfer, this aspect of the design will allow many imaginative uses to be created relatively easily.

In the inital, "simple" version of the PCNET protocol, two bytes of the packet will be converted to three bytes from a set of 41 characters that may be safely sent even to large mainframe computers that would otherwise do undesirable things when they receive special characters such as RETURN, control-C, or ESC. This should make it possible for a version of the PCNET protocol to run on almost all large computers without requiring special system modifications. Later versions of the PCNET protocol will send and receive all 256 possible bytes, which will improve the communication speed, but may not be suitable for older operating systems on large computers.

The PCNET protocol is designed to use modems that can dial the telephone (electronically) and answer incoming calls. This will make possible fully automatic operation, so the computer can make calls when the phone rates are lowest. In the most advanced versions of the PCNET protocol, the computer will be able to automatically receive an incoming call, store a file or message, and call another computer to forward the material all without requiring that the user be present. The protocol will include the necessary estimation of time ("timeouts") so that expensive phone calls are not continued indefinitely when something goes wrong.

We will propose an addressing method that uses latitude and longitude to help the computers figure out which direction to forward messages, and the phone number to dial once the message reaches a computer that knows how to call the desired destination computer. Sometimes, calling the destination computer will require stored knowledge of a login procedure.

Although designed for automatic dial and answer modems, the PCNET protocol may be modified so that less versatile modems may be assisted by a user who is willing to do the dialing and answering when needed.

The simple version of PCNET will only communicate in half-duplex mode, sending information in only one direction at a time. While this is necessary for some kinds of hardware and our goal of universality, later versions will add the ability to communicate in full duplex, sending information in both directions at the same time (on different pairs of tones).

An advanced feature that will only be available later is the ability to have multiple logical "streams" of data multiplexed together. This could become useful as personal computer operating systems become more powerful, so that several programs on one machine could be communicating with several on another machine at the same time, with only one phone line. The PCNET protocol would get data from the correct programs and deliver it to the correct destination programs, automatically.

Perhaps more important than the technical aspects of the protocol is the process by which we intend to make the PCNET protocol available to everyone. This protocol is a creation of the personal computer community. Almost all the designers are owners of personal computers who recognize the immense value of a standard method for communicating among personal computers. We dream of grocery price clubs, of sending educational materials, of medical care advice, of software being shared between incompatible brands of computers, of electronic mail around the world, of robots communicating by radio links in noisy environments, of coded messages being delivered without the errors that would render the message unreadable. We work together, sharing programs, building upon each other's work, to make the dreams become reality.

When we have created and tested an implementaiton of the PCNET protocol

*(Continued on page 52)*

# MINCE: Not Just Another Editor

by Barry A. Dobyns

As computing power gets cheaper and cheaper, we approach the problem of getting useful data into computers in a fashion that is acceptable to the user of the computer. Programmers in particular are going to be under increasingly greater pressure to be more and more productive. Their ability to deal with the problem of getting the solution out of their heads and into the computer will be influenced in great measure by the ease with which they can physically enter data into the machine. Electronic communications, still a budding field in many ways, will have its success measured in part by the ease with which the user can use the system, primarily the ease with which messages can be composed and edited.

With few exceptions, all the programs and data in every computer originated with someone typing to (at) an editor or parser of some sort. This is, I realize, a gross over-simplification, and there are cases which may be considered to be not connected with any editors at all (a friendly IBM 029 card punch, for instance), but the point is well made: in most cases someone uses a text editor of some sort to generate a source program (even DDT can be considered a crude sort of editor for hex), or to enter data, or to generate a program which may, itself, be a restricted type of editor or parser for data entry. A lot of users never consider using any editor other than the one which came with their system for the production of text and programs. Some realize they can improve their efficiency by picking another editor that is better suited to their requirements. Most users have video terminals of one sort of another, yet don't have an editor that takes advantage of the power available to them. This article is about a class of editors whose popularity is gaining, screen editors, and one of these for CP/M in specific, MINCE ($125.00, Scribble $125.00, AMETHYST $350.00, available from Mark of the Unicorn, P. O. Box 423, Arlington, MA 02174).

## Screen Editors in General

Simply put, screen editors are a class of editors that are meant to take advantage of the fact that CRT's (Cathode Ray Tube terminals) don't use paper. The major advantage of this is that the image on the screen can be changed, modified, and updated quickly. This is in stark contrast to paper terminals, where any modifications usually require that the entire element to be modified be printed out again rather than just changing the part that was modified; or worse, the editor requires that one remember the changes made, until one specifically requires that the region which has been modified be printed out.

Many people are making the move from paper-terminal-oriented editors to screen-oriented editors which show the text being edited at all times. These editors show the changes in the text dynamically, so the user always knows what effect his changes have made. These usually allow the user to move about in the text with simple single character commands and allow modification of the text at the cursor. (Cursor: 1. the place on a terminal at which the next character will be inserted, usually a block or underscore, sometimes blinking. 2. the place in the text file where most commands will have effect next, sometimes called the Point usually the same as 1. in screen editors.) Many display editor users can dispense with numerous listings of their text, as it is often easier to enter the editor and use a search facility to find something than to peruse many pages of print. Other users may find that the act of composition itself is made much easier, rather than writing the text out on paper first, one can start by typing it into the editor, using the editor to update and revise it. In general, display editors are also easier to use, as the user can see the text he is editing and immediately see the effect of the commands he enters on it. On a printing terminal this usually requires a non-trivial amount of mental effort to remember what has happened, or results in a lot of wasted paper and time, especially if a lot of changes are to be made.

A simple type of screen editor could have merely five commands, and still be very powerful. These would be up, down, right, left, and delete (one character). The screen of the terminal would display the area of text around the cursor, called a "window" because it acts just as a window onto the text might, hopefully redisplaying it if the user moved off the portion of text currently shown on the screen. Usually the commands would be delegated to single characters easily typed from the keyboard (control characters are relatively easy to type on most terminals), and characters which are not commands should be self-inserting, that is they should insert themselves into the text at the place marked by the cursor. Most screen editors do that and a lot more, offering most of the usual search and replace facilities that other editors offer, and often many special features of their own. Some editors also offer an embedded text formatter which can be used for the production of "pretty" printed copy.

## What Makes a Screen Editor Good

Unfortunately, even the minimum screen editor is not a simple program to write at all when one has to deal with more text than there is available memory. There is also the problem of getting already prepared text into the editor, and saving it when the user is done. How the editor deals with these problems and what other conveniences or inconveniences it offers to the user, determine in a large part the utility of that editor.

Herewith I examine the features that I believe a good editor should be endowed with (this is by no means the final word on this subject, you can still love ED all you like, I just don't use it myself anymore).

Firstly, it should be at least as fast as I am. I am by no means a speed typist. I flunked every typing course I took (three), and my technique at a keyboard has been described in numerous degrading ways that I will decline to mention. Still, if I am provided with a delete key (the correction key on those very expensive Selectrics does fine), I can make any teletype in the land tremble with fear and awe (not recommended, they tremble enough as it is). Insertion from the keyboard should never be interrupted by disk I/O, calculation, redisplay or Acts of God. I refuse to use any editor that loses characters that I type. However, I may be convinced to use it if it is merely slow in showing me what it is up to, but doesn't lose anything. On the other hand, my spooler causes my system to lose characters all the time. I do not like to see any piece of software choke to death. In particular, the editor should not choke on files over 200K long. Maybe you can always get along with very short files (much less than 64K), but I always seem to have all these files around that exceed 100K (source code for interpreters and

compilers can often get big). Indeed, there should be no limit to the length of the file you can edit other than the amount of storage you have available.

I want to be able to move about faster than one character at a time. Character movement commands in line, screenful, word, sentence, paragraph (or atom, statement, and procedure) increments are a must. There should be modes for different types of text. If I am editing Pascal instead of English I should be able to tell the editor (or insert a keyword at the beginning of the file as a comment that the editor will pick up on) this fact so that my paragraph (English) commands now work on procedures (Pascal) or defuns (LISP) or whatever. There should be commands for deletion of the same units as movement. I should also be able to specify a "region" not normally delimited by the regular units for destruction and other purposes like saving, copying and moving. Any command which kills more than one character at a time should be reversible, should I turn fickle. Having to return to the original file in the event of massive failure is only marginally acceptable (it is expected after catastrophic failure). I should not be forced to save the new text out to the same filename, (nor should backups always have the extension .BAK). I should be able to create more than one output file from a single editing session, without exiting and reentering the editor. I may want to create a number of slightly different files, all with different names from the same initial file. (Form letters the hard way, or ten programs which only differ by one line or so.)

Searching and replacement should be straightforward and related. Insertion from other files should be easy. If it is doing redisplay, and I give it another command which requires redisplay also, it should execute the second command before finishing the redisplay for the first (if I give it sixteen consecutive "next screenful" commands, I want the sixteenth screen, not all fifteen in between). All commands should take numeric arguments where appropriate (one should type a command for "5 line deletes" rather than typing "delete a line" five times). In fact, the single action version of each command may merely be the special case of the command without arguments. In addition, whole command sequences should be delimitable in order to accept repetition counts. No command primitive should require more than two characters (keystrokes). All potentially catastrophic commands (like deleting whole paragraphs, and saving files) should be several keystrokes long.

Command-naming strategies should be as internally consistent as possible. If they are to be mnemonically related, make them all that way. If they are to be positionally related, pick a home key that gives you enough space around it to do the job properly. My personal preference is for mnemonically related command structures, as they tend to be distributed all over the keyboard, and can actually be touch-typed on many terminals, whereas position-oriented schemes usually are impossible to touch-type, and cramp my hand. There is much to be said for an editor, all of whose commands can be touch-typed (that is, typed with the hands remaining in the "home" position on the keyboard). This is often a function of both the editor and the terminal/keyboard design (many keyboards have the "Control" key in an awkward place, and have repositioned many of the "normal" keys, like "DEL" and "BACKSPACE" so that they are hard to get to). In any event I am vehemently opposed to the use of "special function" keys that require the operator to move his hands away from the keyboard. The time lost by locating the function key and relocating the home key is much greater than the time saved by typing one less keystroke. Sufficient documentation should be built in so as to enable a new user who is familiar with editors of that type to be able to learn it quickly.

Special cases should not break the editor (a file made up of only one very long line, for instance, will break many editors; trying to edit object code usually will break an editor, too). Broken programs are not to be tolerated, unless I have authored the program.

The file produced by the editor should be compatible with other editors for the same operating system, and should not be filled with "silly" things. I do not like the idea of an editor doing its bookkeeping in the file being edited. "Soft" characters is one such concept that still gives me the creeps. The problem here seems to be that the editor/formatter can't tell where a paragraph begins or ends, so the user must put in a carraige return to insure that paragraph breaks are not "filled" over. Maybe I'm harsh, but "soft" characters seem to be an attempt to overcome a design oversight, rather than a feature. Why not give the user of the editor a command that "fills" the text out to a specified column, so the user can get the column he wants it filled to and "grind" his text all he wants? Maybe give the user a mode for input that automatically breaks lines off as they are input too, but keep non-ASCII garbage out. I have a need for a utility that does nothing but take "soft characters" and replace them with "hard" ones if PIP didn't do it already.

My, he doesn't want very much, you might say. Indeed, as long as I can pick and choose about what I want, I will pick lots of features designed to make life easier for me. I compose at the keyboard of my terminal, and I like having only one editor for everything (text, papers, documentation, programs, data). I want my editor and my formatter separate though, thank you, as (I can hope) the creation of two programs (an editor and a formatter) will result in a more powerful one of each, as there is more memory available if they don't have to be co-resident. Besides, I may have a number of formatters, one that is fast and clumsy, and one that is slow but fancy, and one that is worthless but I wrote it myself, so I use it anyway. It is difficult to separate the editor and the formatter in many of the do-it-all programs, especially if they do "bookkeeping" in your files. On the other hand, there is the case for several editors and only one formatter, but the need for separate formatters and editors is still valid.

## Reality vs. Fantasy

Unfortunately, as you may have suspected, gentle reader, the perfect text editor will take infinite memory, an infinitely fast computer, and will be perfect to only one user. (No two users have ever agreed completely on what constitutes a perfect editor.) I often feel that there is no middle ground in computers at all (be it languages, editors, or hardware), which is fine since many of us depend on just that fact for our livelihoods. As it is, many of the editors now available for CP/M-based 8080/Z80 computers are very good, even by the standards of what is available for many "big mainframes." (Anyone who has ever used the CMS editor on a video terminal and agonized about the fact that there wasn't anything any better understands.)

At some point (usually very early,

before any code ever gets written) the designer of an editor is going to make some very important trade-offs and some decisions that will affect the nature of his editor. One way to protect against making the wrong choice is to put these decisions off as long as possible, i.e., don't start with a command set in mind that you may not want to use later, but build a command dispatch table so that you can link any of the commands (like Control-P) with any of the procedures that actually accomplish the work you want done (like Previous-Line). Many of these sort of choices, if deferred long enough, can be incorporated into a set of configurable options that the user can use to make the designer's version of a "perfect" editor into his own version of perfection.

Most of us can't afford infinite memory, but our editor and operating system can often be expected to gobble up as much as 32K, and that's not very much space left to edit in. The editor needs to make some very smart decisions about what part of the file to read in, and when, if it is to give the user the feeling of having the entire file in memory at once; or it will have to do something like ED does, and make one pass at the file, with no going back to the beginning unless it's still in memory. The editor can swap sections of the file back out to disk on a least-recently-used (LRU) basis, but one can get into trouble if one spends too much time doing disk I/O (one can lose characters from the keyboard), and it usually requires a special file on disk to swap to, which must be as big as any file that is to be edited. Smart swapping on an LRU basis is what many operating systems (and cache buffers) do to act as if available memory (or available fast memory) is much larger than it really is.

## MINCE, AMETHYST & EMACS

MINCE (Mince Is Not Complete Emacs) is a recent addition to the proliferation of Screen Editors. MINCE is modeled after EMACS, an editor which grew out of many years of experience with TECO at MIT. EMACS is interactively extensible, a nice feature indeed. MINCE is merely extensible. (The difference being that the High Level Code underlying EMACS (TECO) is interpreted, so that one can edit up a routine, and then execute it immediately, whereas the High Level Code underlying MINCE (C) is compiled.)

The advantages of extensibility are numerous. Short of actual extension, one can customize the editor to meet one's tastes. The simplest sort of customization, the rearrangement of commands, is by itself of inestimable value to those who do not share my penchant for mnemonically related commands. (You can use the "D" key for your home key if you like, and run out of room if you like too, while trying to design a position-oriented command set). Another sort of customization is making supplied commands behave the way you think they should behave, not the way the manual says they do. Movement commands often behave as intended, but rarely as expected. It is a simple matter to change the behavior of existing commands. In fact, the dividing line between extension and complex customization is not a clear one. MINCE can be bought in a standard configuration, or can be bought with the source code for the command set, so that extensions can be made. Users can write commands of their own to perform often-needed functions, or write "modes" in which some command definitions are slightly altered to facilitate easier editing of different text types. For example, one might have an ALGOL mode where the function that looked for paragraphs (an internal one) looked for "BEGIN" and "END" pairs starting in the first column while in English Mode it might merely look for a lone tab in the first column, so that the existing paragraph movement and modification commands will work in a useful manner. This extensible MINCE (which comes with BDS C and Scribble, a text formatter) is called AMETHYST.

MINCE is completely portable. Written in the language C, it is conceivably portable to any machine that will support C. Although developed on a CP/M system, Mark of the Unicorn has brought MINCE up on a DEC 11/70 already and the conversion and installation took a mere two days to perform. This sort of portability means that you can continue to use MINCE on your 16-bit or 32-bit computer which supports **NIX or C. Site license agreements (some clubs can qualify as sites) are also available. It is conceivable that MINCE could be the last editor you will ever need to get, no matter what you upgrade to or grow into.

MINCE tackles the problem of file management with an LRU scheme that works very well, even for files of appreciable size. MINCE does all if its work in in a MINCE.SWP file either on the logged disk, or on the A: drive. MINCE also allows the user to edit up to seven files concurrently. This may seem like a good way to get oneself into trouble at first, but it's an excellent way to compose a paper or program from several other papers or programs. MINCE also will display one or two windows at once. These windows can be into the same file or into different files.

MINCE's ability to display multiple windows and to edit multiple files is an invaluable aid in composing text or programs from many small subunits in different files. One might use it to create a new program which contains routines or functions swiped from a number of others, or to include part of a previous letter in a follow-up letter.

MINCE allows the user to save files to whatever filename (d:filename.ext) he wants, at any time in the edit session. File saving is only done when an explicit command to do so is issued. No .BAK files are created. Furthermore, no files are created automatically, even upon exit from the editor, although an attempt to exit when modifications have been made since the last save will result in a prompt to allow you to reconsider abandoning your work. A disadvantage of this scheme is that the default filename for a C-X C-S (Save File) is the same filename that the file originally had, which could potentially obliterate the original. There is also a C-X C-W (Write File) which prompts for a filename. Users should probably get accustomed to using Write File instead of Save File unless they are certain there is no need for backup. Using Write File with a generation number for the extension (extensions of the form .001, .002 and so on) would probably be the best bet for text creation. An operating system that provided for generation counts as a third extension would be the best.

MINCE produces files which are compatible with ED, and every interpreter and compiler I have tried. MINCE-produced files also can be fed into most of the text formatters I have attempted to use. The only problem will be with "automatic justification" with those formatters that will not take out "hard" carriage returns. Justify your text in MINCE first. MINCE leaves no bookkeeping at all in the files it creates, either during the session or after it is written. (You can reset out of MINCE with impunity, and be assured that nothing is damaged,

except your ego.)

For all my ravings about losing characters, MINCE occasionally loses characters on my system, but that is because my DMA disk controller can't possibly poll my console. MINCE would probably lose even less on a system that had interrupt-driven console input. MINCE has a type-ahead buffer of 80 characters, and does a lot of console input checking. It will only lose characters when a disk operation is in progress, and prints a message on the screen to this effect during every disk operation. Still, MINCE waits until the console has been idle for a while (a time unit that the user sets for himself when he first configures the program for his system) before doing any disk operations. MINCE should never have to go to disk while the user is typing (it really hasn't on me, either), unless you can type continuously for 20K or more without pausing. (If you can, I'm very impressed.)

The largest file size which can be edited is determined by the size of the MINCE.SWP file, another parameter you set when configuring. The authors recommend a 64K .SWP file, which I find quite reasonable for everything but those few monster files I mentioned earlier. Currently, the maximum allowable size for a .SWP file is 248K, and the largest file I have is 208K (but I never edit it anyway, I just print it out), so I'm not in trouble yet. I presume the maximum .SWP file size will be upped in future versions of MINCE (or when the authors get a hard disk).

The command set, as supplied, is a delight. There are several types of commands, determined by command prefix (or lack of prefix). Control-F (Control characters will be represented by the prefix "C-") moves forward one character (F for forward, get it?). C-B moves back one character (guess what the B is for). Meta-F forward one word (Meta characters are actually characters preceded by the ASCII "ESC" character, but for reasons of MINCE's heredity, they are called Meta- characters. (It is possible to generate Meta- characters in hardware, the authors' keyboard does this). If the term Meta- is taken to mean "greater than," then many of the command definitions make lots of sense), and M-B goes back one word. C-A and C-E go to the beginning and end of a line, respectively, while M-A and M-E do the same for sentences. "DEL" and C-D delete forward and back words. C-K deletes lines, and M-K deletes sentences. M-[ (square bracket) moves to the beginning of a paragraph, and M-] moves to the end.

File commands are all preceded by C-X (for eXtended commands). C-X C-F finds a file and reads it into a buffer. C-X C-R reads a file into an already existing buffer. C-X C-S saves a file to the last specified filename, destroying anything with that filename. C-X C-W writes to a different filename (prompting you for it). Other C-X commands select Modes (C-X M prompts for and selects a mode, like Page Mode (overwrite)) or tell you just where you happen to be in the text (C-X =). C-X K kills the current buffer (if you are using more than one, and you need to free up space). Termination is accomplished by C-X C-C, which prompts if you have modified anything since the last save, asking if you really want to throw it away.

MINCE uses two lines at the bottom of the screen to give useful information. On the "Model Line" MINCE shows its version number, the currently selected mode (if any, otherwise it shows "normal") the buffer name, the current filename, the distance the point is from the beginning of the buffer expressed as a percentage, whether the buffer has been modified or not and whether there is anything in a special buffer called the "kill buffer." On the last line of the screen, the "Echo Line," MINCE prompts for input, echoes command prefixes if you type too slow, tells you when it is writing to disk, or swapping, and signals completion of certain commands.

There is no help available within MINCE, but I suppose one could make a second window (C-X 2), switch to it (C-X 0) and read the MINCE short command list into it (C-X C-F SCOMM.DOC'CR') and scroll through it C-V forward, and M-V backwards) until one finds what one wants. On the other hand, the tutorial that comes in the manual is as good as any tutorial I have seen in a long time, and it comes in two versions, one for novices, and one for folks who think they have experience. If one has had experience with EMACS one may never need to consult the manual at all (except to confirm that you can't really have minibuffers full of TECO code, and to get the wall chart out of the manual and onto your wall). The wall chart can be cut up and pasted to your terminal (it fits on mine) or posted on the wall by your terminal (or bathroom mirror). Once you successfully make it through the tutorial you can do most of the terribly difficult things, and understand enough that you needn't always look in the manual for assistance. The command set is easy enough to remember so that one doesn't really miss internal documentation. (I learned how to use EMACS without any sort of manual at all, but there really is no excuse for stupidity.)

## Extending MINCE with AMETHYST

Purchasers of the AMETHYST package will receive a MINCE, a Scribble (a text formatter to be the subject of another article), a BDS C compiler and the source code in C for the MINCE command set, and support routines for the commands and the terminal in C. It does not include the source for the multiple-window redisplay, or the buffered file I/O, which is most of the body of MINCE. It is the intent of Mark of the Unicorn that you can rewrite the command set, and your own commands all you want, but that you cannot rewrite the editor itself (or gronk it for all eternity) and resell it. For **NIX users, who are usually provided with source code for everything, the source for MINCE is negotiable on a per site basis.

This may sound disheartening at first to all the hard core C hackers out there who are gearing up to perform wizardry on their version of MINCE. However, Mark of the Unicorn has properly abstracted most every conceivable function so that linking to internal functions is a simple task. AMETHYST is provided with an exhaustive program logic manual which details the internal organization and function of all the routines accessible to the programmer, entry point documentation is provided for every primitive available within MINCE, and an excellent discussion of state of the art theory of text editors is included.

MINCE is implemented as two editors, one inside the other. The inner editor consists of the redisplay and file handling functions, and editing primitives. The outer editor parses the terminal input, provides command dispatch, and interprets many of the more complex commands. It is the outer editor for which the source code is provided. In the outer editor, command names (characters typed at the keyboard) are bound to the commands (functions in C which in turn call the primitives contained within

MINCE). Such is the flexibility of this scheme that it is plausible that a programmer could create a MINCE mode to emulate nearly any video editor (although the motivation for such an act is beyond me).

Also provided is a terminal abstraction which could be modified to allow MINCE to support any sort of video device whatsoever. The terminal abstraction could be written to support memory-mapped video boards, terminals which do not support cursor addressing, or terminals which cannot be supported through CONFIG.COM (the MINCE configuration program). The terminal abstraction could also be used to make an inconvenient terminal livable. My terminal has the "DEL" and "←" (underbar) on the same key, but the "DEL" is a shifted character. I could alter the Terminal abstraction to swap the two on input, so I don't have to shift to type "DEL."

## Conclusions

MINCE is one of the finest editors I have ever used, on any machine. It is without question, in my opinion, a absolute steal at the base price, and a fantastic bargain as AMETHYST. MINCE is not perfect, and I am ready to admit that. MINCE lacks some features like provision for iteration of more than one command as a group, and a "grind region" in addition to the "grind paragraph" it now has. The exquisite beauty of AMETHYST is that the features I think it lacks, I can write in. And I will. So can you.

I hope there will be some sort of exchange program for AMETHYST functions and modes that users write. Judging from the EMACS history this will be the major way in which the program will grow, through input from its users. Most of the current library functions in EMACS were submitted by users in the community. This sort of exchange and support could ensure that MINCE will stay as far ahead of everyone else as it is now.

Mark of the Unicorn is ambitious about MINCE, AMETHYST, and Scribble. I feel they have every reason to be. The complete system is supportable on nearly every machine available, and is patterned after systems (EMACS and SCRIBE) which have fared very well in demanding user environments. There is no reason why MINCE and Scribble could not make the basis of an office-type word processing system that could rival the best of the "big name" systems. There is no reason that MINCE could not be used as a program development tool of the highest caliber. If you feel like you got locked into an editor that is merely mundane, then maybe you need MINCE.

## Bibliography

C. A. Finseth, B. N. Hess, S. Layson, J. T. Linhart, MINCE User Documentation (Mark of the Unicorn, 1980).

C. A. Finseth, B. N. Hess, S. Layson, J. T. Linhart, MINCE Internal Documentaiton (Mark of the Unicorn, 1980).

B. W. Kernighan and D. Ritchie, *The C Programming Language* (Prentice Hall 1978).

B. K. Reid and J. H. Walker, *SCRIBE Introductory Users Manual*, Second Edition (Carnegie-Mellon University, 1979).

R. M. Stallman, *EMACS The Extensible, Customizable, Self-Documenting Display Editor* (Massachusetts Institute of Technology, AI Memo No. 519, 1979).

R. M. Stallman, *EMACS Manual for TWENEX Users* (Massachusetts Institute of Technology, AI Memo No. 555, 1980).

L. Zolman, *BD Software C Compiler V1.4 User's Guide* (BD Software, 1980).

■■

## PCNET Protocol

*(Continued from page 47)*

that is understandable enough, we will follow the example of the FORTH Interest Group, conducting a large workshop in which the owners of many different kinds of computers recode the "model" implementation into their favorite language, for their own machine – but also for all other users of their kind of computer. We will distribute the basic versions of the PCNET protocol without copyright. We will then encourage people to develop other services and products that use the PCNET protocol as the basic means of communication. We have the support of People's Computer Company, which is noted for encouraging the beneficial application of personal computers.

We are grateful for the talented contributions of many individuals who have recognized this effort's importance. We would welcome further contributions in money, materials, or participation. We will soon be ready for the large workshop to recode the model implementation for many kinds of computers, and enthusiastic volunteers from the San Francisco Bay area are particularly welcome. ■■

## San Diego Computer Society

The SDCS Board of Directors has authorized the formation of an Educator's Interest Group. In order to better support the use of computers in schools, EDIG will provide teachers, students and administrators (presumably) with information, techniques and other forms of assistance in areas of computers and computer science. They have scheduled monthly meetings for different parts of the San Diego metro area. Questions or comments may be directed to Mel Zeddies, c/o SDCS, P. O. Box 81537, San Diego, CA 92138.

## British Computer Club

Hobby computer club members at the Polytechnic of North London will run an inner city computer center in conjunction with staff members and senior students. They will be available to give demonstrations, familiarization courses and advice to visitors.

The computer center aims to help small businesses in the urban area. A £20,000 grant from the Hackney/Islington Inner City Partnership, in association with the Inner London Education Authority, will be used to acquire several computer systems in a broad range of prices. Small businesses will be able to use these for a computer-based business control system, integrated with the rest of their business and with the owner educated in how to exploit the full potential of the computer operations.

The project is hoped to alleviate a major cause of job loss – the high mortality rate of small business in the area. A principal cause of the failure appears to be lack of financial control, poor purchasing and marketing systems and need for better management skills. The Polytechnic has a thriving computer club with an active business computing section. They plan to open the center each evening and all day on Thursdays and Fridays.

## FORTH Interest Group

In the San Francisco Bay area, FIG members and interested parties meet the fourth (FOURTH?) Saturday of each month from 10 a.m. to 4 p.m. This longish meeting is in two parts. The mornings are for demonstrations and various presentations, while the afternoon schedule consists of more formal proceedings. You can find it in the Community Hall of the Southland Mall in Hayward.

■■

$2.50 ($3.00 In Canada)

# DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

## FOR USERS OF SMALL COMPUTER SYSTEMS

Number 55 May 1981 Volume 6, Issue 5

**Santa Barbara Tiny BASIC for 6809's**
*by James A. Hinds*

**A Smart 2716 EPROM Programmer**
*by Eric M. Miller*

**CAD for Time Delay Circuits**
*by Hal Knippenberg*

**What to Do with Those Extra Function Keys**
*by John B. Palmer*

**and more!**

P.O. Box E
Menlo Park, CA
94025

# DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Volume 6, Issue 5
Number 55
May 1981

# Contents

## Articles

**Using (and Misusing) the Z-80 Microprocessor** . . 7
*by Ray Duncan*

**A Smart 2716 EPROM Programmer** . . . . . 10
*by Eric M. Miller*

**What to Do With Those Extra Function Keys** . . . 14
*by John B. Palmer*

**CAD for Time Delay Circuits** . . . . . 17
*by Hal Knippenberg*

**Santa Barbara Tiny BASIC for 6809's**. . . . . 22
*by James A. Hinds*

*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025.

© 1981 by People's Computer Company unless otherwise noted on specific articles. All rights reserved.

## Departments

Letters. . . . . 4
Of Interest. . . . . 12
Book Reviews . . . . . 18
Dr. Dobb's Clinic. . . . . 20
Software Reviews . . . . . 40

**Publisher**
Michael Madaj
**Editor**
Marlin Ouverson
**Contributing Editors**
Dave Caulkins
Dave Cortesi
**Business Manager**
Maureen Christine
**Art Director**
Clifford West
**Artist**
Barbara Ruzgerian
**Typesetter**
Renny Wiggins
**Proofreader**
Nancy Heubach
**Marketing Director**
Craig S. Harper
**Advertising Sales**
Janice Powell
**Circulation Manager**
Peter Clark
**Circulation Assistant**
Leah Dansby

# Editorial

Computers and their owners are at the leading edge of this information era. Whether they will remain there long enough for us humans to more fully realize our potential for self-determination depends upon several things.

First, the computer user must know what information he requires. Next he needs to learn who has it, how to access it and what to do with it. Networking in general, and computer bulletin boards specifically, are moving toward the fulfillment of personal computing power. This is done by making many kinds of information available to many people who would otherwise not have access to it (or else it would take much longer periods of time without computers).

*Dr. Dobb's Journal* has provided a simple list of CBBS's for some time. Too simple a list, actually, to be used in the efficient process outlined above. Instead, the user is forced to browse through the telephone numbers as he would through books on a bargain table, sifting through thousands of items before finding one or two that only coincidentally meet his needs.

Reader response has been a great help to us, with updates, deletions, changed numbers and additions coming in daily. But what is the best way to present all these valuable numbers? One reader sent a list of *six hundred* CBBS's, stating "... I can't always keep every number current, but at least 90% of the numbers should work."

*DDJ* needs to know what you need to know about these bulletin boards so we can be of greater service to you. Some points might be: the necessity to have a particular piece of hardware or software to use it; hours of the day and night when it is available; if it is mainly for a special interest group; the type(s) of information provided.

Additionally, bulletin board pros (and those just testing the water) can provide another kind of assistance. Because much CBBS information changes rather often, it will take a lot of manpower to keep track of it. If you can monitor the numbers within a given area code on a periodic basis (say, once a month), send your name and other pertinent information to my attention with a "CBBS" on the envelope's lower left-hand corner.

Let us know what you need. We'll incorporate that into our soon-to-be-redesigned Electric Phone Book.

Marlin Ouverson

Marlin Ouverson
Editor

# Letters

## Analysis of an Analysis

Sirs:

I read Howard Whitlock's "Analysis of the Use of the 6502's Opcodes" (*DDJ* #53) with considerable interest, partly because of my own researches into an improved instruction set, and partly because of my experience in comparing the 6800 with the 6502. I think his conclusions are not adequately supported.

The power of the 6502, compared to the 6800, is indeed based on its variety of addressing modes, including indirect post-indexed. However, this particular addressing mode on the 6502 is only powerful by comparison when applied to the small tables and data structures typical of small control applications, for which the 6502 was intended. The index register is too small to be useful in large applications like a general purpose interpreter such as Pet BASIC might be, except and unless arbitrary restrictions are placed on such structures as string and line length (though this is common in BASIC).

My 6502 Tiny BASIC shows exactly the same high incidence of this one address mode for exactly the same two instructions, namely LDA and STA. I have not examined the code in Pet BASIC, but I would guess that these two instructions are used the same way there as in my interpreter, namely with a constant zero in Y (notice the very high incidence of LDY – Howard does not mention the value of the constant so loaded: is it usually zero?). The point is that the supposed power of the post-indexed indirect addressing mode is not used at all, but this happens to be the only way to simulate in the meager 6502 instruction set the power of a 16-bit index register.

This is not to say that the 6502 is not "more powerful than the 6800" or that post indirect indexing is useless. The intelligent programmer will use all of the tools (opcodes and addressing modes) at his disposal to get the job done, and the algorithms to do the same job in the 6502 are necessarily different than in the 6800. For a long time the Microsoft 6502 BASIC was the "fastest" BASIC around, even after normalizing for memory cycle time. I do not know whether this was because the 6502 was indeed more powerful than the others, or if the programmer was more able to adapt his style to its instruction set, or if even it was simply the most recent try so it benefitted from the mistakes of all previous versions. I hope that the new IEEE standards work on benchmarks will enable a reasoned and balanced analysis of the various instruction sets.

It is interesting that the most frequent instruction in Pet BASIC is the JSR subroutine call. This tends to imply that the programmer has redefined the 6502 into a virtual machine of his own liking, whose opcodes are the JSRs. I wonder how many of the subroutines so called could be replaced by one or two in-line instructions from the Z-80 or 6809 instruction set?

In any case, a general comment on the utility of a class of instructions is not justified on the basis of the analysis of a single program, especially one with so limited a scope as a BASIC interpreter. I suspect that a chess program will show a radically different distribution, because the data structures are more compatible with the limitations of the 6502 register sizes. A video game might be still different than either.

I recognize that Howard's analysis is based on hard facts, and that my criticism is largely based on speculation and supposition. It may be that his conclusions are true and correct. I only claim that not enough data has been examined to establish that they are valid.

Tom Pittman
Itty Bitty Computers
P. O. Box 23189
San Jose, CA 95153

## FORTH Notes

Dear Marlin:

H. T. Gordon's recent article about FORTH (*DDJ* #51 and #52) have motivated me to write in with these comments. I received the FIG installation manual and 8080 assembly language listing last September and have had a running version of FORTH since December. My experiences in implementing and using FORTH make me agree that FORTH ". . . is a high-level assembly language for a virtual machine. . . ." I do not agree, though, that his machine ". . . ought to have a much more powerful core structure than it has." The strength of FORTH comes from the dictionary definition that acts as a macro for the high-level assembler. These macros string (thread?) together the primitives of the processor to form what appear to the user as new primitives.

I feel that Dr. Gordon has not been careful to distinguish between "macros" and "primitives" in his articles about FORTH. For example, Dr. Gordon proposes extensions of the stack manipulation instructions based upon the actions of the FORTH primitive ROT. This primitive, however, can be defined in terms of other operations such as transfer between the two FORTH stacks [ >R R R> ] and manipulation of the top two items on the stack [ SWAP ]. Thus, ROT becomes a higher-level macro given by the colon definition:

```
: ROT  >R SWAP R> SWAP ;
```

Other FORTH words can be defined in a similar manner:

```
: DUP  >R R R> ;
: OVER  SWAP >R R SWAP R> ;
```

Of course, the above definitions may be implemented directly in the machine code of the target processor as in the FIG implementation for the 8080. The choice of a macro-like colon definition or machine code depends upon the target processor. Machine code will take advantage of particular features of the target processor and provide maximum speed. A colon definition macro, on the other hand, allows a very compact nucleus of primitive functions. The FIG implementation, for example, has only about 25% of over two hundred definitions in machine code for the 8080. FORTH Inc. uses only about 512 bytes of machine code primitives in their 4K polyFORTH package as described in *FORTH Dimensions* Vol. 1, No. 4.

The proliferation of primitives proposed to enhance the FORTH virtual machine runs counter to the simplicity inherent in a zero-address machine. Operating only on the values on the stack allows great simplification since the operands are not scattered around in memory. A multitude of addressing modes is therefore not necessary. In addition, a multitude of registers is not required since the stack serves as the general purpose accumulator. These considerations are reflected in the P-Codes used by Wirth to develop a PL/0 compiler (*Algorithms + Data Structures = Programs*, by Niklaus Wirth, Prentice-Hall, 1976). Chung and Yuen also give a thorough explanation of a Zero-address machine used to implement their "Tiny" Pascal compiler (*Byte*, September-November, 1978). Both of these examples define machines with simple memory reference instructions, a few special purpose registers, and a few conditional and unconditional branch instructions. The power inherent in a

simple zero-address machine is hinted at by R. L. Gulbranson in *Computer*, April 1979 (reprinted in *FORTH Dimensions*, Vol. 1, No. 5).

The extensions suggested by Dr. Gordon for PARFOR should in my opinion, be directed towards the higher level of FORTH and not at the virtual stack machine level. I am basing this on the macro-like character of the colon definitions. Unfortunately, the "macro" defined by a colon definition cannot have parameters that control the macro expansion. A new defining word that would allow compile-time parameters would thus be needed to achieve this end. An extension at this level would allow defining the operations of PARFOR in terms of other FORTH primitives. Such an extension would truly lead to a parameterized FORTH-like high-level language that retains all of the FORTH structure. The basic core of FORTH would remain intact while extending the language to a higher level. Of course, such a parameterized macro-type defining word needs to be carefully thought out in more detail than is presented here. In addition, the nucleus of primitives necessary for FORTH has not really been addressed here. The 512 bytes used in polyFORTH probably defines the minimum for the core; however, thought needs to be given here to the tradeoffs of having a larger core of primitives in the code of the target machine.

G. Wayne Smith
7413 S. Woodward
Woodridge, IL

## Bullet Proofing for Everyone

Dear Marlin,

I feel Gene Head missed the point of having an A (for Assemble) command in a debugger, in his review of ZDM in the February issue of *Dr. Dobb's*.

An in-line assembler is not only a "hot patcher" using mnemonics, it can be a useful development tool in its own right. Short subroutines can often be quickly keyed in using mnemonics and executed immediately in Trace mode to ensure correct operation. Also an in-line assembler can be very useful for small throwaway routines. For example, the Morrow Disk Jockey disk controller uses an inverted internal data bus – recently I needed to make a small modification to the contents of the firmware ROM. The data read in from the ROM (which by the way is different from the listing supplied in the Disk Jockey documentation) was complemented, so a short in-line assembler routine was located in a vacant area of memory to invert the data read from the ROM so that I could check it with the in-line disassembler. It took perhaps two minutes to key AND execute the routine, had I been forced to use a text editor to generate an assembler source, run the source through the assembler, load the hex output of the assembler and then execute the object program it would have taken at least 8 to 10 minutes; all for a program I'll probably only ever use once!

As a programmer who tries to take care over documenting programs I can only despair at Gene's "I didn't even bother to read the manual." No matter how friendly a program is made in terms of user prompting, there are ALWAYS some aspects that can only be made clear in external documentation. While I guess we should all try to write bullet proof programs, it's certainly discouraging to think that the bullet proofing isn't only there for the inexperienced users. Let's encourage end users to thoroughly read all documentation before attempting to execute any program!!!!

It was disappointing to see that a new product like ZDM is still using the TDL mnemonics for Z80 code. At present there are 4 sets of Z80 mnemonics in use: Zilog, TDL/XITAN/PSA, Digital Research (Z80Lib) and Intersystems. The dismaying part is that the last 3 are all supposedly extensions of the Intel 8080 mnemonics to "logically" cover the Z80, yet all are different. While the Zilog mnemonics aren't perfect, it IS possible to use them quite successfully and they are the prime manufacturer's recommendation. Microsoft's M80 will assemble Zilog mnemonics, their L80 will generate a symbol table that Digital Research's ZSID can use for symbolic debugging – all the basic elements for developing machine language programs using Zilog mnemonics are now READILY available in the marketplace. New Z80 software should be using Zilog mnemonics, not mnemonics sets from companies long out of business.

*Dr. Dobb's* keeps getting better all the time, keep up the good work!

Bill Bolton
Software Tools
P. O. Box 80
Newport Beach
NSW, 2106
Australia

## Of Words and Cardboard

Dear *DDJ*:

Readers interested in Caulkin's article in *DDJ* #53 ("Computer Based High Speed Reading") can use ED and PIP in the CP/M system to obtain essentially the same results without additional programming of any kind. Although the speed of single word presentation is not continuously variable, it can be readily changed. Very high speeds (depending on available terminal) may be achieved.

The CP/M user will require a video terminal which clears and homes the cursor when receiving a form-feed (such as Intertec and others). First, edit any existing text file:

| | |
|---|---|
| A > ED FILE.TXT | [↑Z is control-Z, etc.] |
| *#A | (for simplicity, assume it fits in memory) |
| *Ib↑Z | (insert initial blank) [note: b=blank] |
| *#S↑I↑Zb↑Z | (change all tabs to blanks) |
| *B#S↑L↑Zb↑Z | (change all line-feeds to blanks) |

*(Continued on page 45)*

Don't let your hot utilities
or computing ideas
sit on the shelf.
*DDJ* is a great place
for bright authors
to get a start.
Contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# Using (and Misusing) the Z-80 Microprocessor

Ever since its introduction, the Zilog Z-80 microprocessor has had an aura of glamor and power compared to the workhorse Intel 8080 or 8085. This reputation is justified to some extent by the additional instructions, registers, addressing modes, and vectored interrupt capabilities of the Z-80. However, unless the additional Z-80 instructions are employed very carefully, they may cause a definite decrease in performance compared to using the 8080 instruction set alone (assuming the same machine speed, of course). In particular, indiscriminate use of the index registers can lead to severe speed and memory requirement penalties. In this brief communication I will try to point out some "good" and "bad" examples of Z-80 coding. Zilog mnemonics will be used throughout for clarity. All execution times assume a 4 mHz system clock, and are obtained from the specifications published by Zilog, Inc.

Observe the following common example of an "indirect" 16-bit fetch from memory to register BC, using the contents of a named variable for the target address:

| **8080 version** | | **bytes** | **microseconds** |
|---|---|---|---|
| LD | HL,(label) | 3 | 4.00 |
| LD | C,(HL) | 1 | 1.75 |
| INC | HL | 1 | 1.50 |
| LD | B,(HL) | 1 | 1.75 |
| | | 6 | 9.00 |

| **Z-80 version** | | **bytes** | **microseconds** |
|---|---|---|---|
| LD | IX,(label) | 4 | 5.00 |
| LD | C,(IX) | 3 | 4.75 |
| LD | B,(IX+1) | 3 | 4.75 |
| | | 10 | 14.50 |

The Z-80 oriented code looks much neater, but causes an approximately 60% increase in both execution time and memory requirements.

How about an indirect increment of a counter in memory?

| **8080 version** | | **bytes** | **microseconds** |
|---|---|---|---|
| INC | (HL) | 1 | 2.75 |

| **Z-80 version** | | **bytes** | **microseconds** |
|---|---|---|---|
| INC | (IX) or (IY) | 3 | 5.75 |

Using the index in this case causes more than doubling of executing time and memory requirements!

Your use of the index registers also causes penalties when subroutines must save and restore them:

| **8080 version** | | **bytes** | **microseconds** |
|---|---|---|---|
| PUSH | BC or DE or HL | 1 | 2.75 |
| POP | BC or DE or HL | 1 | 2.50 |

| **Z-80 version** | | **bytes** | **microseconds** |
|---|---|---|---|
| PUSH | IX or IY | 2 | 3.75 |
| POP | IX or IY | 2 | 3.50 |

One would imagine that the index registers would be well suited to table translations by incrementing the index to compare bytes through one table until a match was found, then exploiting the indexed offset addressing mode to extract a corresponding byte from another table. Yet in the simpler cases at least, the overhead of the index registers again outweighs the cleanness of the assembler code. For example:

| **8080 version** | | **bytes** | **microseconds** |
|---|---|---|---|
| CP | A,(HL) | 1 | 1.75 |
| JP | NZ,nomatch | 3 | 2.50 |
| LD | DE,offset | 3 | 2.50 |
| ADD | HL,DE | 1 | 2.75 |
| LD | A,(HL) | 1 | 1.75 |
| | | 9 | 11.75 |

| **Z-80 version** | | **bytes** | **microseconds** |
|---|---|---|---|
| CP | A,(IX) | 3 | 4.75 |
| JP | NZ,nomatch | 3 | 2.50 |
| LD | A,(IX+offset) | 3 | 4.75 |
| | | 9 | 12.00 |

What about the relative jump (JR)? Many programmers new to the Z-80 observe that the unconditional JR instruction occupies only two bytes compared to the three required by the absolute jump (JP), and quickly set about editing all their tight little time-dependent loops to incorporate this neat looking mnemonic. Surprise – the JR instruction takes 3 microseconds to execute, compared to 2.50 microseconds for JP, an increase of about 20% in execution time. Now that we have excellent relocating assemblers available, use of unconditional JR should probably be avoided. On the other hand, the conditional relative jump gives a slight advantage in execution time (1.75 microseconds false case, 3.00 microseconds true case) over the conditional absolute jump (always 2.50 microseconds), as long as the branch condition is not met over 50% of the time.

Naturally, the Z-80 would never have become so popular unless there were significant hardware and software advantages to its use. Careful study of the code set reveals a number of instances where use of the Z-80 instructions and registers can yield striking improvements in execution time and memory requirements.

Consider an entry to a subroutine where all registers will be needed for working storage yet the main program requires that the contents of all registers be unchanged at the end of the call. The 8080 requires that everything be pushed onto the stack and later restored, but the Z-80 allows you to swap with the alternate register set for a dramatic increase in speed:

| **8080 version** | | **bytes** | **microseconds** |
|---|---|---|---|
| PUSH | BC | 1 | 2.75 |
| PUSH | DE | 1 | 2.75 |
| PUSH | HL | 1 | 2.75 |
| (subroutine body) | | | |
| POP | HL | 1 | 2.50 |
| POP | DE | 1 | 2.50 |
| POP | BC | 1 | 2.50 |
| | | 6 | 15.75 |


## by Ray Duncan

*Ray Duncan, Laboratory Microsystems, 4147 Beethoven Street, Los Angeles, CA 90066.*

| Z-80 version | bytes | microseconds |
|---|---|---|
| EXX | 1 | 1.00 |
| (subroutine body) | | |
| EXX | 1 | 1.00 |
| | 2 | 2.00 |

The augmented sixteen-bit arithmetic capabilities of the Z-80 are a godsend both for the sake of performance and clarity of program code. For example, subtracting DE from HL, leaving the result in HL:

| 8080 version | | bytes | microseconds |
|---|---|---|---|
| LD | A,L | 1 | 1.00 |
| SUB | A,E | 1 | 1.00 |
| LD | L,A | 1 | 1.00 |
| LD | A,H | 1 | 1.00 |
| SBC | A,D | 1 | 1.00 |
| LD | H,A | 1 | 1.00 |
| | | 6 | 6.00 |

| Z-80 version | | bytes | microseconds |
|---|---|---|---|
| SBC | HL,DE | 2 | 3.75 |

(This assumes that the status of the carry flag is known, otherwise it must be turned off with a preceding OR A,A instruction which wastes some time.)

The additional direct register load and store instructions of the Z-80 are both handy and speedy. Contrast the task of getting a 16-byte quantity from memory into the BC register:

| 8080 version | | bytes | microseconds |
|---|---|---|---|
| PUSH | HL | 1 | 2.75 |
| LD | HL,(label) | 3 | 4.00 |
| LD | B,H | 1 | 1.00 |
| LD | C,L | 1 | 1.00 |
| POP | HL | 1 | 2.50 |
| | | 7 | 11.25 |

| Z-80 version | | bytes | microseconds |
|---|---|---|---|
| LD | BC,(label) | 4 | 5.00 |

The Z-80 has a string move instruction, as everyone knows, which is practically worth the price of the chip itself. If one sets up HL with the source address, DE with the destination address, and BC with the number of bytes to transfer:

| 8080 version | | | bytes | microseconds |
|---|---|---|---|---|
| LOOP: | LD | A,(HL) | 1 | 1.75 |
| | LD | (DE),A | 1 | 1.75 |
| | INC | HL | 1 | 1.50 |
| | INC | DE | 1 | 1.50 |
| | DEC | BC | 1 | 1.50 |
| | LD | A,B | 1 | 1.00 |
| | OR | A,C | 1 | 1.00 |
| | JP | NZ,LOOP | 3 | 2.50 |
| | | | 10 | 12.50 per loop |

| Z-80 version | bytes | microseconds |
|---|---|---|
| LDIR | 2 | 5.25 per byte transferred |

The corresponding compare with auto-increment instructions is more difficult to use effectively, but can be exploited with good results in table lookup routines.

The point of this little demonstration was that the Z-80 is indeed a powerful processor, but the proper use of its many added capabilities in a time-dependent application requires an intimate knowledge of the instruction set at the machine level. The very elegant Zilog mnemonics, which tremendously improve the understandability of an assembly listing compared to the Intel mnemonics, blur the machine-level code distinctions for a new programmer, since they make all register-memory and register-register operations appear very similar. I hope that other readers will be provoked into contributing other examples of "bad" and "good" uses of the Z-80. ■■


**You Read Dr. Dobb's Journal And You Don't Subscribe?!**

**Save $13 off newsstand prices for 2 yrs. Save $5 for 1 yr.**

Can you afford to miss an issue with information vital to your interests? As a subscriber you can look forward to articles on Small-C, FORTH, CP/M, S-100, Compiler optimization, Concurrent Programming and more, delivered right to your door. And you'll never miss the issue that covers your project.

**DR. DOBB'S JOURNAL**
**P.O. Box E**
**Menlo Park, CA 94025**

**Yes!** Sign me up for ___2 yrs. $47 ___1 yr. $25

___ I enclose a check/money order
___ Charge my Visa, MasterCard, American Express
___ Please bill me later

Name _____
Address _____
_____
Credit Card _____ Exp. date _____
Acct No. _____
Signature _____

# A Smart 2716 EPROM Programmer

by Eric M. Miller

Most of us reach a stage where it seems ludicrous to use our home computer system with 64K of RAM and dual floppy disk drives to control the lawn sprinklers. Many such applications can best be served by small dedicated microcomputer systems. The industrial and commercial electronics companies enjoy the luxury of purchasing single board microcomputers from Intel, National Semiconductor, Pro-log and others. Unfortunately, these boards are priced at $350 and up, a prohibitive cost for all except the most avid (read fanatic) microcomputer enthusiast.

Miller Technology has recently introduced the M-80 single board microcomputer, priced in kit form as low as $69 – see inset.

## A 2716 Programmer

Although the application of a computer like the M-80 is limited only by your imagination, I used it to build a 2716 EPROM programmer for less than $150.

EPROMs may be programmed using data that was copied from another ROM, by data obtained from downloading from a computer or development system, or from manual entry byte by byte. The EPROM can be checked for full erasure, and can be verified to ensure the success of the programming. Contents of the EPROM or on-board RAM may be dumped to the terminal, and checksums generated. Checksums of EPROM contents are handy for microcomputer systems performing self-testing functions. All communications to the programmer are via a serial RS-232 data link.

## Programmer Hardware

The 2716 is a single supply 2K by 8 EPROM. Erasure is accomplished by exposing the die to UV light. Programming is done by pulsing pin 18 at TTL levels for 50 milliseconds while +25 volts is applied at pin 21.

Figure 1 shows the 2716 pinout. Lines $O_0$ through $O_7$ are the data outputs, and $A_0$ through $A_{10}$ are for addressing the contents of the EPROM. $V_{cc}$ is 5 volts plus or minus 5%, and $V_{pp}$ must be brought to +25 volts, plus or minus 1 volt, for programming. Pins 18 and 20 select the chip and enable the data outputs respectively.

Three 2716 modes of operation are used by the programmer. The READ mode is used for copying the EPROM contents into RAM, generating checksums, verifying programming, checking erasure, and dumping contents of the EPROM to the terminal. The PROGRAM mode is used only for entering data into the EPROM. The PROGRAM VERIFY mode is used for verifying that the programming was successful immediately after programming. These three modes and the proper pin conditions are summarized in Figure 1.

## Generating $V_{pp}$

The only non-TTL signal level needed is +25 volts for programming. This voltage, which might otherwise need a special power supply, can be generated by a simple DC to DC converter, along with the remainder of the electronics for the programmer. U3 is the ubiquitous 555 timer. The 555 produces 17 volt pulses at pin 3 which are stepped up by voltage double D1, D2 and C1, C2. Regulation at the +25 volt level is maintained through feedback resistors R3 and R4 by varying both frequency and pulse. Load regulation is better than 0.16% from no load to 45 milliamperes.

Programming a location of the 2716 is done by pulsing the $\overline{CE}$ line for 50 milliseconds while $V_{pp}$ is held at +25 volts. There is a secondary requirement that the address and data lines be stable for 2 microseconds before the application of the programming pulse. Section one of dual one shot U2 gives a 5 microsecond delay for the address and data setup times. It also pulls the wait line low to preserve the status of the address and data buses. The second section of U2, triggered after the 5 microsecond timeout, produces the 50 millisecond programming pulse. For programming to occur, both a manual enable via the PROGRAM switch and a software enable by one of the I/O bits must be true to allow a programming pulse to appear. This makes it a little more difficult to accidentally program an EPROM. The EPROM is located B000 to B7FF, and from a software point of view, programming is as simple as writing to those locations. Program switch status is sensed by bit 1 of Port A, and software programming enable is by bits 0 and 2 of Port A. The INSERT/RUN switch removes program power from $V_{pp}$ of the 2716 and halts the Z-80 during the insertion or removal of a 2716 from the zero insertion force socket. 2716's may be damaged by applying +25 volts to $V_{pp}$ before $V_{cc}$ is present.

Communications with the outside world are via a serial RS-232 link. Bits 4 and 5 of Port B perform serial I/O by a software UART.

## Programmer Software

Once the hardware is in place, an EPROM can be loaded in its entirety by a sequence like this:

```
LXI  D,0B000H   ;DE → EPROM
LXI  H,data address
LXI  B,2048
```

followed by a single Z80 LDIR instruction! Of course, those instructions must be preceded by a routine that tests the program-enable switch, setting Port A bit 0 true when the switch is operated. I've expanded that simple idea with a number of commands to accept data from a terminal, to verify the EPROM after it's been programmed, and so forth. Any reader wanting to build this circuit may write to me, enclosing $1.50 for copying costs, and I'll send a hard copy of the complete program.

## Construction

Building the basic M-80 takes no longer than a half-hour. Sockets and the few discrete components are quickly soldered into the board. The point to point wiring of the additional circuitry for the programmer takes about two hours.

A Textool zero insertion force socket is used for the 2716 EPROMs. A mother-daughter socket approach is used, since ZIF sockets do eventually wear out. The mother socket is soldered directly to the M-80 board. The daughter socket consists of the ZIF socket soldered to two headers and plugs into the mother socket. The two headers provide the correct height for the ZIF socket when the M-80 board is attached behind the front surface of the enclosure.

A PS-80 power supply is mounted to the base of the enclosure and provides the regulated voltages for the programmer. 115 VAC line voltage is converted to +5 at 800 milliamperes, +12 at 150 milliamperes, -5 and -12 at 150 milliamperes. In this application, the -5 supply was not used. The PS-80 also has an

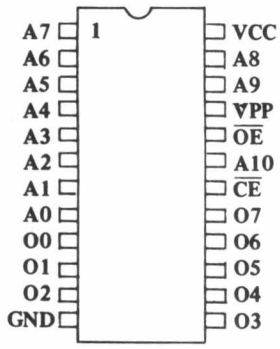
on-board fuse for the incoming 115 VAC, and a connector for easy wiring to a power switch and AC power.

## Modifications

Customizing the programmer for your particular needs can most likely be taken care of with software changes alone. Programming a selected byte or portion of an EPROM may be accomplished by enabling programming by the program switch, setting Port A bit 2 to a 0, Port A bit 0 to an input, and performing a memory write to the desired location with the data byte to be programmed.

Stand-alone operation for use of the programmer as a 2716 copier can be accommodated without the use of terminal or host computer. The necessary switches or pushbuttons may be added to the enclosure and sensed by a few of the input bits. LEDs could display status, such as "Programming Complete", and be driven by the output bits.

Multiple 2716s may be programmed simultaneously by simply paralleling them. Verifying each 2716 would, however, mean separately driving each $\overline{OE}$ line. Four output bits, along with a 74 LS32 quad or gate could control four 2716s by OR-ing the present $\overline{OE}$ line with an output bit. Setting an output to zero would enable that particular 2716. Complete software operation of the programmer would require a reed relay, driven by one of the output bits, to control the +25 volt supply to $V_{pp}$.




2716

| MODE | PINS $\overline{CE}$ / PGM | $\overline{OE}$ | $V_{pp}$ |
|---|---|---|---|
| READ | 0 | 0 | +5 |
| PROGRAM | Pulsed 0 to 1 | 1 | +25 |
| PROGRAM VERIFY | 0 | 0 | +25 |

**Figure 1. 2716 pinout and mode selection.**

## Conclusion

There are many other instances in which the application of inexpensive intelligence via single board microcomputers can be useful. Bus-oriented microcomputer systems with mass storage and a multiplicity of peripherals cannot address the convenience and economy necessary for dedicated control and processing situations. The M-80 single board microcomputer provides the needed computing power, versatility, and compact size at a low cost. ■■

*Miller Technology products are distributed by Quest Electronics, 2322 Walsh Ave., Santa Clara, CA 95051. The M-80 contains a Z80 CPU, sockets for 2K of 2708 EPROM and 2.1K of RAM, 16 I/O lines and a system clock. Bare board prices $28.50. Minimum kit (128 bytes RAM) $69. Full kit (2176 bytes RAM) $99. Fully assembled and tested $185. Monitor ROM $20. Tiny BASIC ROM $30. Power supply kit $35, assembled and tested $60.*


Even if you have never been published before,
*Dr. Dobb's* wants
to see your good computing ideas.
Contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# Of Interest

by Dave Cortesi

It must have been a slow month at the ad agencies; the press release heap is fairly small. Even so there are several items you might like to know about. Let's begin with two conferences, one popular and the other esoteric.

## Two Conferences

The **Personal Computer Arts Festival** is to be held in conjunction with Personal Computing 81, the big Philadelphia show scheduled for August. We haven't seen anything on the parent show as yet, but the Arts Festival Committee is on the ball with an early announcement. They want anyone working in digital art or music to participate with a talk, an exhibit, or a performance. If you want to take part you should send a short description of your presentation to them by July 1st. Include tapes or slides if possible; return is promised. If you want to be listed in the resource guide they're assembling, get your entry to them by June 1st.

Readers who are into robotics might want to investigate the Tenth Annual Symposium on Incremental Motion Control Systems **(IMCSS)** to be held in Rosemont IL June 1-4. Experts from many countries will talk about all sorts of stepping motors and their microprocessor controllers.

## The Contest Is Still Open

One or two readers may have been in a submarine (or much the same thing, a university computer center) for the past couple of months and so won't know about the national competition in personal computer aids for the handicapped. The competition is being run by John Hopkins University and is co-sponsored by Radio Shack. There's a $10,000 grand prize, a bunch of computers, and a lot of glory up for grabs. Information and entry forms should be available at any Radio Shack store, but hurry because the contest closes June 30th.

## Elicon Goes FORTH to an Oscar

The **Elicon** company sent us a brag on their motion-control system, which as part of their Camera Control System for movie special effects received an Academy Award nomination for technical achievement. What caught our interest was that the programming for the system was done entirely in FORTH. Elicon credits the language with easing design and lowering costs.

While we have the attention of FORTH fans, we'll pass on the **FORTH Interest Group**'s claim that they now have over 2500 members. That doesn't mean that they're satisfied, of course. They want even more (at $12 a year) and have lots of pubs and cheap code to attract them with.

## APLers Need Love, Too

Members of the class of applicative languages seem to engender strong loyalties in the breasts of their users. There are FORTH fans, of course and LISP lovers, and APL fanatics. The latter are deprived in that an efficient interpreter is too complex a beast for a really satisfactory implementation on present micros, although we have hopes for the new 16-bit machines. While they're waiting, frustrated APLers might pass the time with the **APL Market Newsletter**, a small quarterly which reports on new implementations and applications. It costs $8 a year. The latest issue predicts that Microsoft's version of Unix will have APL, and reports that **Softworks Limited** has an APL for Alpha Micro systems.

## The Software Market Served

The chaotic state of the software marketplace is drawing the attention of a number of publishers. **Software News**, a new monthly tabloid, will commence publication in May 1981 to "fill the industry's compelling need for a newspaper devoted exclusively to software." We wish them well in a challenging task.

**Micro-Serve** writes to announce the fourth edition of their *Software Vendor Directory*. It claims to list over 4000 products from over 1000 vendors, indexed by hardware and application type. They sell it two ways: as a book for $57.95, and as a CP/M-compatible diskette, with a search program, for $78. Semi-annual updates are $25.

## OK, Who *Is* Who?

An organization called **Alternet** produces a *Community Computerist's Directory*, which they rather grandiosely bill as "the Who's Who of people and computers." It appears to contain entries (written by the subjects) for just about any person or organization that relates to computers in any way. Buy the book for $4, or list yourself in it for $5.

## What's in a Name?

Carl Helmers, the well-known fellow who recently stepped down from the editorship of *Byte*, notifies us that he is commencing a personal newsletter, the *Carl Helmers Personal Computer Letter*. It will be published by **North American Technology**. Carl expects the newsletter to provide you with "a valuable and perceptive perspective on the field of personal computing." We can't believe an experienced editor wrote that sentence. Regardless, you can become a charter subscriber for just, um, 200 dollars. Now you know what's in a name.

## Pubs Potpourri

**Tab Books** have sent us a wad of book flyers, and several of the books look good. Here are the titles that turned us on, with the paperback price and order number in parentheses: *33 Challenging Computer Games for TRS-80, Apple, and PET*, D. Chance ($7.95, 1275); *Programmer's Guide to LISP*, K. Tracton ($6.95, 1045); *The MC6809 Cookbook*, C. D. Warren ($6.95, 1209); *Microcomputer Interfacing Handbook: A/D and D/A*, J. J. Carr ($8.95, 1271); *The PLL Synthesizer Cookbook*, H. Kinley ($7.95, 1243). We take pleasure in noting Tab's resistance to the current practice of charging 10 cents a page and up for technical paperbacks.

## Software For All

There weren't very many software announcements this month, but those that came were nicely spread over a number of systems. **Phase One Systems** wrote that they now have OASIS device drivers for the Corvus hard disks. Phase One also sent us a copy of their new OASIS reference card. It looks useful and costs a buck.

The lone CP/M entrant comes from **Softronics**. It tells of DATABS, a data abstraction language derived form the CLU language (CLU was one of the first structured systems programming languages; it made a few waves in computer science circles some years back). The release is not awfully specific about DATABS, so we can't be sure just what they mean by "data abstraction," but the usual idea is that you declare not only the arrangement of a data type as in Pascal, but the operators that will work upon that type as well. At any rate, the price of $49.95 for compiler and run-time library is low enough. Language freaks might want to

check it out.

Users of Pascal, whether it be Pascal/M or UCSD on an Apple, may like to know that **HDP** has three software tools for sale. The first is a Screen Handler ($75); you describe a screen layout in a file and the Handler takes care of formatting the screen. The Output Formatter ($37.50) does something similar for printed reports. The Forms Generator ($50) is a superset of the latter. With it, the programmer sets things up and thereafter the non-programming user can choose and run his own output format.

**TSC** tells us of their new sort-merge package for UniFLEX systems. The sort is written in 6809 machine language; it claims to handle any size and format of input file, sort on 20 keys of up to 250 bytes each, and allow for record selection or exclusion. The package costs $100.

The Electronic Typing package for **Zenith** and Heath Machines has been upgraded. Version 5.0 supersedes version 4.5, adding automatic page numbering (it didn't have it before?) and other features.

## Electronic Ordering

We noted above a software directory that is available in print or as a diskette. The new **Westico** software distribution company aims to be equally imaginative; they've set up a dial-up order line. Call (203) 853-0816 with your 300 baud modem and you can review their product list and place orders. Their release does not say so, but we can see the day approaching when one might report problems and check lists of known bugs the same way. It's good to see some of the endless chatter about the electronic mail being realized.

## Another Single-Board Computer

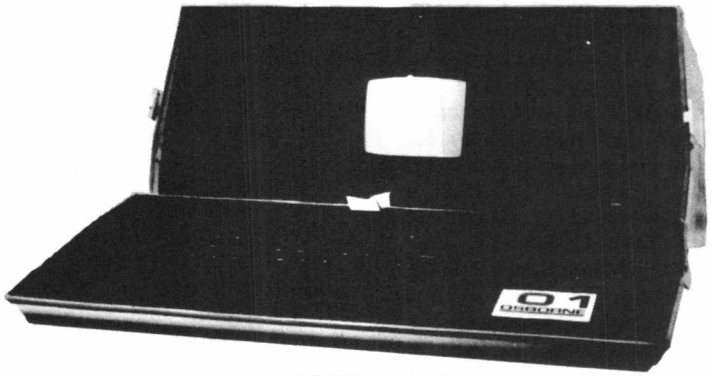
BASYS/1 is the monicker that **TMSI** have used for their all-CMOS computer board. The design places an 1802 CPU, healthy hunks of ROM and RAM, and I/O circuitry on a 4.5" x 6.5" board. A motherboard and memory expansion board are available, and more boards are planned to follow. Prices range upward from $175.

## 64K Chips Are Really Here

**OKI** Semiconductor wants you to know that they are shipping their 64K-bit RAM chips. The ICs are still a bit pricey; they cost from $30 to $60 (depending on speed) in lots of 1000. Still, how long can it be until a 64K-byte board will use just eight chips? Do we dare hope that, when it uses just one extra chip, we might actually see personal computer storage with parity checking? Dream big; single-bit error correction only needs four extra bits per 16-bit word. A board that contains 64K of 16-bit words with error correction can now be built around just 20 ICs. . .

## Osborne I Is Born

Adam Osborne, well known as a publisher of reference books and business software, has launched a hardware venture, the **Osborne Computer Corporation**. Its first product is the Osborne I Computer System, a dual-drive, 64K, CP/M-compatible computer with a retail price of $1795. After its price, the machine's most remarkable feature is its portability; keyboard, screen and drives are all built into a single aluminum carrying case. The only design compromise apparent from the initial press material is the small size of the CRT. Osborne elected to use a 5" CRT, formatted into 24 rows of 52 characters each. The small size may be mitigated by the fact that the small CRT "is a window on a 128 x 34 row screen." We're all eager to see how that works out in practice. ■■




## Contact Points

**Alternet**, POB 4872, Santa Rosa, CA 95402, (707) 542-7200.

**APL Market Newsletter**, POB 5314, Mt. Carmel, CT 06518.

**Elicon**, 254 Viking Avenue, Brea, CA 92621, (714) 990-6647.

**FORTH Interest Group**, POB 1105, San Carlos, CA 94070, (415) 962-8653.

**HDB**, 222 East Anapamu Street, Santa Barbara, CA 93101, (805) 965-4477.

**IMCSS**, Dr. B. C. Kuo, POB 2772-Station A, Campaign, IL 61820, (217) 333-4341.

**Micro-Serve, Inc.**, 250 Cedar Hill Avenue, Nyack, NY 10960, (914) 358-1340.

**North American Technology**, Strand Building, 174 Concord St., Suite 23, Peterborough, NH 03458, (603) 924-6048.

**OKI**, 1333 Lawrence Expressway, Suite 401, Santa Clara, CA 95051, (408) 984-4842.

**Osborne Computer Corporation**, 26500 Corporate Avenue, Hayward, CA 94545, (415) 887-8080.

**Personal Computer Arts Festival**, c/o PCAF-81, POB 1954, Philadelphia, PA 19105.

**Phase One Systems**, 7700 Edgewater Drive, Oakland, CA 94621.

**Softronics**, 36 Homestead Lane, Roosevelt, NJ 08555.

**Software News**, 5 Kane Industrial Drive, Hudson, MA 01749, (617) 562-9308.

**Softworks Limited**, 607 W. Wellington, Chicago, IL 60657, (312) 327-7666.

**Tab Books**, Blue Ridge Summit, PA 17214, (717) 794-2191.

**TSC**, POB 2570, 1208 Kent Avenue, West Lafayette, IN 47906.

**TMSI**, POB 7227, Ann Arbor, MI 48107, (313) 994-0784.

**Westico**, 25 Van Zant Street, Norwalk, CT 06855, (203) 853-6880 for people, (203) 853-0816 for computers.

**Zenith Data Systems**, 1000 Milwaukee Avenue, Glenview, IL 60025, (312) 391-8181.

Since 1976, the most innovative magazine for the technically-sophisticated microcomputer user.

# Dr. Dobb's Journal

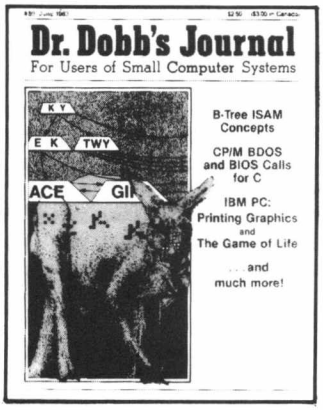
## For Users of Small Computer Systems


dr dobb's journal of $1.50
COMPUTER
Calisthenics & Orthodontia
Running Light Without Overbyte
January, 1976 Box 310, Menlo Park CA 94025 Volume 1, Number 1
A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS
THIS IS A REPRINT OF
THE ORIGINAL
VOLUME 1, NUMBER 1

1976



1983




Have you been looking for a magazine that will provide you with serious articles on programming? *Dr. Dobb's Journal* can give you the technical depth you need to understand the latest software advances. We cover every microprocessor with articles on C, Forth, Unix, CP/M and much more. Our reputation for editorial quality is well known. Subscribe now, and discover what industry insiders have enjoyed since 1976.

Coming articles will cover:

- new language development
- 8- and 16-bit technology
- operating systems
- compilers, editors
- assemblers/cross-assemblers
- clever algorithms

**$25**

one year subscription
12 issues
saves $5 off the newsstand price

**YES!** Enter my subscription to *Dr. Dobb's Journal* and bill me later.

name

address

city state zip

**Mail to: DDJ, P.O. Box E., Menlo Park, CA 94025**

# What to Do With Those Extra Function Keys

by John B. Palmer

*John B. Palmer, P. O. Box 23, Boonville, CA 95415.*

Here is a CP/M program to allow a user to define special functions of his keyboard so that a single keystroke can recall a word or an entire phrase. There has already been a similar program in *DDJ* for users of North Star BASIC. This program is perhaps more versatile.

So you have a CRT with those 16 extra function keys? You say that someday you will do something with them, but you just have not got around to it? I know how you feel. I finally did something about it though, and had some fun in the process. Maybe you would like to know how I did it so that you can be sure to do it different and perhaps better.

## The Motivation

I have a nice Lear-Siegler terminal with lots of fancy keys. Many will never be of any use in a single-user system. I thought it would be nice to define the 16 keys that run along the top of the keyboard. I had seen the same layout on other makes of terminals. Beehive, Televideo and Hazeltine offer the extra keys as standard equipment on higher price models.

After playing with my terminal I discovered that the 16 extra keys all generate a three-character sequence for just one push of the key button. (I could have read the manual and saved some time.) The operator's manual calls this a "Special Function Transmission" that always starts with control-A, which is the ASCII code for SOH, followed by one of the characters @ through the letter O, terminated with RETURN. Use of the SHIFT key gives 16 more functions. Figure 1 is a table to show how this works.

The ASCII code for SOH is 1, the code for RET is 13. So if you hit the F1 key, the sequence would be 1,64,13; if you hit the F16 key it would be 1,80,13.

From this I could see that the SOH is a lead-in character. It means that a function code follows. The RET character is not needed in my application, so I must plan to discard it.

It seems easy. Just write a keyboard input routine that tests to see if the SOH character ever comes in. Hold on! That means I have to redo my CP/M BIOS. And what if I ever want to change it again? And what about some of my software that already uses the SOH for a special purpose.... This sounds like a lot of work. So I put it off for awhile. Needed to think it out.

**Figure 1.**

| Function Key | Sequence | Sequence with SHIFT |
|---|---|---|
| F1 | SOH, @, RET. | SOH, ', RET. |
| F2 | SOH, A, RET. | SOH, a, RET. |
| F3 | SOH, B, RET. | SOH, b, RET. |
| F4 | SOH, C, RET. | SOH, c, RET. |
| F5 | SOH, D, RET. | SOH, d, RET. |
| F6 | SOH, E, RET. | SOH, e, RET. |
| F7 | SOH, F, RET. | SOH, f, RET. |
| F8 | SOH, G, RET. | SOH, g, RET. |
| F9 | SOH, H, RET. | SOH, h, RET. |
| F10 | SOH, I, RET. | SOH, i, RET. |
| F11 | SOH, J, RET. | SOH, j, RET. |
| F12 | SOH, K, RET. | SOH, k, RET. |
| F13 | SOH, L, RET. | SOH, l, RET. |
| F14 | SOH, M, RET. | SOH, m, RET. |
| F15 | SOH, N, RET. | SOH, n, RET. |
| F16 | SOH, O, RET. | SOH, o, RET. |

Later, I got a Sinclair ZX80 and discovered that it has built-in, single-stroke keywords. Real nice, just hit the SHIFT and a letter and a word appears on the screen. But only the words used in BASIC programming. There is no provision to redefine the letters to represent a new set of words. Still need to think it out.

Then along comes *DDJ* number 50. Leonard E. Garcia has done it for me! Well, almost. He has a North Star system. I have a one-of-a-kind collection of odds and ends that is held together by the CP/M operating system. If I can make this idea work on my system it should be usable on any microcomputer with CP/M.

So I would take on this project only if I could make a program that meets these requirements:

1. It does not require the user to rewrite the BIOS software;
2. When not in use it does not hinder other software that uses SOH code (control-A);
3. The user may rewrite the definitions and save them on disk;
4. It is hardware independent and even works on the Apple II.

Is this going to be hard to explain to someone in your family or group? Can anybody use it without a course in assembly language? Yes. It is very easy to use. Here is a description of how to make use of this method of Keyboard Input Definition.

## The Procedure

After you power up your system and hit reset, or whatever button it is that you push to get the disk to boot, give the DIR command and see if the following files are on the disk:

DDT.COM
KIP.HEX
KID.COM
CPM.KID

Now give the command "DDT KIP.HEX"' The words "NEXT", "PC" should appear and some letters and numbers. If all is well, give the command "GO".

It is not necessary to repeat the above procedure until the computer is again reset or the power is turned off and back on.

At this point you may give the command "KID CPM". But if you are going to be doing BASIC programming, you may rather give the command "KID BASIC" if your disk has BASIC.KID on it.

Anytime you are at the CP/M command level you may redefine the special function keys with the command "KID filename", where filename is a proper file of type ".KID" and is on your disk.

If the filename you gave is not on the disk, the message "NO KID FILE" will appear. You may have spelled the filename wrong. Or else it is not on your disk.

It is possible that you may get the message "KIP ABSENT". This means that you forgot to load "KIP.HEX" with DDT or else the computer has been reset.

When everything goes right you will get this message: "KID READ OK". Now hold the CTRL key down and strike the letter A twice. You should see the contents of the file that was just read and now defines the Special Functions of the keyboard.

You may wish to use transparent tape and a marker pen to identify keys F1 through F16. The definition for F1 is quotation after the "@" sign, F2 corresponds to "A", F3 to "B" and so on. (Sometimes the "¦" is used in place of

a quotation mark.)

From now on, whenever you hit the F1 key the word or words inside the quotation marks after the "@" sign should now appear. If you hit the F2 key the quotation after the "A" will appear and so on. Notice that the quotation marks do not show. Neither does the "@" or the letters that identify the 16 keys F1 through F16.

Remember, whenever you wish to see the contents of the current Keyboard Input Definition, just hold the CTRL key and strike the letter "A" twice. If you do this while using a word processor program, be sure to clear the screen or home the cursor before you do any more editing.

You may use the text editor to make your own custom definitions and save them on disk. Use the present definitions as a guideline for your work. Remember to put quotation marks before and after each definition. You do not have to define all 16 keys. If for some reason you must have a quotation mark itself as part of a group of words of a definition, then use some other symbol to mark the start and finish of that group of words.

If you have made an error in your custom definition, some of the function keys will give bizarre results. Anytime you change your definitions you must give the command "KID filename" again in order to put the new definition into its proper place in memory.

If you get a question mark when you strike a function key it means that key was not defined. An improper definition may also cause a question mark to appear.

## The Theory

So much for the user guide. Now about the theory. Really, I don't have a very good explanation about how it works. One of the requirements of this system is that you have about 512 bytes of RAM somewhere in your system that can be used by the programs "KIP" and "KID". Let's call its first address FRERAM.

The program "KID" is an almost-normal CP/M program that is called into the TPA. It can be used as often as you like. It reads into FRERAM + 256 the definition file the user specifies.

The definition file must be of type ".KID". Thus a file that might be used in BASIC programming could be named "BASIC.KID". Of course you can name it anything you like, just make the filetype ".KID". I have included in the listings my personal definitions for CP/M and BASIC.

If I wish to do quite a bit of work with CP/M at the command level, I give the command "KID CPM". CP/M loads KID.COM into the TPA. It in turn loads the sequential file "CPM.KID" into FRERAM + 256. Next a test is made to see if the file "KIP.HEX" has already been called in by DDT. There should be a JMP instruction at FRERAM. If so, then another test is made to see if the address field after the JMP is all zeros. If it is all zeros then a patch must be made to the address fields of BIOS vectors for Console Status and Console Input. But first we must store those address fields into FRERAM + 1 and FRERAM + 4 or we will really mess up everything.

From this you can understand that this method is really very risky. Do something wrong and the whole system will crash. It really would be better if there was a nice clean way of doing this trick inside the CP/M system rules. I consider this method to be very nasty and against the rules. You can not use KID with the CP/M XSUB utility. The XSUB program also will modify the BIOS vectors. I tried to use the two together and they did not respect each other. But there are few cases where anybody would use XSUB.

Now what about using KID with any program that wants to use the SOH character? You can do either of two things. One is to just reset your computer and reboot. This will restore the BIOS to its original condition. Or you can define a special function to generate a control A, which is SOH. I did it like this:

q"↑A"

So if I hit control-A and then lower case Q, I get a control-A. Why not define control-A as being itself? Because the program KIP (Keyboard Input Patch) expects control-A to be the KIFUN lead-in. If you want, you could edit the program KIP.ASM, define some other control to be the KIFUN constant and assemble it. But most terminals use control-A as the lead-in. What about the ESC key? That would be alright if your terminal does not use ESC as the lead-in for cursor address, clear screen or whatever.

This leads to a consideration of how the program KIP does its job. Because KIP must be loaded into an area above the BIOS, it can not be a regular CP/M ".COM" file. You have to use DDT to read and load KIP.HEX into your free RAM area. On my system this is at 0FC00 hex and that value is given the constant FRERAM on line 14 of the listing for KIP. Change this if your system has reserved RAM at some other location. If you are now running a 64K CP/M system, you will have to move CP/M down to 63K to make room for KIP.

All console status calls and console input calls via the BIOS are vectored first to KIP. If the SWITCH is inactive, KIP will cause a call for status to be vectored to the original status routine in the BIOS. Whenever there is a call for the console input, KIP calls the input routine in the BIOS and then tests for a KIFUN character. (KIFUN is control-A, unless you change it to something else.) If it is not the KIFUN character then a normal re-return is made and the calling program gets to have the character that just came in from the keyboard. But if it was the KIFUN character, sparks start to fly. A call is made again to the original BIOS input routine. If it is another KIFUN, then a jump is made to PEEK so that the user gets to see what is in the current Definition Block; otherwise the Definition Block is searched for a definition that corresponds to the character that followed KIFUN.

One minor note. I did not want the RETURN character that is given whenever keys F1 through F16 are hit. If the character after KIFUN is in the range of 40 hex through 4F hex or in the range of 60 hex through 6F hex, then a dummy call is made to the BIOS input routine to destroy the unwanted RETURN.

Now then, where were we? Oh, yes, a search was made for a definition. If found, the SWITCH is turned on, the EOSTR (end of string) is noted, the STRPTR (string pointer) is set to the first character of the string and we start giving the string a character at a time. When we get to the last character it is time to turn off the SWITCH and restore normal console input.

What are the rules for the Definition Block? The Definition Block is nothing more than a text file created by the user with any standard CP/M editor. TAB, RETURN and LINE-FEED are allowed, but do not put other control codes nor high parity bits in the file. The first character in the file other than SPACE, TAB, RETURN and LINE-FEED identifies the start of another definition. This goes on until the end of the file is found. The CP/M editor automatically puts the end

of file marker in the file although you don't see it.

Some examples

Example 1
@"STAT " A"ASM" B"BASIC"

Example 2
1"PIP↑M" 2"DIR↑M"

In the first example there are three definitions. When the user hits the F1 key, the word STAT will appear. If F2 is hit the word ASM will appear. If F3 is hit the word BASIC will appear.

The second example has two definitions. The function keys are not used here. When the user hits control-A followed by the number 1 the command PIP will be given. When the user hits control-A followed by the number 2 the command DIR will be given. In both cases the up arrow and the letter M represent the control code for RETURN. Then how could you ever put an up arrow in the definition? Use two. Notice example 3:

Example 3
8"↑>" 9"↑↑"

These two definitions have little practical value, but show how to put control up arrow and up arrow itself into the Definition Block. Control-A followed by 8 gives control up arrow. Control-A followed by 9 gives up arrow.

You may use other symbols for the string delimiter instead of the quotation mark. In BASIC programming the quotation mark is used as an unchangeable string delimiter. In the Definition Block for KIP the string delimiter is always whatever comes after the string identifier. Notice example 4:

Example 4
6¦FILE$="¦ 7¦RUN FILE$¦

In example 4 above we wanted to put the quotation mark into the definition. We used a funny character that is found on some keyboards by hitting the SHIFT and the backslash. In fact, the backslash could have been used too, but some BASICs use the backslash to mean something. Try to pick a string delimiter that you know will never have to appear inside the string. You can not use the string delimiter inside of the string. It only and always comes at the start and the finish.

## Improvements

So much for the theory. What about improvements? That is up to you. I tried to keep the KIP program inside of 256 bytes using 8080 code. If you wish to modify either KID or KIP, go ahead.

These programs are not of commercial quality. There are too many things that can go wrong. A better method would be to put this sort of thing into the operating system itself instead of grafting it into the BIOS. As mentioned earlier, it can not be used with XSUB. Hopefully, Digital Research will put something like this into a future version of CP/M. ■■

**LISTING ON PAGE 34**

Do you have a microcomputing manuscript
that seems too advanced for most magazines,
but too exciting for an academic journal?
*DDJ* may be the niche
you have been looking for.
Let us hear from you at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# CAD for Time Delay Circuits

In the September 1979 issue of *Popular Electronics*, there is an excellent article called "A Monostable Catalog for Experimenters." This article, by Clement S. Pepper, outlines the key features of the most popular monostable IC's. With the article as a guide, an experimenter can design his own one-shot timing circuits.

Even with the article, however, I found my self spending far too much time "designing" before I could construct a working circuit. My time was spent making numerous design decisions and solving equations to come up with correct timing components – tasks that could be done easily with a hobby computer. With that idea in mind, I spent an evening developing a BASIC computer program that would take all the drudgery out of IC design.

Once you hit RUN, the computer asks you for the most important design information only. It then goes on its own to determine the closest standard value parts needed. The program design is straightforward and has been structured so that it is self-documenting.

Lines 265 through 340 allow you to choose an IC chip you have on hand if it can supply your needed pulse width.

Lines 360 through 595 make the design limit decisions for you. The program is quite conservative when setting design limits.

Lines 615 through 655 ask for your required pulse width. These lines also test your requirements against the design limits, just to keep you honest.

Lines 675 through 715 ask for the standard value capacitor you wish to use and check it against the design limits. The program could have been designed to compute a specific value capacitor, but the method actually used has one big advantage for the experimenter – he can use parts values he already has on hand. If you are like me, your junk box will contain a part close enough for your design. By the way, if you don't have a capacitor on hand, I suggest you use a value that is a multiple of 10, 22, or 47. You should be able to find these values in any parts store.

Lines 725 through 780 compute the proper timing resistor value for the chip and capacitor you have chosen. The computed resistance is checked against design limits and, if it doesn't work, the program will ask for another value capacitor. It usually takes two or three tries before the program presents a usable set of values.

Lines 800 through 825 print out the results of a successful choice of values. However, the resistance called for will not be a standard value since it is computed to six- or eight-digit accuracy.

Lines 845 through 890 ask for the closest standard value resistor you can use. I suggest you pick a value from a list of standard 5% values. As with the suggested capacitor values, 5% resistors can be bought in almost any parts house.

Lines 910 through 965 compute the actual pulse width you should get using the standard values you have entered.

If you require closer tolerance pulse widths, you should use an adjustable resistor. Try a ten- or twenty-turn adjustable resistor with a value about twice that computed for R by the program. Also, use a fixed resistor in series with the adjustable one. The fixed resistor should have a value about 20% of the value computed by the program. This combination will allow you to set your pulse width right on-the-money.

Now that my computer can call upon this program, I can design a one-shot multivibrator in just a few mintues. Not only that, but most of the time, my designs use parts already on hand. I hope this program helps you do the same. ■■

**LISTING ON PAGE 39**

```
RUN
COMMON ONE-SHOT CHIPS   (POSSIBLE PULSE WIDTHS)
1.  9600 (74 NS – 8 SEC)          6.    74123 (40 NS – 8 SEC)
2.  9601 (45 NS – 8 SEC)          7.  74LS221 (30 NS – 70 SEC)
3.  9602 (72 NS – 8 SEC)          8.   74C221 (50 NS – 175 SEC)
4. 74121 (30 NS – 14 SEC)         9.    4098B (30 NS – 270 SEC)
5. 74122 (40 NS – 8 SEC)         10.  4528B/ (100 NS – 270 SEC)
                                       MC14528C
WHICH CHIP WILL YOU USE ?  (1 – 10)  6
WHAT LENGTH PULSE IN MICROSECONDS ?  10
WHAT STANDARD VALUE CAPACITOR IN MICROFARADS ?  1
USE A SMALLER VALUE C
WHAT STANDARD VALUE CAPACITOR IN MICROFARADS ?  .1
USE A SMALLER VALUE C
WHAT STANDARD VALUE CAPACITOR IN MICROFARADS ?  .01
USE A SMALLER VALUE C
WHAT STANDARD VALUE CAPACITOR IN MICROFARADS ?  .001
USE A LARGER VALUE C
WHAT STANDARD VALUE CAPACITOR IN MICROFARADS ?  .0047
USE A SMALLER VALUE C
WHAT STANDARD VALUE CAPACITOR IN MICROFARADS ?  .0022
FOR A PULSE WIDTH OF  10 MICROSECONDS, USE:
   CAPACITOR =  .0022 MICROFARADS
   RESISTOR APPROX =  14204 OHMS
WHAT IS THE CLOSEST STANDARD VALUE RESISTOR YOU
CAN USE ?  15000
USING A CAPACITOR OF .0022 MICROFARADS AND A
RESISTOR OF  15000 OHMS WILL RESULT IN A PULSE WIDTH OF
10.560493 MICROSECONDS.
   ANOTHER DESIGN ?  N
READY
```

## by Hal Knippenberg

*Hal Knippenberg, 2514 Blueberry Drive, Augusta, GA 30906.*

Reprinted from the *CSRA Computer Club Newsletter*.

# Book Reviews

**Disassembled Handbook for the TRS-80**
By **Robert M. Richardson**
Published by **Richcraft Engineering, Ltd.**
**$10.00**

**TRS-80 Disk and Other Mysteries**
By **Harvard C. Pennington**
Published by **IJG, Inc.**
**$21.95**
Reviewed by **John R. Culleton, Jr.**

You can always spot the future programmer in a gang of pre-schoolers. Just look for the tyke who has dismantled a perfectly good music box in the vain hope of catching a glimpse of the tiny orchestra inside. Radio Shack has provided software for the TRS-80 aimed at obedient appliance operators, not prying tinkerers. The two principal programs (Microsoft BASIC and Randy Cook's TRSDOS) are slick, easy to use, and relatively impenetrable. A challenge of this character is not to be ignored, and two venturesome technologists have attacked the Tandy software offerings with ingenuity and persistence. The results are fascinating.

The books are complementary in that Richardson deals with the subroutines found in the Microsoft BASIC, and Pennington discusses the organization of TRSDOS files and the file directory system. There is little overlap of information. Both books are self-published, set in large clear type using the TRS-80 and the Electric Pencil program. Pennington has a handsome product, with illustrations by the author and carefully drawn graphics to hold the reader's interest. Richardson has turned out a more utilitarian volume, and could have spent more time on proofreading. Both present plenty of fascinating inside information on how the TRS-80 software does what it does, and both leave plenty of room for further investigation by the reader. Both feel obligated to castigate Tandy Corporation with tiresome regularity. Richardson at least has the grace to offer this curious disclaimer:

> "Please disregard the 'curse you, Red Baron' comments in chapters 2 and 3. They were put in for reviewers/publishers who were given only a single chapter of this handbook."

We leave it to others to explain why public and frequent excoriation of Tandy Corp. and all its works is considered a required exercise. This reviewer neither understands nor appreciates this curious form of reverse partisanship on the part of TRS-80 users.

If Mr. Pennington has problems finding any virtue in Tandy, he has equal difficulty finding any vice in his two favorite software products – NEWDOS+ and Electric Pencil. Nevertheless, on page 82 under the heading "Electric Pencil Goodies" he lets slip the fact that the two are mutually incompatible as sold. He does give the necessary modifications to Pencil to correct the problem, but this reviewer wonders what he would say about a Tandy product that had the frequent compatibility problems of NEWDOS. (Microsoft FORTRAN and Percom PATCH-PAK are among those programs that won't work with NEWDOS, by the way.) Where Pennington really shines is in his lucid and exhaustive explanation of the intricacies of the many file formats used under TRSDOS, and the inner workings of the directory. There are many file and directory layouts illustrating the variants, from TRSDOS to VTOS and from Electric Pencil files to those created for the Microsoft Macro Assembler. Pennington is particularly strong on file recovery techniques. He has unlocked most of the secrets, although the "hashing" algorithm for file names eludes him. His clever use of graphics to highlight file formats is an excellent feature. It should be noted that Pennington deals with TRSDOS files and not with the activity of TRSDOS itself. There is no penetration of the operating system routines proper. Perhaps that will be in his next book. Whatever the limitations and annoyances, Pennington has written a "must have" book for all card-carrying Tandy tinkerers.

Richardson plumbs the innards of TRS-80 software more directly, specifically the Level II BASIC found in the TRS-80 ROM. But while Pennington packs 128 pages with information, Richardson has to use considerable padding to fill 69. For example page 37 to 52 of Richardson consist of a dump of the Level II ROM. Not a disassembly, mind you, just a dump, presumably produced by the ubiquitous SUPERZAP program. He has a real problem. He has come in possession of a body of immensely valuable information, but only needs a dozen or so pages to tell all he knows. So he pads. Never mind, the information he provides is well worth the ten bucks. He has unveiled the locations of the Level II ROM routines and learned how to use at least some of them in assembly language programs. An assembly language program consisting primarily of calls to well-written subroutines has obvious advantages in brevity, readability and efficiency over the handcrafted variety. Armed with Richardson's book a clever programmer could even develop his own BASIC compiler and really make the TRS-80 zip along. Curiously, he fails to decode the Level II shorthand notation for BASIC reserved words (verbs and such). No matter, Pennington lists them in an appendix. One could wish for greater variety of "worked examples" covering routines in addition to the arithmetic functions and data type conversion. Nevertheless, the book whets the appetite very well, and few readers will resist the temptation to try their hand at direct calls of ROM functions.

This reviewer has found Ms. TRS-80 to be a demure and secretive female. Pennington and Richardson have exposed some of her fascinating features. Of course, even more fascinating are the secrets yet kept, such as the inner workings of the file access routines. Sally Rand and Gypsy Rose Lee well understood the principle. We will all be a little sadder when the last veil is dropped and there are no more mysteries in Mr. Tandy's marvelous creation. In the meantime, all dedicated software voyeurs will have to have both Richardson's *Disassembled Handbook* and Pennington's *TRS-80 Disk and Other Mysteries.*

**Pascal Programming**
By **Laurence Atkinson**
Published by **Wiley & Sons, Inc.**
**$49.50 hardbound, $21.00 softbound**
**428 pages**
Reviewed by **Doug DeGroot**

With the ever-increasing popularity and acceptance of Pascal, it is inevitable that a flurry of Pascal instructional books will continue to appear. *Pascal Programming* by Laurence Atkinson is one of the most recent of these. The book has a dual purpose: to teach Pascal to experienced programmers and to teach progrmming to those with no previous experience. It is intended for use as a main text in a teaching course. These two aspects combine to give the book most of its problems, although the author does an admirable job of overcoming them. The first few chapters, intended to introduce programming to novices, suffer from poor organization and lack of clarity. It is doubtful whether or not a true novice can, without an instructor available to answer his questions,

wade his way through the material and come out with sufficient understanding to complete the remainder. Truly experienced programmers wishing only to learn Pascal will almost certainly find the book too slow, too verbose, but certainly not useless. Medium level programmers and those with their own home computer who have mastered BASIC will best find the book suited for their needs.

First, a few of the faults. The coding conventions used throughout the text are abysmal and even inconsistent. They can easily be tolerated since most of the program examples are short and not complex, but they definitely should not be adopted. Occasionally a programming function or concept is used before it is introduced and explained. Also, some of the author's stylistic attitudes are highly questionable and are not to be recommended.

As for good points, the book is well written, explanations are well worded (if not always lucid) and there is an abundance of material. The organization is highly conceptual – a concept such as looping is introduced and then given a thorough and detailed examination. One problem with this approach when accompanied by the author's dual purpose is that programming constructs often get introduced painfully late. The author states that his solution to this problem is skipping around the chapters as needed. Readers unfamiliar with the text's contents and purpose may not readily be able to do this on their own. A wealth of examples accompany each concept, and each chapter contains a comprehensive list of challenging exercises. Concepts covered include different data types, loops, procedures and functions, program construction, scalar types, and advanced data structures. A set of short, useful appendices is included.

**How To Build Your Own Working Microcomputer**
By **Charles K. Adams**
Published by **Tab**
**$9.95, 308 pages**
Reviewed by **Carl E. Whitney**

This book is ill-conceived, and its audience poorly targeted. Beginners would be better off with a kit, since much of the information presented here is in a condensed, formal style. Experienced hobbyists will not be very interested in an 8080 machine with hexadecimal input and LED display. Adams should devote more thought to his potential audience and their information needs – and he should insist that his publisher not print a cover depicting a typewriter-ASCII input and video output, when the project includes neither.

There is some useful material here on the hardware and software required to program EPROM's – but periodic typographical errors make one wonder how carefully the book has been proofread, and how bug-free the programs are.

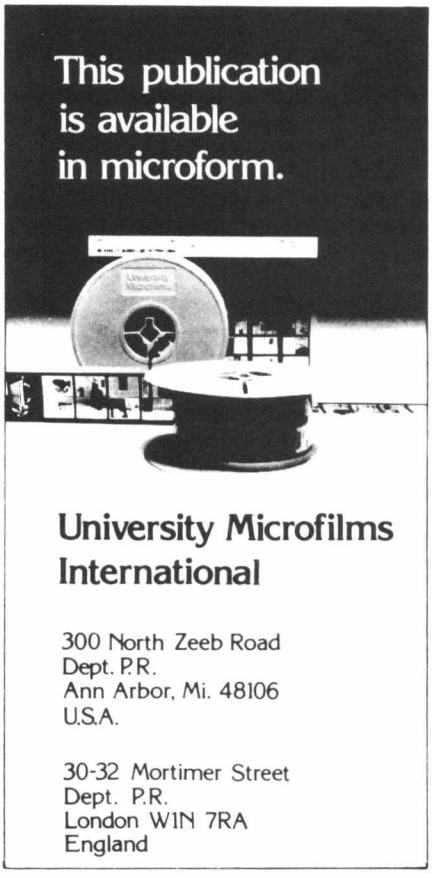
**The Dartmouth Time Sharing System**
By **G. M. Bull**
Published by **Halsted Press (John Wiley & Sons)**
**$65.00, 240 pages**
Reviewed by **Edward T. Ordman**
**New England College**

It is hard to understand the high price of this book. Does the publisher feel there will be enough library sales to pay for the publishing costs, and sales to individuals can be neglected? It is an excellent book, and could be very widely recommended at one-third of the price. At this cost, it can be recommended only for the libraries of schools or companies which actively use the Dartmouth Time-Sharing System (DTSS) or which are engaged in writing a time-sharing system large enough to require man-years of effort. Perhaps others can read it by inter-library loan.

DTSS, while physically installed at about a dozen sites, is far more widely used: Dartmouth College is tied into all the major networks, and DTSS was a major formative influence in eduational computing. The commands LIST, RUN, SAVE and others common nowadays first existed (about 1964) as system commands in DTSS.

The book is a very technical, but very readable, distillation of a great many manuals and technical documents about DTSS. It is not an introductory user's manual, and does not replace the standard manuals; it is very useful reading for someone who has a modest acquaintance with the system and would like to become an expert. More important, for the person interested in internals of operating systems, it gives a great deal of detail about the parts of DTSS and how they communicate with one another. Organizations of buffers, flags, job tables, details of fault handling, communication between jobs, and techniques used in deadlock avoidance are described in detail. There is an excellently documented list of parameters used in adjusting scheduling and of the consequences of changing parameter values. There is a good explanation of swapping algorithms (determining which jobs should be in core at any moment). There is a discussion of how it looks to the operator, jobs to be performed by the operator and by the computer in starting the system, stopping the system, modifying the operating system.

DTSS is definitely a "big computer" system – at least in 1981. As the operating systems of small computers grow, more small computer users will want to know about big operating systems and the techniques they use. DTSS is conceptually simpler and more user-oriented than many large systems, and this book is full of ideas that will be valuable to someone writing a large operating system. It does not replace a textbook on operating systems, since it refers purely to DTSS and does not discuss other possible solutions to the problems faced by operating systems; but it discusses the solutions used in DTSS in far more detail than can be done in a more general text. ■■

# Dr. Dobb's Clinic

D. E. Cortesi, Resident Intern

*Doctor Dobb's Clinic* is a new venture. It is a place for the display of techniques and discoveries. We want to cover any sort of method or trick you've found in your exploration of (for example, but not limited to): CP/M or OASIS or FLEX or APPLE DOS or TRSDOS or NEWDOS or RSTS or UNIX or UCSD PASCAL or . . .

During our rounds at the clinic we may examine a compiler or an interpreter, revealing its bugs or showing how it can be made to perform better. We'd like to tell of unobvious uses for standard utilities. We want to uncover errors in published documentation, to warn people away from pitfalls, and to show off those *eureka!* moments that make systems work rewarding.

This first appearance of *Doctor Dobb's Clinic* has used up all the material we have at hand. It can't appear again until we have more things to tell. Where will that material come from? From you, the readers, or not at all. The requirements are very simple, much easier than those for an article.

Write to us. Describe your discovery or technique in a couple of pages, in any format you like – so long as we can read it. Don't worry about being "literary." Don't bother with a tutorial introduction for the novice reader; this isn't a column for novices. Do include a daytime phone number so that if we have questions we can call you.

The items that follow are examples of the kind of thing we think the Clinic should contain; if you think differently, send us what you have.

## Getting 'Round DDT

DDT, CP/M's degugging tool, can be started against a program file from any drive by naming that file as the command operand. For example,

```
A>ddt b:first.hex
```

will load DDT, which in turn will load FIRST.HEX from the B-drive. Once it's started, DDT doesn't want to load further program files from any drive but the default one. You can try it:

```
-ib:second.hex
-r
```

but DDT will insist on searching for SECOND.HEX on the A-drive (or whatever the default drive currently is).

DDT's R subcommand uses the default File Control Block (FCB) at 5Ch to open and read the file. The I subcommand is just a way of initializing that FCB with the name and type of the file to be read. The problem is that the I subcommand ignores the drivecode. It always leaves the drive number in the FCB set to 00h, meaning "use the default drive," regardless of what drive you said to use.

The S subcommand can be used to circumvent this irritating restriction. Use it to set a drive number (1=A, 2=B, and so on) in location 5Ch:

```
-isecond.hex
-s5c
005C 00 02
005D 53 .
-r
```

Now the FCB is set up to search the B-drive, and the R subcommand will find the file you want.

## A Handy Approximation

We are working on a kid's game we call "Flea Circus." In it, a bunch of fleas (spots) hop randomly around the screen. The player controls a spider with an endless appetite for fleas. The object of the game is to maneuver the spider onto one flea after another until they're all eaten. To make the game harder,we want the fleas to avoid the spider by hopping harder when the spider is nearby.

But what does "nearby" mean, in the terms of the program? Spider and fleas are represented by screen coordinate pairs (h,v). A flea's distance from the spider could be computed with the Pythagorean rule:

```
dh = abs( h(flea) - h(spider) )
dv = abs( v(flea) - v(spider) )
d  = squareroot( dh*dh + dv*dv )
```

but that would mean several floating-point operations per fleahop, and we want to support a *lot* of fleas. We need a fast approximation to the hypotenuse that can be done on integer values.

And we've stumbled onto one. Calculate dh and dv as above. Then,

```
dapprox = max(dh,dv) + min(dh,dv)/2
```

That is, the hypotenuse of a right triangle is approximately the sum of the longer side plus half the shorter side.

We dusted off our memories of algebra and worked out that the approximation is dead accurate when

```
min(dh,dv) = 0
```

that is, when flea and spider are on a horizontal or vertical line and the triangle degenerates to a straight line. We had some trouble analyzing the worst case, but a check with this program reassured us:

```
FOR DH=1 TO 20
   FOR DV=1 TO DH
      APPROX = DH+(DV/2)
      ACTUAL = SQR(DH↑2+DV↑2)
      ERRPCT = INT(100*(ABS(ACTUAL-APPROX)/
         ACTUAL))
      PRINT DH,DV,ACTUAL,APPROX,ERRPCT
   NEXT DV
NEXT DH
```

It would appear that the approximation is never worse than 12% off. Surprisingly, when changed to shift instead of divide (use "APPROX=DH+INT(DV/2)") it is still never worse than 12% off, except for the case of dh=dv=1. Not too shabby for an algorithm that uses only two compares, an add, and a shift!

The Doctor opines that an approximation this neat must be common knowledge among people less ignorant than we. Does any reader know if it has a name, and what (presumably famous) mathematician found it first?

## Comments In Submit Files

CP/M's Submit command can be very useful. It could be even more useful if there were any way to put comments and operator instructions into a submitted file. But all submitted lines pass through the Console Command Processor (CCP), and the CCP has no format for a comment statement. Or so the CP/M manuals would lead you to believe.

Surprise! The CCP has a comment command format. If an input line begins with a colon, the CCP echoes it to the terminal and (apparently) does nothing else. Therefore, you can decorate submit files with all the comments you like:

```
: THIS WILL SHOW ON THE SCREEN BUT DO NOTHING
:   AND SO WILL THIS.
DIR *.COM
: . . .BUT THE ABOVE WAS A COMMAND.
```

Comments are also useful when you've tied the printer to the terminal (with control-p) in order to create hardcopy documentation of an operating procedure or of a bug.

Doc observes that since Digital Research hasn't documented this feature, they aren't committed to supporting it in later releases. Does any reader know whether MP/M or CP/M-86 supports it?

## Handling MP/M's Integer Date

MP/M contains a Get Date and Time service request. It returns the time of day in BCD, very handy for printing. It returns the date as a 16-bit integer representing a count of the number of days that have elapsed since 31 December 1977. If we read the manual right, on 1/1/78 the date value would have been 0000h; on 1/2/78 it would have been 0001h, and so on.

Now, that's a handy format for some purposes. You can put the date in a file at the cost of only two bytes; you can compare dates as integer values. What is not simple is the conversion of that number into a year and a Julian day number, or into a year, a month, and a day.

If it weren't for leap years, conversion would be simple: just divide the number by 365. the quotient (plus 1978) would be the year and the remainder (plus 1) would the be day number. Unfortunately, leap years foul things up. 1980 was one, and 1984 will be another. A program you write now might well be in use then, but you probably won't be nearby to fix it when it comes up with the wrong date. What's needed is a general algorithm for conversion, one that will still be working when the last MP/M system is retired to the Smithsonian around 1997.

It's a darn shame Digital Research didn't choose to originate their date value at the start of a leap year, because we know a cute algorithm for conversion that only works under that condition. Of course the MP/M date can be adjusted by subtracting 730 from it; that moves day zero to 1/1/80. Assume we've done that. The date value will have two high-order zero bits until 2022. Therefore it can't hurt to multiply it by four (shift it left twice). The average length of a year is 365.25 days; four times that is 1461. The integer division

```
( mpmdate shl 2) div 1461
```

is a sort of very-short-precision float divide by 365.25. Thus:

```
quotient  = (mpmdate shl 2) div 1461
remainder = (mpmdate shl 2) mod 1461
      year = 1980 + quotient
       day = 1 + (remainder shr 2)
```

which comes down to shifts and increments, plus one call on a standard 16-bit integer divide routine. If you don't believe something that simple can cope with leap years, here it is in Microsoft BASIC. Try it out:

```
DEFINT A-Z
FOR MPM=0 TO 1462
   MPMSHL2 = MPM*4
   YEAR = 1980 + MPMSHL2 1461
   DAY  = 1 + (MPMSHL2 MOD 1461) \ 4
   PRINT MPM,YEAR,DAY
NEXT MPM
```

This algorithm goes awry in 2000, which isn't a leap year, but a 19-year span isn't bad. The problem of getting from the day number to a month and day is something else again. Has anyone a good trick for that?

## The Doctor is Aghast

We recently tried to use the built-in functions MIN and MAX and were astounded to find that MBASIC 5.0 seems to lack them. Maybe that isn't hot news to habitual users of BASIC, but we try to avoid BASIC as much as the micro milieu will allow. We just sort of assumed that those useful functions, solid stand-by primitives in every other programming language, would be there when we needed 'em. Who could conceive of a language without MAX and MIN? Microsoft, apparently.

OK, so BASIC doesn't have them. It has a DEF FN statement; we'll make our own. Or so we thought, until we tried, and ran up against a stone wall. MBASIC allows only expressions, not verbs, in functions, so the obvious

```
DEF FNMIN(X,Y) = IF X<Y THEN X ELSE Y
```

is out of the question. And we can't think of any *expression* on two values that will always yield the lesser of the two values. Can a reader help?

## Length of an 8085 Instruction

Tom Martin sent us the following listing, demonstrating a compact method of calculating the length of an 8085 instruction from its opcode. Can you find a more compact one? The Doctor opines that the 8086 would be rather more difficult. Has anyone a solution for it? ■■

LISTING ON PAGE 32


### Are you ready?

*DDJ*, the world's foremost microcomputer publication, has been working for years to prepare its readers to be innovators, to lead the wave of breakthroughs in our changing technology.

Every issue of *Dr. Dobb's Journal* helps one to understand the nuts and bolts of small computer systems. We offer entire listings of valuable software: our pages have included compilers, cross-assemblers, editors, new languages, hardware interfaces and more – usually before anyone else thinks of them!

### Even more important!

Our subscribers share insights, correspond, and contribute to one another's work, more than any other group we know. They treat *Dr. Dobb's Journal* as a "hands-on" publication.

This warm cooperation has done more to refine software products, and generally to advance the state of microcomputer technology, than perhaps any other resource. And it is available to you through our pages!

### For the straight facts . . . . . .

If you are a serious computing professional or enthusiast, then you should take a very close look at what *DDJ* offers you. We've been on the cutting edge since 1976.

# Santa Barbara Tiny BASIC for 6809's

## by James A. Hinds

*DDJ is pleased to present this version of Tiny BASIC for the 6809. It is a direct descendant of Li-Chen Wang's Palo Alto Tiny BASIC (refer to* DDJ *#5) which its author, James A. Hinds, has named Santa Barbara Tiny BASIC.*

*SBTB is an example of a transliteration of a large 8080 program, and contains a number of 6809 techniques which should be useful to those who want to get familiar with the 6809. The program was developed on an Apple II with The Mill (which puts the 6809 under the hood) and Technical Systems Consultants' operating system, FLEX.*

*The following introductory paragraphs are the same as those printed with the original Palo Alto Tiny BASIC. The few differences in documentation can be noted as follows: in this version, there is no Control O function; NEXT statements do not require a variable name; there is no overprint function; and no spaces are allowed in the input language except in GO TO commands.*

*In addition, this program is completely relocatable on a page basis to any position in memory. It requires about the same amount of space as did Li-Chen Wang's PATB. – Ed.*

*Original Introduction to Palo Alto Tiny BASIC by Li-Chen Wang*

**Numbers.** All numbers are integers and must be less than 32767.

**Variables.** There are 26 variables denoted by letters A through Z. There is also a single array @(I). The dimension of this array is set automatically to make use of all the memory space that is left unused by the program. (I.e., 0 through SIZE/2, see SIZE function below.)

**Functions.** There are 3 functions:
ABS(X) gives the absolute value of X.
RND(X) gives a random number between 1 and X (inclusive).
SIZE gives the number of bytes left unused by the program.

**Arithmetic and Compare Operators:**

/ divide.
* multiply.
- subtract.
+ add.
> greater than (compare).
< less than (compare).
= equal to (compare).
# not equal to (compare).
>= greater than or equal to (compare).
<= less than or equal to (compare).

+, -, *, and / operations result in a value between -32767 and 32767. (-32768 is also allowed in some cases.) All compare operators result in a 1 if true and a 0 if not true.

**Expressions.** Expressions are formed with numbers, variables, and functions with arithmetic and compare operators between them. + and - signs can also be used at the beginning of an expression. The value of an expression is evaluated from left to right, except that * and / are always done first, and then + and -, and then compare operators. Parentheses can also be used to alter the order of evaluation. Note that compare operators can be used in any expression. For example:

```
10 LET A= (X>Y)*123+(X=Y)*456+(X<Y)*789
20 IF (U=1)*(V<2)+(U>V)*(U<99)*(V>3) PRINT
   "YES"
30 LET R=RND(100), A=(R>3)+(R>15)+(R>56)+
   (R>98)
```

In statement 10, A will be set to 123 if X>Y, to 456 if X=Y, and to 789 if X<Y. In statement 20, the "*" operator acts like a logical AND, and the "+" operator acts like a logical OR. In statement 30, Y will be a random number between 0 and 4 with a prescribed probability distribution of: 3% of being 0, 15-3= 12% of being 1, 56-15=41% of being 2, 98-56=42% of being 3, and 100-98=2% of being 4.

**Direct Commands.** All the commands described later can be used as direct commands except the following three; they can only be used as direct commands and not as part of a statement:

```
RUN
```

will start to execute the program starting at the lowest statement number.

```
LIST
```

will print out all the statements in numerical order.

```
LIST 120
```

will print out all the statements in numerical order starting at statement 120.

```
NEW
```

will delete all statements.

**Abbreviation and blanks.** You may use blanks freely, except that numbers, command keywords, and function names can not have embedded blanks.

You may truncate all command keywords and function names and follow them by a period. "P.", "PR.", "PRI.", and "PRIN." all stand for "PRINT". Also the word LET in LET command can be omitted. The "shortest" abbreviation for all keywords are as follows:

| | | | |
|---|---|---|---|
| A.=ABS | F.=FOR | GOS.=GOSUB | G.=GOTO |
| IF=IF | IN.=INPUT | L.=LIST | N.=NEW |
| N.=NEXT | P.=PRINT | REM=REMARK | R.=RETURN |
| R.=RND | R.=RUN | S.=SIZE | S.=STEP |
| S.=STOP | TO=TO | | |
| Implied=LET | | | |

**Statements.** A statement consists of a statement number of betweeen 1 and 32767 followed by one or more commands. Commands in the same statement are separated by a semi-colon ";". "GOTO", "STOP", and "RETURN" commands must be the last command in any given statement.

**Commands.** Tiny BASIC commands are listed below with examples. Remember that commands can be concatenated with semi-colons. In order to store the statement, you must also have a statement number in front of the commands. The statement number and the concatenation are not shown in the examples.

*REM or REMARK Command*

```
REM anything goes
```

This line will be ignored by TBI.

*LET Command*

```
LET A=234-5*6, A=A/2, X=A-100, @(X+9)=A-1
```

will set the variable A to the value of the expression 234-5*6 (i.e., 204), set the variable A (again) to the value of the expression A/2 (i.e., 102), set the variable X to the value of the expression A-100 (i.e., 2), and then set the variable @(11) to 101 (where 11 is the value of the expression X+9 and 101 is the value of the expression A-1).

```
LET U=A#B, V=(A>B)*X+(A<B)*Y
```

will set the variable U to either 1 or 0 depending on whether A is not equal to or is equal to B; and set the variable V to either X, Y or 0 depending on whether A is greater than, less than, or equal to B.

*PRINT Command*

PRINT

will cause a carriage-return (CR) and a line-feed (LF) on the output device.

PRINT A*3+1, "ABC 123 !@#", ' CBA '

will print the value of the expression A*3+1 (i.e., 307), the string of characters "ABC 123 !@#", and the string " CBA ", and then a CR-LF. Note that either single or double quotes can be used to quote strings, but pairs must be matched.

PRINT A*3+1, "ABC 123 !@#", ' CBA ',

will produce the same output as before, except that there is no CR-LF after the last item is printed. This enables the program to continue printing on the same line with another "PRINT".

PRINT A, B, #3, C, D, E, #10, F, G

will print the values of A and B in six spaces, the values of C, D, an E in 3 spaces, and the values of F and G in 10 spaces. If there are not enough spaces specified for a given value to be printed, the value will be printed with enough spaces anyway.

PRINT 'ABC',-,'XXX'

will print the string "ABC", a CR without a LF, and then the string "XXX" (over ABC) followed by a CR-LF.

*INPUT Command*

INPUT A, B

When this command is executed, Tiny BASIC will print "A:" and wait to read in an expression from the input device. The variable A will be set to the value of this expression. Then "B:" is printed and variable B is set to the value of the next expression read from the input device. Note that not only numbers, but also expressions can be read as input.

INPUT 'WHAT IS THE WEIGHT'A, "AND SIZE"B

This is the same as the command above, except the prompt "A:" is replaced by "WHAT IS THE WEIGHT:" and the prompt "B:" is replaced by "AND SIZE:". Again, both single and double quotes can be used as long as they are matched.

INPUT A, 'STRING', -, "ANOTHER STRING", B

The strings and the "-" have the same effect as in "PRINT".

*IF Command*

IF A<B LET X=3; PRINT 'THIS STRING'

will test the value of the expression A<B. If it is not zero (i.e., if it is true), the commands in the rest of this statement will be executed. If the value of the expression is zero (i.e., if it is not true), the rest of this statement will be skipped over and execution continues at next statement. Note that the word "THEN" is not used.

*GOTO Command*

GOTO 120

will cause the execution to jump to statement 120. Note that GOTO command cannot be followed by a semi-colon and other commands. It must be ended with a CR.

GOTO A*10+B

will cause the execution to jump to a different statement number as computed from the value of the expression.

*GOSUB and RETURN Commands*

GOSUB command is similar to GOTO command except that: a) the current statement number and position within the statement is remembered; and b) a semi-colon and other commands can follow it in the same statement.

GOSUB 120

will cause the execution to jump to statement 120.

GOSUB A*10+B

will cause the execution to jump to different statements as computed from the value of the expression A*10+B.

RETURN

A RETURN command must be the last command in a statement and followed by a CR. When a RETURN command is encountered, it will cause the execution to jump back to the command following the most recent GOSUB command.

GOSUB can be nested. The depth of nesting is limited only by the stack space.

*FOR and NEXT Commands*

FOR X=A+1 TO 3*B STEP C-1

The variable X is set to the value of the expression A+1. The values of the expressions (not the expressions themselves) 3*B and C-1 are remembered. The name of the variable X, the statement number and the position of this command within the statement are also remembered. Execution then continues the normal way until a NEXT command is encountered.

The STEP can be positive, negative or even zero. The word STEP and the expression following it can be omitted if the desired STEP is +1.

NEXT X

The name of the variable (X) is checked with that of the most recent FOR command. If they do not agree, that FOR is terminated and the next recent FOR is checked, etc. When a match is found, this variable will be set to its current value plus the value of the STEP expression saved by the FOR command. The updated value is then compared with the value of the TO expression also saved by the FOR command. If this is within the limit, execution will jump back to the command following the FOR command. If this is outside the limit, execution continues following the NEXT command itself.

FOR can be nested. The depth of nesting is limited only by the stack space. If a new FOR command with the same control variable as that of an old FOR command is encountered, the old FOR will be terminated automatically.

*STOP Command*

This command stops the execution of the program and returns control to direct commands from the input device. It can appear many times in a program but must be the last command in any given statement; i.e., it cannot be followed by a semicolon and other commands.

**Stopping the Execution.** By the Control-C key on the input device.

**Control of Output Device.** The Control-O key on the input device can be used to turn the output device ON and OFF. This is useful when you want to read in a program punched on paper tape.

To produce such a paper tape, type "LIST" without CR. Turn on the paper tape punch and type a few Control-Shift-P's and then a CR. When listing is finished, type more Control-Shift-P's and turn off the punch.

To read back such a paper tape, type "NEW", CR, and Control-O, then turn on the paper tape reader. When the paper tape is read, turn the reader off and type a Control-O again.

**Error Report.** There are only three error conditions in Tiny

*(Continued on page 45)*

**LISTING ON PAGE 24**

# S. B. Tiny Basic for 6809's

```
                        * PALO ALTO TINY BASIC BY LI-CHEN WANG
                        *
                        *CONVERTED FOR  'THE MILL' OF STELLATION TWO
                        *  BY JAMES A. HINDS
                        *
                        ********************************
                        RATS    MACRO
                        RETURN  SET     *
                                RTS
                                ENDM
                        *
                        HERE    MACRO
                        TMP     SET     *
                                ORG     TABLE
                                IFC     &1,,1
                                FCC     &1
                                FDB     TMP+$8000
                                IFC     'GOTO',&1
                                FCC     "GO TO"
                                FDB     TMP+$8000
                                ENDIF
                        TABLE   SET     *
                                ORG     TMP
                                ENDM
                        *
                        DISPAT  MACRO
                                JSR     <EXEC
                                FDB     TABLE
                                ENDM
                        *
                        TSTC    MACRO
                                JSR     <IGNBLK
                                CMPA    #&1
                                BNE     &2
                                LEAY    1,Y
                                ENDM
                        * * * * * * * * * * * * * * * *
                        * VARIABLE AREA AT U
0000                            ORG     0               OFFSET FROM U
                FFCA    VARBGN  EQU     -27*2           VARIABLE SPACE
0000                    OCSW    RMB     1               OUTPUT ON/OFF FLAG
0001                    FLDCNT  RMB     1               PLACES FOR NUMERIC OUTPUT
0002                    CURRNT  RMB     2               POINTS TO CURR LINE
0004                    TXTUNF  RMB     2               -> TO LAST OF PRG
0006                    STKGOS  RMB     0               SAVE OF S IN GOSUB
                0006    VARNXT  EQU     *
0006                    STKINP  RMB     2               SAVE S DURING INPUT
0008                    LOPVAR  RMB     2               FOR LOOP INDEX
000A                    LOPINC  RMB     2               FOR LOOP STEP
000C                    LOPLMT  RMB     2               LIMIT
000E                    LOPLN   RMB     2               LOOP BEGINNING LINE
0010                    LOPPT   RMB     2               LOOP PROG RESTART
0012                    SEED    RMB     2
0014                    HIWAT   RMB     2               END OF @ ARRAY
0016                            RMB     2               EXTRA FOR BUFFER
0018                    BUFFER  RMB     72
                0060    BUFEND  EQU     *
                        ****************
                        * PAGE ZERO SUBROUTINES OF 8080
                        * ARE REPLACED ON 6809 BY DIRECT PAGE
                        * SUBROUTINES AND CAN BE ACCESSED
                        * BY 2 BYTE JSR
                        * DIRECT PAGE IS PAGE 1 ($100-$1FF)
                        ********************************
                        *  ENDCHK IS USED TO VERIFY
                        *  A COMMAND IS TERMINATED BY
                        *    A CR BEFORE PERFORMING
                        *    THE COMMAND.
0153                    ENDCHK  TSTC    $D,QWHAT
015B                            RATS
                        ********************************
                        *  FINE  USED TO CONTINUE EXECUTION
                        *     WITH NEXT STATEMENT. IF CURRENT
                        *     STATEMENT IS NOT AT END
                        *     A 'WHAT' ERROR IS SIGNALLED
015C 8D   DD            FINE    BSR     FIN             IF ERROR THEN FALL THROUGH
                        ********************************
015E 17   0465          QWHAT   LBSR    ERROR
0161 57 48 41 54                FCC     'WHAT?',13
0165 3F 0D
0167 17   045C          QSORRY  LBSR    ERROR
016A 53 4F 52 52                FCC     'SORRY',13
016E 59 0D
                        ********************************
                        *   IGNBLK    SCANS PAST ANY
                        *      SPACES AND RETURNS FIRST
                        *      CHAR WITH PROG PTR UPDATED
                        *
0170 31   21            I1      LEAY    1,Y
0172 A6   A4            IGNBLK  LDA     0,Y
0174 81   20                    CMPA    #'              A BLANK
0176 27   F8                    BEQ     I1              GET NEXT
0178                            RATS
                        ********************************
                        *  OUTPUT ROUTINES
                        *      SPACE     PUTS ONE BLANK
                        *      CRLF      PUTS CR
                        *      OUTC      PUTS CHAR IN MAILBOX
                        *        FOR 6502 TO GRAB AND
                        *        CLEAR MAILBOX
0179 86   20            SPACE   LDA     #'              A SPACE
017B 20   02                    BRA     OUTC
017D 86   0D            CRLF    LDA     #13
017F 1E   88            OUTC    EXG     A,A             GIVE 6502 SOME EXTRA CYCLES
0181 7D   FBFF                  TST     $FBFF


0184 26   F9                    BNE     OUTC
0186 B7   FBFF                  STA     $FBFF
0189                            RATS
                        ********************************
                        *  INPUT ROUTINE
                        *     CHKIO  RETURN ZERO IF NO
                        *       CHAR, OTHERWISE GRAB IT
                        *       RESTART BASIC IF IT IS
                        *       CONTROL-C
018A B6   FBFE          CHKIO   LDA     $FBFE           CHAR LOC
018D 27   FA                    BEQ     RETURN          IF NO CHAR JUST SAY SO
018F 7F   FBFE                  CLR     $FBFE           OTHERWISE CLEAR IT
0192 81   03                    CMPA    #$03            IS IT CNTL C
0194 26   F3                    BNE     RETURN          NO RETURN CHAR
                        ********************************
                        *   START  RELAY TO FIRST OF BASIC
                        *
0196 16   04A8          START   LBRA    ST1
```

```
                        *  PAGE 0 CONTAINS TABLE OF COMMAND NAMES
                        * AND THEIR CODE LOCATION EQUIV LOCATIONS
                        * ENTRIES INTO THIS TABLE IS BY
                        * MACRO "HERE"
0000                            ORG     0
0000 16   0193                  LBRA    START
             0003       TABLE   SET     *
             0000       TMP     SET     0
0100                            ORG     $100        ;LEAVE LOTS OF SPACE FOR TABLE
             0001               SETDP   *>>8
                        ******************************
                        * TSTNUM  RETURNS VALUE AND
                        *    LENGTH OF ASCII CODED INTEGER
                        *    IN TEXT. A NON-NUMBER WILL HAVE
                        *    A LENGTH OF ZERO.
0100 5F                 TSTNUM  CLRB
0101 4F                         CLRA
0102 1F   01                    TFR     D,X         CLEAR X AND ACCUM
0104 34   16                    PSHS    D,X         TOS= NUM AND NOS=#DIGITS
0106 9D   72            TN1     JSR     <IGNBLK
0108 80   30                    SUBA    #'0
010A 25   28                    BLO     NUMOUT      CHAR TOO LOW
010C 81   09                    CMPA    #9
010E 22   24                    BHI     NUMOUT      HIT CHAR THAT ISNT DIGIT
0110 31   21                    LEAY    1,Y         BUMP TO NEXT CHAR
0112 34   02                    PSHS    A           SAVE CHAR
0114 6C   64                    INC     4,S         INCREMENT DIGIT COUNT
0116 EC   61                    LDD     1,S         GET ACCUMULATOR
0118 81   0F                    CMPA    #$0F        CHECK IF ROOM
011A 22   10                    BHI     QHOW        TOO BIG TO MULT BY 10
011C 58                         ASLB
011D 49                         ROLA                DOUBLE
011E 58                         ASLB
011F 49                         ROLA                TIMES 4
0120 E3   61                    ADDD    1,S         TIMES 5
0122 58                         ASLB
0123 49                         ROLA                TIMES TEN
0124 EB   E0                    ADDB    ,S+         ADD IN DIGIT
0126 89   00                    ADCA    #0
0128 ED   E4                    STD     0,S         STORE ACCUM BACK
012A 2A   DA                    BPL     TN1         NO ERR? THEN GET NEXT DIGIT
012C 17   0497          QHOW    LBSR    ERROR
012F 48 4F 57 3F                FCC     'HOW?',13
0133 0D
0134 6D   63            NUMOUT  TST     3,S         SET FLAGS ON # DIGITS
0136 35   96                    PULS    D,X,PC      AND GO AWAY
0138 29   F2            TVHOW   BVS     QHOW        DIRECT OVERFLOW TEST FOR ERROR
013A                            RTS

                        ******************************
                        *  FIN IS CALLED TO CONTINUE
                        *     EXECUTION OF NEXT STMT
                        *     ON SAME OR NEXT LINE.
                        *     HOWEVER, IF CURRENT STMT
                        *     HAS MORE TO PROCESS CONTROL
                        *     GOES BACK TO CALLER
013B                    FIN     TSTC    ';,FI1
0143 32   62                    LEAS    2,S         VOID RETURN
0145 16   01EA                  LBRA    RUNSML
0148 81   0D            FI1     CMPA    #$D
014A 26   EE                    BNE     RETURN
014C 31   21                    LEAY    1,Y
014E 32   62                    LEAS    2,S
0150 16   01D1                  LBRA    RUNNXL
```

*(continued on the top of page 24)*

```
                        ******************************
                        * TSTVE  RETURN ADDRESS AND
                        *      VALUE OF VARIABLE
0199 8D   D7            TSTVE   BSR     IGNBLK
019B 80   40                    SUBA    #'@
019D 25   EA                    BCS     RETURN
019F 27   0F                    BEQ     RLYAT       IS STRING SPACE
01A1 81   1B                    CMPA    #27         IS IN ALPHABET??
01A3 2C   0E                    BGE     TVN
01A5 31   21                    LEAY    1,Y
01A7 48                         ASLA
01A8 40                         NEGA
01A9 30   C6                    LEAX    A,U
01AB EC   84                    LDD     ,X
01AD 1C   FE                    CLC
01AF                            RATS    RETURN      WITH CARRY CLEAR
01B0 16   011C          RLYAT   LBRA    ATSIGN
01B3 1A   01            TVN     SEC                 SET CARRY
01B5                            RATS
                        ******************************
                        *   EXEC AND TABLE DISPATCH
                        *
                        *      SCAN TABLE FOR KEYWORD
                        *       JUMP TO ASSOCIATED CODE
01B6 9D   72            EXEC    JSR     <IGNBLK
01B8 AE   F4                    LDX     [0,S]       GET TABLE ADDRESS FROM CALLER
01BA 10AF E4                    STY     0,S
01BD A6   A0            LCT     LDA     0,Y+
01BF A1   80                    CMPA    0,X+        MATCH PRG AGAINST TABLE
01C1 27   FA                    BEQ     LCT
01C3 A6   1F                    LDA     -1,X        LOOK AT MISMATCH
01C5 2B   17                    BMI     EXECF       FOUND!
01C7 A6   3F                    LDA     -1,Y        IS IT AN ABBREV
01C9 81   2E                    CMPA    #'.
01CB 27   0B                    BEQ     EXECP       YES! FORCE MATCH
01CD 10AE E4                    LDY     0,S         RESTORE PROG PTR
01D0 A6   80            EXEC1   LDA     0,X+
01D2 2A   FC                    BPL     EXEC1       SKIP REST OF KEYWORD IN TABLE
01D4 A6   80                    LDA     0,X+
01D6 20   E5                    BRA     LCT         RETRY
01D8 A6   80            EXECP   LDA     ,X+
01DA 2A   FC                    BPL     EXECP
01DC 31   21                    LEAY    1,Y
01DE 31   3F            EXECF   LEAY    -1,Y        BACK UP PRG PTR
01E0 E6   84                    LDB     0,X         NEXT TRANSFER BYTE
01E2 84   7F                    ANDA    #$7F
01E4 30   8D FE18               LEAX    0,PCR
01E8 30   8B                    LEAX    D,X
01EA AF   E4                    STX     0,S
01EC 39                         RTS
                        ******************************
                        *
                        * EXPRESSION ANALYZER
01ED 8D   38            EXPR    BSR     EXPR2
01EF 34   10                    PSHS    X           SAVE LEFT HAND VALUE
01F1                            DISPAT
01F5                            HERE    '>='
01F5 8D   26                    BSR     XP18
01F7 2E   1F                    BGT     ZER         M>=R
01F9 20   1C                    BRA     ONE
                        *
01FB                            HERE    '='
01FB 8D   20                    BSR     XP18        M=R
01FD 27   18                    BEQ     ONE
```

*(continued on the next page)*

# S. B. Tiny Basic for 6809's (Continued)

```
01FF 20   17          BRA    ZER
                      *
0201                  HERE   '>'
0201 8D   1A          BSR    XP18      M>R
0203 2D   12          BLT    ONE
0205 20   11          BRA    ZER
                      *
0207                  HERE   '<='
0207 8D   14          BSR    XP18      M<=R
0209 2C   0C          BGE    ONE
020B 20   0B          BRA    ZER
                      *
020D                  HERE   '='
020D 8D   0E          BSR    XP18
020F 27   06          BEQ    ONE
0211 20   05          BRA    ZER
                      *
0213                  HERE   '<'
0213 8D   08          BSR    XP18      M<R
0215 23   01          BLS    ZER
0217 5C               INCB
0218 ED   E4          STD    ,S        TO SET FLAGS
021A                  HERE   ,
021A EC   E1          LDD    ,S++      POP AND SET FLAGS
021C                  RATS   NO        LOGICAL OP
                      *
021D 8D   08          BSR    EXPR2
021F AC   62          CMPX   2,S       COMPARE WITH LEFT
0221 34   01          PSHS   CC
0223 4F               CLRA
0224 5F               CLRB             SET D TO ZERO (FALSE)
0225 35   81          PULS   CC,PC
                      ********************************
                      *  EXPR2 LOOKS AFTER + AND -
                      *        BINARY OPERATORS
0227 8D   26          BSR    EXPR3     GO AFTER LEFT HAND OPERAND
0229 34   06          PSHS   D
022B 9D   72          JSR    <IGNBLK
022D 81   2B          CMPA   #'+
022F 26   0A          BNE    XP290
0231 31   21          LEAY   +1,Y
0233 8D   1A          BSR    EXPR3
0235 E3   E1          ADDD   ,S++      ADD IN LEFT
0237 28   F0          BVC    EXPR21    AND AGAIN
0239 0E   2C          JMP    QHOW      OVERFLOW
023B 81   2D          CMPA   #'-
023D 26   0E          BNE    XP291
023F 31   21          LEAY   1,Y
0241 8D   0C          BSR    EXPR3
0243 34   06          PSHS   D
0245 EC   62          LDD    2,S
0247 A3   E4          SUBD   0,S
0249 32   64          LEAS   4,S       POP GARBAGE
024B 20   EA          BRA    XP235     CHECK FOR OVERFLOW
                      *
024D 35   90          PULS   X,PC
                      ********************************
                      * EXPR3  HANDLES THINGS WITH
                      *         / AND * IN THEM
                      *
024F 8D   1E          BSR    EXPR4
0251 34   06          PSHS   D
```

```
02AD 0E   38          JMP    <TVHOW
                      ********************************
                      * CHECK FOR NUMBER, VARIABLE OR
                      *  SUBEXPRESSION
                      *
02AF                  HERE   ,
02AF 9D   99          JSR    TSTVE     NOT A FUNCTION MUST BE VAR
02B1 24   F3          BCC    RETURN
02B3 9D   00          JSR    <TSTNUM   NOT VAR TRY NUM
02B5 26   EF          BNE    RETURN    DIGIT CNT<>0 MEANS NUMBER
02B7                  TSTC   '(,XP43
02BF 9D   ED          JSR    <EXPR
02C1 34   07          PSHS   CC,D      SAVE CONDITIONS OF EXPRESSION
02C3                  TSTC   '),XP43
02CB 35   87          PULS   CC,D,PC   FETCH CONDITIONS AND RETURN
02CD 0E   5E          JMP    QWHAT
                      ****************************
                      * GET INDEX INTO STRING SPACE
                      * IF WE GO PASSED THE CURRENT
                      * HIGH WATER MARK WE BUMP THAT
                      * TO KEEP STACK FROM GOING
                      * BANANAS
                      *
02CF 31   21          LEAY   1,Y
02D1 8D   E4          BSR    PARM      GET THE INDEX
02D3 102B FE55        LBMI   QHOW
02D7 AE   44          LDX    TXTUNF,U  GET END OF PROG
02D9 30   02          LEAX   2,X
02DB 30   8B          LEAX   D,X
02DD 30   8B          LEAX   D,X
02DF AC   C8 14       CMPX   HIWAT,U
02E2 23   03          BLS    HIWOK
02E4 AF   C8 14       STX    HIWAT,U
02E7 8D   B8          BSR    SIZE
02E9 102B FE7A        LBMI   QSORRY    NO ROOM
02ED 1C   FE          CLC
02EF EC   84          LDD    0,X       GET VALUE AT INDEX
02F1                  RATS
                      ****************************
                      * DIRECT STATEMENT EXECUTE
02F2 31   22          LEAY   2,Y       STEP PAST LINE NUMBER
02F4                  DISPAT AND       GO TO IT!
                      ****************************
                      * LIST PROGRAM POSSIBLY FROM
                      *      A GIVEN LINE NUMBER, OTHERWISE
                      *      THE WHOLE SHEBANG
                      *
02F8                  HERE   'LIST'
02F8 9D   00          JSR    <TSTNUM   TEST IF A LN #
02FA 34   06          PSHS   D
02FC 9D   53          JSR    <ENDCHK   IF NO # THEN USE 0
02FE 35   06          PULS   D
0300 17   01D5        LBSR   FNDLN     FIND IT
0303 1022 FE8F        LBHI   START     HIT END?
0307 17   0278        LBSR   PRTLN     PRINT THE LINE
030A 9D   8A          JSR    <CHKIO    USER INTERRUPT?
030C 17   01D2        LBSR   FNDLNP    FIND NEXT LINE
030F 20   F2          BRA    LS1
                      *
0311                  HERE   'NEW'
0311 9D   53          JSR    <ENDCHK   ABORT IF NOT ONLY THING IN STMT
0313 30   8D 045F     LEAX   TXTBGN-1,PCR
```

```
0253                EXP3EL  DISPAT
0257                        HERE    '*'
0257 8D   16                BSR     EXPR4
0259 17   046F              LBSR    MULT
025C 27   F5                BEQ     EXP3EL      SIGNAL ERROR IF D REG HAS SIGNIFI
ANT BITS
025E 0E   2C        EXP3ER  JMP     <QHOW
0260                        HERE    '/'
0260 8D   0D                BSR     EXPR4
0262 ED   7E                STD     -2,S        SET FLAGS
0264 27   F8                BEQ     EXP3ER
0266 17   04C4              LBSR    DIV
0269 ED   E4                STD     0,S         REPLACE REM BY QOU
026B 20   E6                BRA     EXP3EL
026D                        HERE    ,
026D 35   86                PULS    D,PC        GET RESULT AND RETURN
                    ********************************
                    * EXPR4 HANDLES FUNCTIONS AND
                    *     UNARY MINUS AND REPRESENTS
                    *     THE HIGHEST PRIORITY
                    *     IF THERE ARE () THEN EXPR
                    *     IS CALLED (OH! NO!) RECURSIVLY
                    *
026F                EXPR4   DISPAT ON           FUNCTIONS
0273                        HERE    'RND'
0273 EC   C8 12             LDD     SEED,U
0276 34   06                PSHS    D
0278 CC   0023              LDD     #%100011
027B 17   044D              LBSR    MULT
027E 35   06                PULS    D
0280 C3   0123              ADDD    #$0123
0283 84   7F                ANDA    #$7F        MASK SIGN
0285 ED   C8 12             STD     SEED,U
0288 34   06                PSHS    D           SAVE FOR DIVISION
028A 8D   2B                BSR     PARM
028C 102F FE9C              LBLE    QHOW        IF ZERO OR NEG
0290 17   049A              LBSR    DIV
0293 35   86                PULS    D,PC        GET REMAINDER AND RETURN
                    ********************************
                    *  ABSOLUTE VALUE
                    *
0295                        HERE    'ABS'
0295 8D   20                BSR     PARM        GET THE VALUE
0297 2A   83                BPL     RETURN
0299 34   06                PSHS    D
029B 4F                     CLRA
029C 5F                     CLRB
029D A3   E1                SUBD    S++
029F 0E   38                JMP     TVHOW       IMPLIED RETURN IF NO OVERFLOW
                    ********************************
                    *  SIZE  HOW MUCH STORAGE LEFT
                    *    BETWEEN STACK AND END OF
                    *    PROGRAM
02A1                        HERE    'SIZE'
02A1 1F   40        SIZE    TFR     S,D
02A3 A3   C8 14             SUBD    HIWAT,U     SUBTRACT END OF STRING ADDR
02A6                        RATS
                    ********************************
                    * UNARY MINUS HANDLED AS FNCT
02A7                        HERE    '-'         UNARY MINUS
02A7 9D   ED                JSR     <EXPR
02A9 43                     COMA
02AA 50                     NEGB
02AB 82   00                SBCA    #0
```

*(continued on the top of page 26)*

```
0317 AF   44                STX     TXTUNF,U    RESET BEGINNING OF PROGRAM PTR
0319 AF   C8 14             STX     HIWAT,U     AND HI WATER STRING
031C 0E   96                JMP     <START
                    ********************************
                    *  RUN--- FIRE THE ROCKETS AND
                    *       BLAST OFF
                    *    EACH STATEMENT IS RESPONSIBLE
                    *    FOR COMING BACK TO RUNNXL,
                    *    RUNNNL,RUNTSL AND RUNSML
                    *    AS PROGRAM POINTER (Y) NEEDS
                    *    CARE AND FEEDING
031E                        HERE    'RUN'
031E 9D   53                JSR     <ENDCHK
0320 31   8D 0453           LEAY    TXTBGN,PCR FIRST LINE
0324 4F             RUNNXL  CLRA
0325 5F                     CLRB                START AT ANY LINE >0
0326 17   01B8      RUNNNL  LBSR    FNDLNP
0329 1022 FE69              LBHI    START
032D 10AF 42        RUNTSL  STY     CURRNT,U
0330 31   22                LEAY    2,Y         BUMP PASSED LINE NUM
0332 9D   8A        RUNSML  JSR     <CHKIO      RUN SAME LINE
0334                        DISPAT START        DOING STATEMENT
                    ********************************
                    *   STOP WILL CAUSE GRACEFUL
                    *    TERMINATION
                    *
0338                        HERE    'STOP'
0338 9D   53                JSR     <ENDCHK
033A 0E   96                JMP     <START
                    ********************************
                    *   PRINT  PRINT OUT EXPRESSIONS,
                    *       STRINGS. WITH POSSIBLE
                    *       FORMAT FOR DIGITS
                    *
033C                        HERE    'PRINT'
033C 86   06                LDA     #6
033E A7   41                STA     FLDCNT,U
0340 9D   72                JSR     <IGNBLK
0342 81   3B                CMPA    #';
0344 26   06                BNE     PR2
0346 31   21                LEAY    1,Y
0348 9D   7D                JSR     <CRLF
034A 20   E6                BRA     RUNSML
034C 81   0D        PR2     CMPA    #$0D
034E 26   06                BNE     PR0
0350 31   21                LEAY    1,Y
0352 9D   7D                JSR     <CRLF
0354 20   CE                BRA     RUNNXL
0356 81   23        PR0     CMPA    #'#         IS IT A FORMAT LENGTH?
0358 26   08                BNE     PR1
035A 31   21                LEAY    1,Y
035C 9D   ED                JSR     <EXPR
035E E7   41                STB     FLDCNT,U
0360 20   0A                BRA     PR3
0362 17   0236      PR1     LBSR    QTSTG       PRINT POSSIBLE STRING
0365 27   05                BEQ     PR3
0367 9D   ED                JSR     <EXPR
0369 17   01D5              LBSR    PRTNUM
036C 9D   72        PR3     JSR     <IGNBLK
036E 81   2C                CMPA    #',
0370 26   06                BNE     PR6
0372 31   21                LEAY    1,Y
0374 9D   3B                JSR     <FIN        IF NOT ENDLINE THEN COME BACK
0376 20   DE                BRA     PR0
```

*(continued on the next page)*

# S. B. Tiny Basic for 6809's (Continued)

```
0378 9D   7D        PR6     JSR     <CRLF       END OF STATEMENT WITH
037A 0E   5C                JMP     <FINE       IMPLIED CR AND LF
                    *********************************
                    *   GOTO
                    *
037C                        HERE    'GOTO'
037C 9D   ED                JSR     <EXPR
037E 34   20                PSHS    Y           SAVE TXT POINTER FOR ERROR SUB
0380 17   0155              LBSR    FNDLN
0383 26   7C                BNE     GOERR       COULDN'T FIND DESTINATION
0385 32   62                LEAS    2,S         DISCARD OLD LINE NUM
0387 20   A4                BRA     RUNTSL
                    *
                    *********************************
                    *   IF
                    *
0389                        HERE    'IF'
0389 9D   ED                JSR     EXPR        EVALUATE THE EXPRESSION
038B 26   A5                BNE     RUNSML      IF <>0 THEN CONT
                    *    FALL THROUGH TO
                    *
                    *********************************
                    *  REM  IGNORE THIS WHOLE LINE
                    *
038D                        HERE    'REM'       LOOKS LIKE 'IF 0'
038D 10AE 42                LDY     CURRNT,U
0390 CC   0001              LDD     #1
0393 E3   A4                ADDD    0,Y         CURRENT LINE PLUS ONE
0395 20   8F                BRA     RUNNNL
                    *********************************
                    *  INPUT
                    *
0397 10EE 46        INPERR  LDS     STKINP,U    REGAIN
039A 35   36                PULS    D,X,Y       CURRNT,JUNK,TEXT PTR
039C ED   42                STD     CURRNT,U    AND RETRY THIS INPUT STMT
039E                        HERE    'INPUT'
039E 34   20        IP1     PSHS    Y           SAVE TEXT POINTER
03A0 17   01F8              LBSR    QTSTG       PRINT QUOTED PROMPT
03A3 26   06                BNE     IP2         NO TRY VARIABLE
03A5 9D   99                JSR     TSTVE
03A7 25   3D                BCS     IP4
03A9 20   17                BRA     IP3         YES INPUT VARIABLE
03AB 34   20        IP2     PSHS    Y           SAVE FOR PRTSTG
03AD 9D   99                JSR     TSTVE
03AF 1025 FDAB              LBCS    QWHAT
03B3 A6   A4                LDA     ,Y
03B5 6F   A4                CLR     ,Y          SET TO ZERO AS STOPPER
03B7 35   20                PULS    Y
03B9 34   02                PSHS    A
03BB 17   01CE              LBSR    PRTSTL      PRINT PROMPT
03BE 35   02                PULS    A
03C0 A7   A2                STA     ,-Y         REPLACE CHAR OF PROG
03C2 1F   10        IP3     TFR     X,D         REGAIN ADDRESS OF VAR
03C4 34   20                PSHS    Y
03C6 AE   42                LDX     CURRNT,U
03C8 34   10                PSHS    X
03CA 30   8C F5             LEAX    IP3,PCR     POINT CURRNT TO NG AS INPUT FLAG
03CD AF   42                STX     CURRNT,U
03CF 10EF 46                STS     STKINP,U    SAVE STACK POINTER
03D2 34   06                PSHS    D           THE DESTINATION ADDRESS
03D4 86   3A                LDA     #':         PROMPT
03D6 17   0125              LBSR    GETLN       GET THE INPUT FROM USER
```

```
044D EC   84        NEXT3   LDD     0,X         AND PUT VALUE INTO ACCUM
044F E3   4A                ADDD    LOPINC,U
0451 ED   84                STD     0,X
0453 6D   4A                TST     LOPINC,U    WHICH DIRECTION ARE WE GOING?
0455 2B   0F                BMI     NEXT4
0457 10A3 4C                CMPD    LOPLMT,U
045A 2E   0E                BGT     NEXT5       DONE
045C AE   4E        NEXT6   LDX     LOPLN,U
045E AF   42                STX     CURRNT,U
0460 10AE C8 10             LDY     LOPPT,U
0464 0E   5C                JMP     <FINE
0466 A3   4C        NEXT4   SUBD    LOPLMT,U
0468 2C   F2                BGE     NEXT6
046A 17   0183      NEXT5   LBSR    POPA
046D 0E   5C                JMP     <FINE
                    *********************************
                    * FOR
                    *
046F                        HERE    'FOR'
046F 20   05                BRA     FORE        JUMP AROUND DEFAULT STMT CODE
                    *          TO ALLOW TABLE ENTRY
                    *          FOR ASSIGNMENT TO BE
                    *          PRIOR TO KEYWORDS OF
                    *       FOR STATEMENT IN TABLE
                    *     THIS IS REQUIRED BY THE
                    *     MACRO HERE AND THE WAY IT
                    *     WORKS
0471                          HERE    ,           DEFAULT ASSIGNMENT
0471 9D   3B                  JSR     <FIN        CAN BE EMPTY
0473 16   FF7E                LBRA    LET         OTHERWISE TREAT AS LET
                    *********************************
                    * ACTUAL CODE OF FOR
                    *
0476 17   019E      FORE      LBSR    PUSHA       SAVE LOOP VARS
0479 8D   45                BSR     SETVAL
047B AF   48                STX     LOPVAR,U    SAVE ADDR OF LOOP VAR
047D                        DISPAT
0481                        HERE    'TO'        'FOR'I=1 'TO'
0481 9D   ED                JSR     <EXPR
0483 ED   4C                STD     LOPLMT,U    STOPPING VAL
0485 20   02                BRA     FOR2        JUMP AROUND DEFAULT
0487                        HERE    ,           DID NOT HAVE 'TO'
0487 0E   5E                JMP     <QWHAT      SO COMPLAIN
0489                FOR2    DISPAT
048D                        HERE    'STEP'
048D 9D   ED                JSR     <EXPR
048F 20   03                BRA     FOR3
0491                        HERE    ,           NO STEP VALUE
0491 CC   0001              LDD     #1          SO DEFAULT TO ONE
0494 ED   4A        FOR3    STD     LOPINC,U
0496 AE   42                LDX     CURRNT,U
0498 AF   4E                STX     LOPLN,U     LOOP LINE NUMBER
049A 10AF C8 10             STY     LOPPT,U     LOOP PROG POINTER
049E 31   76                LEAY    -10,S       START LOOKING AT OLD 'PUSHA' RECOF
DS
04A0 31   2A        FOR7    LEAY    10,Y
04A2 AE   A4                LDX     0,Y         GET LOOP VAR ADDRESS
04A4 27   14                BEQ     FOR8        EXHAUSTED ALL FORS
04A6 AC   48                CMPX    LOPVAR,U    IS OLD SAME AS THIS VAR?
04A8 26   F6                BNE     FOR7
04AA 30   2A                LEAX    +10,Y       FOUND MATCH MUST DELETE IT
04AC 10EF E3                STS     ,--S        SAVE STACK POINTER
04AF A6   A2        FOR7L   LDA     ,-Y
```

```
03D9 31   C8 18          LEAY   BUFFER,U
03DC 9D   ED             JSR    EXPR        AND EVALUATE IT AS AN EXPRESSION
03DE ED   F1             STD    [0,S++]     SAVE VALUE IN VAR
03E0 35   10             PULS   X
03E2 AF   42             STX    CURRNT,U
03E4 35   20             PULS   Y
03E6 35   06     IP4     PULS   D           PURGE JUNK
03E8                     TSTC   "',",IP5    IS NEXT A COMMA
03F0 20   AC             BRA    IP1         YES, DO ANOTHER VARIABLE
03F2 0E   5C     IP5     JMP    <FINE       FINISH
                 ********************************
                 * LET   THE ASSIGNMENT STATEMENT
                 *
03F4                     HERE   'LET'
03F4 17   00C9   LET     LBSR   SETVAL      DO THE ASSIGNMENT
03F7                     TSTC   "',",IP5    CHECK END LINE
03FF 20   F3             BRA    LET
                 ********************************
                 * GOSUB
                 *
0401 35   20     GOERR   PULS   Y           REGAIN CURRENT LINE
0403 0E   2C             JMP    <QHOW       AND PRINT DIAGNOSTIC
0405                     HERE   'GOSUB'
0405 17   020F           LBSR   PUSHA       SAVE FOR PARAMETERS
0408 9D   ED             JSR    <EXPR       GET VAL IN D
040A 34   20             PSHS   Y           SAVE TEXT POINTER
040C 17   00C9           LBSR   FNDLN
040F 26   F0             BNE    GOERR       NO LINE SAY 'HOW'
0411 EC   42             LDD    CURRNT,U
0413 34   06             PSHS   D
0415 EC   46             LDD    STKGOS,U
0417 34   06             PSHS   D
0419 4F                  CLRA
041A 5F                  CLRB
041B ED   48             STD    LOPVAR,U    SAVE NEW ONES
041D 10EF 46             STS    STKGOS,U
0420 16   FF0A           LBRA   RUNTSL      AND RUN DEST LINE
                 ********************************
                 *  RETURN   FROM GOSUB
                 *
0423                     HERE   'RETURN'
0423 9D   53             JSR    <ENDCHK     BAD IF NOT END OF LINE
0425 EC   46             LDD    STKGOS,U    RELOAD STACK POINTER
0427 1027 FD01           LBEQ   QHOW        TOO MANY RETURNS
042B 1F   04             TFR    D,S         RESTORE STACK POINTER
042D 35   36             PULS   D,X,Y
042F ED   46             STD    STKGOS,U
0431 AF   42             STX    CURRNT,U
0433 17   01BA           LBSR   POPA        RESTORE FOR--NEXT ENVIRONMENT
0436 0E   5C             JMP    <FINE
                 ********************************
                 * NEXT
                 *
0438                     HERE   'NEXT'
0438 AE   48             LDX    LOPVAR,U    IF NO FOR THEN
043A 27   0F             BEQ    NEXT2       COMPLAIN LOUDLY
043C 9D   99             JSR    <TSTVE
043E 24   02             BCC    NEXTO       IF NO VARIABLE SPECIFIED
0440 AE   48             LDX    LOPVAR,U    THEN UST USE LATEST FOR VARIABLE
0442 AC   48     NEXTO   CMPX   LOPVAR,U
0444 27   07             BEQ    NEXT3
0446 17   01A7           LBSR   POPA
0449 26   F7             BNE    NEXTO
044B 0E   2C     NEXT2   JMP    <QHOW
```

*(Continued on top of page 28)*

```
04B1 A7   82             STA    ,-X
04B3 10AC E4             CMPY   0,S
04B6 2E   F7             BGT    FOR7L
04B8 32   02             LEAS   2,X         CUT BACK STACK
04BA 10AE C8 10  FOR8    LDY    LOPPT,U     GET PROG POINTER
04BE 0E   5C             JMP    <FINE
                 ********************************
04C0 17   FCD6   SETVAL  LBSR   TSTVE       MUST BE VARIABLE
04C3 25   11             BCS    QW1         NO VARIABLE
04C5 34   10             PSHS   X
04C7                     TSTC   '=,QW1
04CF 9D   ED             JSR    <EXPR       EVALUATE THE EXPRESSION
04D1 35   10             PULS   X
04D3 ED   84             STD    0,X         LEAVE ADDRESS IN X (FOR USES IT)
04D5                     RATS
04D6 0E   5E     QW1     JMP    <QWHAT      RELAY TO QWHAT
                 ********************************
                 * UTILITIES AND EXTRAS
                 *
                 * FINDLINE LINE NUMBER IN D
04D8 4D          FNDLN   TSTA
04D9 102B FC4F           LBMI   QHOW        NEGATIVE LINE NUM
04DD 31   8D 0296        LEAY   TXTBGN,PCR  INITIAL LINE PTR
04E1 10AC 44     FNDLNP  CMPY   TXTUNF,U    PASSED END?
04E4 22   EF             BHI    RETURN
04E6 10A3 A4             CMPD   0,Y         LOOK AT LINE NUMBER
04E9 23   EA             BLS    RETURN      IF REG <= LINE NUM
                   04EB  FNDNXT EQU    *
04EB 10AC 44             CMPY   TXTUNF,U    DONT SCAN PAST END
04EE 22   E5             BHI    RETURN
04F0 31   22             LEAY   2,Y
04F2 34   02             PSHS   A           SAVE LINE NUMB HIGH BYTE
04F4 86   0D             LDA    #13
04F6 A1   A0     F1      CMPA   ,Y+
04F8 26   FC             BNE    F1          SCAN TO CR OF LINE
04FA 35   02             PULS   A           REGAIN  HIGH LINE NUM
04FC 20   E3             BRA    FNDLNP      TRY THIS LINE
                 ********************************
                 * GETLN      GET INPUT LINE AND
                 *            EDIT ACCORDING TO
                 *            BACKSPACE, CNTL X
                 *
04FE 9D   7F     GETLN   JSR    <OUTC       PROMPT
0500 31   C8 18          LEAY   BUFFER,U    START AT BEGINNING
0503 30   C8 60          LEAX   BUFEND,U
0506 34   30             PSHS   Y,X
0508 9D   8A     GL1     JSR    <CHKIO
050A 27   FC             BEQ    GL1         NOTHING YET
050C 9D   7F             JSR    <OUTC       ECHO
050E 81   0A             CMPA   #$0A        LINE FEED
0510 27   F6             BEQ    GL1         IGNORE IT
0512 81   08             CMPA   #$1F&'H     DELETE CODE
0514 27   0F             BEQ    GL3
0516 81   18             CMPA   #$1F&'X     CONTROL X
0518 27   1A             BEQ    GL4
051A A7   A0             STA    ,Y+
051C 81   0D             CMPA   #$D
051E 27   1C             BEQ    GLO         RETURN
0520 10AC E4             CMPY   S           WITH BUFFER END
0523 26   E3             BNE    GL1
0525 10AC 62     GL3     CMPY   2,S         ANYTHING IN BUFFER?
0528 27   0A             BEQ    GL4
052A 31   3F             LEAY   -1,Y
052C 9D   79             JSR    <SPACE
```

*(Continued on next page)*

# S. B. Tiny Basic for 6809's (Continued)

```
052E 86   08            LDA    ##1F&'H    CONTROL H
0530 9D   7F            JSR    <OUTC      AND BACK UP OVER SPACE
0532 20   D4            BRA    GL1
0534 9D   7D      GL4   JSR    <CRLF
0536 32   64            LEAS   4,S        POP OFF TEMPORARIES
0538 86   3F            LDA    #'?        QUESTION MARK PROMPT
053A 20   C2            BRA    GETLN
053C 32   64      GLO   LEAS   4,S
053E A7   22            STA    2,Y        THROW ANOTHER RETURN
0540                    RATS
                ********************************
                * PRTNUM AND FRIENDS
                *
0541 8E   000A    PRTNUM LDX     #10
0544 34   36            PSHS   D,Y,X      SAVE PRG POINTER AND STOPPER
0546 E6   41            LDB    FLDCNT,U
0548 5A                 DECB
0549 49                 ROLA               PUT SIGN OF QUANTITY INTO LOW B
054A 59                 ROLB
054B 1D                 SEX
054C 1F   02            TFR    D,Y         PUT INTO COUNTER
                ********
054E EC   E1            LDD    0,S++       SET FLAGS
0550 2A   07            BPL    PN2         COMPLEMENT IF NEGATIVE
0552 43                 COMA
0553 53                 COMB
0554 C3   0001          ADDD   #1
0557 31   3E      PN1   LEAY   -2,Y        DECREMENT PLACE COUNT
0559 34   06      PN2   PSHS   D            SAVE QUOTIENT

055B CC   000A          LDD    #10         DIVIDE BY 10 TO GET
055E 17   01CC          LBSR   DIV         NEXT DIGIT
0561 26   F4            BNE    PN1         HAVE NOT GOT ALL
0563 1F   20            TFR    Y,D         PUT PLACE COUNT INTO B
0565 20   02            BRA    PN3
0567 9D   79      PN4   JSR    <SPACE      PRINT SPACE
0569 C0   02      PN3   SUBB   #2          DEC SPACE COUNT
056B 2A   FA            BPL    PN4
056D 56                 RORB               PUT SIGN BIT INTO CARRY
056E 24   04            BCC    PN6         DONT PRINT MINUS
0570 86   2D            LDA    #'-         LEADING MINUS
0572 9D   7F      PN8   JSR    <OUTC
0574 35   06      PN6   PULS   D           GET REMAINDER
0576 C1   0A            CMPB   #10         DONE?
0578 27   06            BEQ    PN7
057A 1F   98            TFR    B,A
057C 8A   30            ORA    ##$30
057E 20   F2            BRA    PN8
0580 35   A0      PN7   PULS   Y,PC        AND RETURN
                ********************************
                * PRTLN   PRINT LINE OF BASIC
                *           PROGRAM
                *
0582 C6   04      PRTLN LDB    #4          FOUR SPACES
0584 E7   41            STB    FLDCNT,U
0586 EC   A1            LDD    ,Y++        GET LINE NUMBER
0588 8D   B7            BSR    PRTNUM      AND PRINT IT
058A 9D   79            JSR    <SPACE
058C 5F         PRTSTL  CLRB               FORCE NO MATCH
                *  AND FALL INTO
                ********************************
                * PRTSTG  PRINT STRING
                *
058D E1   A0      PRTSTG CMPB    ,Y+         PRINT STRING AT Y
```

```
0613 32   62            LEAS   2,S
0615 1F   05            TFR    D,PC
                *
0617 17   FC87    PUSHA LBSR   SIZE        ANU ROOM?
061A 83   000C          SUBD   #12
061D 1023 FB46          LBLS   QSORRY
0621 35   06            PULS   D           RETURN ADDR
0623 AE   48            LDX    LOPVAR,U    TEST THIS
0625 27   13            BEQ    PU1
0627 AE   C8 10         LDX    LOPPT,U
062A 34   10            PSHS   X
062C AE   4E            LDX    LOPLN,U
062E 34   10            PSHS   X
0630 AE   4C            LDX    LOPLMT,U
0632 34   10            PSHS   X
0634 AE   4A            LDX    LOPINC,U
0636 34   10            PSHS   X
0638 AE   48            LDX    LOPVAR,U
063A 34   10      PU1   PSHS   X
063C 1F   05            TFR    D,PC        AND RETURN
                ********************************
                * ST1 (ALIAS START)
                *    INITIALIZE AND ENTER COMMAND
                *    MODE
                *
063E 4F 4B 0D   OK      FCC    'OK',13
0641 30   8D FABB ST1   LEAX   TSTNUM,PCR
0645 1F   10            TFR    X,D
0647 1F   8B            TFR    A,DP
0649 33   8D 0512       LEAU   DATAS,PCR
064D 32   C8 CA         LEAS   VARBGN,U
0650 9D   7D            JSR    <CRLF
0652 31   8C E9         LEAY   OK,PCR      PRINT OK AND PROMPT
0655 17   FF34         LBSR   PRTSTL
0658 4F                 CLRA               B IS ALREADY ZERO
0659 ED   48            STD    LOPVAR,U    NO CURRENT FOR LOOPS
065B ED   46            STD    STKGOS,U    NOR GOSUB RETURNS
065D 86   3E      ST3   LDA    #'>         PROMPT TO ENTER A STATEMENT
065F 17   FE9C          LBSR   GETLN
0662 34   20            PSHS   Y           SAVE POINTER TO END OF TEXT
0664 31   C8 18         LEAY   BUFFER,U    POINT TO START
0667 9D   00            JSR    <TSTNUM     WILL RETURN ZERO IF NO NUM
0669 34   07            PSHS   CC,D
066B 9D   72            JSR    <IGNBLK
066D EC   61            LDD    1,S         REGAIN LINE NUM
066F ED   A3            STD    ,--Y        PUT LINE NUMBER IN FRONT
0671 10AF 42            STY    CURRNT,U    MAKE THE CURRENT LINE
0674 35   07            PULS   CC,D
0676 1027 FC78          LBEQ   DIRECT      DO IMMEDIATE MODE IF NO LEADING NU
MBER
                * Y POINTS TO BEGINNING OF LINE
                * AND TOP OF STACK POINTS TO END
067A 34   20            PSHS   Y
067C 17   FE59          LBSR   FNDLN
067F 34   20            PSHS   Y           TOS->TEXT AREA
0681 26   17            BNE    ST4         INSERT
0683 17   FE65          LBSR   FNDNXT
0686 AE   E4            LDX    0,S         BEGINNING OF LINE TO DELETE
0688 20   04            BRA    STL
068A A6   A0      STLE  LDA    ,Y+
068C A7   80            STA    ,X+
068E 10AC 44      STL   CMPY   TXTUNF,U    ARE WE DONE MOVING?
0691 23   F7            BLS    STLE
```

```
058F 27   AF              BEQ    RETURN     IF SAME AS STOPPER
0591 A6   3F              LDA    -1,Y       GET THAT CHAR
0593 9D   7F              JSR    <OUTC
0595 81   0D              CMPA   #13
0597 26   F4              BNE    PRTSTG     NOT A CR SO GO
0599 4D                   TSTA              IF RETURN  THEN NON ZERO
059A                      RTS
                    *
                    * PRINT QUOTED STRING
                    *  RETURN WITH NOT ZERO IF DIDNT RECOGNIZE AS OUR STUFF
059B                QTSTG  TSTC   '",QT3     DOUBLE QUOTE
05A3 C6   22              LDB    #'"
05A5 8D   E6        QT1    BSR    PRTSTG
05A7 1026 FD79            LBNE   RUNNXL     RUN NEXT LINE IF HIT RETURN
05AB                      RTS    RETURN     W ZERO SET
05AC                QT3    TSTC   $27,QT4    SINGLE QUOTE
05B4 C6   27              LDB    #$27       PRINT TILL MATCH
05B6 20   ED              BRA    QT1
05B8                QT4    TSTC   '!,RETURN
05C0 86   8D              LDA    #$8D       FUNNY RETURN
05C2 9D   7F              JSR    <OUTC      APPLE DOES NOT SUPPORT THIS
05C4 4F                   CLRA              ZET ZERO STATUS
05C5                      RTS
                    ********************************
                    *  ERROR REPORT
05C6 34   20        ERROR  PSHS   Y          SAVE LINE NUMBER
05C8 10AE 62              LDY    2,S        GET THE RETURN ADDR
05CB 8D   BF              BSR    PRTSTL     PRINT HOW OR WHAT
05CD 35   20              PULS   Y
05CF A6   A4              LDA    0,Y
05D1 34   02              PSHS   A          SAVE THE CHAR AT ERROR POINT
05D3 6F   A4              CLR    0,Y
05D5 10AE 42              LDY    CURRNT,U   IS THIS IN
05D8 EC   A4              LDD    0,Y        IMMEDIATE MODE?
05DA 27   12              BEQ    ERR0
05DC 102B FDB7            LBMI   INPERR     INPUT COMMAND??
05E0 8D   A0              BSR    PRTLN      PRINT UP TO ZERO
05E2 86   3F              LDA    #'?
05E4 9D   7F              JSR    <OUTC      PRINT QUESTION
05E6 35   02              PULS   A
05E8 A7   A2              STA    0,-Y       RESTORE LINE
05EA 32   62              LEAS   2,S        POP PHONY RETURN
05EC 8D   9E              BSR    PRTSTL
05EE 0E   96        ERR0   JMP    <START
                    ********************************
                    *   POPA AND PUSHA
                    *
                    *
05F0 EC   62        POPA   LDD    2,S
05F2 27   1B              BEQ    POPA0
05F4 ED   48              STD    LOPVAR,U
05F6 EC   64              LDD    4,S
05F8 ED   4A              STD    LOPINC,U
05FA EC   66              LDD    6,S
05FC ED   4C              STD    LOPLMT,U
05FE EC   68              LDD    8,S
0600 ED   4E              STD    LOPLN,U
0602 EC   6A              LDD    10,S
0604 ED   C8 10           STD    LOPPT,U
0607 35   06              PULS   D
0609 32   6A              LEAS   10,S
060B 1C   FB              ANDCC  #$FB       TURN OFF ZERO
060D 1F   05              TFR    D,PC
060F ED   48        POPA0  STD    LOPVAR,U
0611 35   06              PULS   D
```

*(Continued on top of page 30)*

```
0693 30   1F              LEAX   -1,X
0695 AF   44              STX    TXTUNF,U   UPDATE END POINTER
0697 AF   C8 14           STX    HIWAT,U    AND STRING SPACE
069A EC   64        ST4    LDD    4,S        POINT TO END OF LINE
069C A3   62              SUBD   2,S        FIND IF LENGTH  OF LINE IS EMPTY
069E 1083 0003            CMPD   #3
06A2 27   9D              BEQ    ST1        YES, DO NOT INSERT
                    *MAKE ROOM FOR NEW LINE
06A4 10AE 44              LDY    TXTUNF,U
06A7 30   AB              LEAX   D,Y        ADD IN LENGTH
06A9 AF   44              STX    TXTUNF,U   UPDATE END POINTER
06AB AF   C8 14           STX    HIWAT,U
06AE 86   0D              LDA    #$0D
06B0 A7   84              STA    0,X
06B2 A6   A2        ST2LE  LDA    ,-Y
06B4 A7   82              STA    ,-X
06B6 10AC E4              CMPY   0,S        HAVE WE REACHED BEGINNING?
06B9 22   F7              BHI    ST2LE
                    * START MOVING LINE
06BB 35   20              PULS   Y
                    *X POINTS TO END OF LINE AREA
                    *Y POINTS TO BEGINNING OF LINE
                    *TOS POINTS TO  BEGINNING OF TEXT
                    *NOS POINTS TO END OF BUFFER
06BD 35   10              PULS   X
06BF A6   80        ST3LE  LDA    ,X+
06C1 A7   A0              STA    ,Y+
06C3 AC   E4              CMPX   ,S
06C5 25   F8              BLO    ST3LE
06C7 35   10              PULS   X          DISCARD GARBAGE
06C9 20   92              BRA    ST3
                    *
                    ********************************
                    * MULTIPLY AND DIVIDE ROUTINES
                    * CALLING SEQUENCES:
                    * MULT  LDD #MULTIPLIER
                    *       PSHS D
                    *       LDD #MULTIPLICAND
                    *       LBSR MULT
                    * ON EXIT D HAS HIGH ORDER BITS
                    * TOS HAS LOW ORDER BITS
                    * EQAL ZERO IT TRUE IFF THE
                    * CONTEDTS OF D ARE THE SAME
                    * AS THE HIGH BIT OF TOS
                    *   NEG IS SET  ON THE SIGN
                    * OF  TOS
06CB 32   7D        MULT   LEAS   -3,S
06CD 8D   3D              BSR    UNSIGN     RETURNS W/D=0
06CF 8E   0010            LDX    #16
06D2 66   65              ROR    5,S
06D4 66   66              ROR    6,S
06D6 24   02        M1     BCC    M2
06D8 E3   61              ADDD   1,S
06DA 46             M2     RORA
06DB 56                   RORB
06DC 66   65              ROR    5,S
06DE 66   66              ROR    6,S
                    * RESULT IS NOW IN D CAT 5,S
06E0 30   82              LEAX   ,-X
06E2 26   F2              BNE    M1
06E4 6D   E4        SIGN   TST    0,S        SIGN LOC
06E6 2A   10              BPL    SETFLG
06E8 43                   COMA
06E9 53                   COMB
```

*(Continued on next page)*

# S. B. Tiny Basic for 6809's (Continued)

```
06EA 63   65              COM    5,S
06EC 60   66              NEG    6,S
06EE 26   08              BNE    SETFLG
06F0 6C   65              INC    5,S
06F2 26   04              BNE    SETFLG
06F4 5C                   INCB
06F5 26   01              BNE    SETFLG
06F7 4C                   INCA
06F8 6D   65    SETFLG    TST    5,S
06FA 2B   07              BMI    S2
06FC ED   7E              STD    -2,S       SET FLAGS
06FE 1C   F7              ANDCC  #$F7       SET POSITIVE
0700 32   63              LEAS   3,S        CUT GARBAGE
0702 39                   RTS
0703 1083 FFFF  S2        CMPD   #$FFFF     SET FLAG
0707 1A   08              ORCC   #8         SET NEG
0709 32   63              LEAS   3,S        CUT GARBAGE
070B 39                   RTS
070C ED   63    UNSIGN    STD    3,S
070E A7   62              STA    2,S        SIGN POSITION
0710 67   62              ASR    2,S        DOUBLE THE SIGN
0712 2A   06              BPL    US2
0714 4F                   CLRA
0715 5F                   CLRB
0716 A3   63              SUBD   3,S
0718 ED   63              STD    3,S
071A A6   67    US2       LDA    7,S
071C 2A   0C              BPL    USR
071E A6   62              LDA    2,S
0720 88   80              EORA   #$80
0722 A7   62              STA    2,S
0724 4F                   CLRA
0725 5F                   CLRB
0726 A3   67              SUBD   7,S
0728 ED   67              STD    7,S
072A 4F         USR       CLRA
072B 5F                   CLRB
072C 39                   RTS
                ********************************
                * DIVIDE ROUTINE
                *  ENTER WITH DIVISOR IN D
                * DIVIDEND ON STACK TOP
                * JSR DIV
                * REMAINDER ON TOP OF STACK
                * QUOTIENT IN D
                * FLAGS SET AS TO D
                *
072D 32   7D    DIV       LEAS   -3,S       MAKE ROOM
072F 8D   DB              BSR    UNSIGN
0731 8E   0011            LDX    #17
0734 20   02              BRA    DIV1
0736 59         DIV5      ROLB
0737 49                   ROLA
0738 A3   61    DIV1      SUBD   1,S
073A 2B   12              BMI    DIV3
073C 1A   01              SEC
073E 69   66    DIV2      ROL    6,S
0740 69   65              ROL    5,S
0742 30   1F              LEAX   -1,X
0744 26   F0              BNE    DIV5
0746 20   10              BRA    DIV4       DONE
0748 59         DIVR      ROLB
0749 49                   ROLA
074A E3   61              ADDD   1,S
```



# Dr. Dobb's Clinic

```
;*****************************************************************
;
;    Name:  Tom Martin
;
;    Date:  January 12, 1981
;
;    8085 Instruction Length Calculator.
;
;*****************************************************************


bytesz  EQU     8               ;Number of bits per byte
rnglwr  EQU     X'40'           ;Lower bound of 1 byte instruction range
rngupr  EQU     X'BF'           ;Upper bound of 1 byte instruction range
;
;    Calculate the length of an 8085 instruction from its opcode.  (A,H,L,PSW)
;
;    Inputs:
;
;       A   - The opcode.
;
;    Outputs:
;
;       A   - The length in bytes of the corresponding instruction.  A value of
;             0 signifies that the hexidecimal value passed as input does not
;             represent a valid opcode in the 8085 instruction set.
;


INLEN   EQU     *
        CPI     rnglwr          ;If not in the range of 1 byte instructions
        JC      INLEN1
        CPI     rngupr+1
        JC      INLEN2
        SUI     rngupr-rnglwr+1
INLEN1  EQU     *
        LXI     H,INLNTB        ;   Get instruction length from table
        CALL    GETBTP
        RET
INLEN2  EQU     *               ;Else
        MVI     A,1             ;   Instruction length is one
        RET


;
;    Get specified bit pair from vector of bit pairs.  (A,C,H,L,PSW)
;
;    Inputs:
;
;       A   - The index of the bit pair to get.
;       H,L - The base address of the vector of bit pairs.
;
;    Outputs:
;
;       A   - The specified bit pair in the least significant two bits with all
;             other bits reset.
;
GETBTP  EQU     *               ;Loop
        INX     H               ;   Increment table address
        SUI     bytesz/2
        JNC     GETBTP          ;Until byte containing bit pair is addressed
        DCX     H
        ADI     (bytesz/2)+1
        MOV     C,A
```

```
074C 2A F0              BPL     DIV2
074E 68 66      DIV3    ASL     6,S
0750 69 65              ROL     5,S
0752 30 1F              LEAX    -1,X
0754 26 F2              BNE     DIVR
0756 E3 61              ADDD    1,S
          0758  DIV4    EQU     *
                *RAW REMAINDER IN D, QUO IN 5,S
0758 ED 61              STD     1,S         SAVE IT FOR A WHILE
075A 68 E4              ASL     0,S         SHOULD WE COMPLEMENT QUO?
075C 24 06              BCC     NCQ         NO
075E 4F                 CLRA
075F 5F                 CLRB
0760 A3 65              SUBD    5,S
0762 ED 65              STD     5,S
0764 68 E0      NCQ     ASL     0,S+        SHOULD WE COMPLEMENT REMAINDER?
0766 24 06              BCC     NCR         NO
0768 4F                 CLRA
0769 5F                 CLRB
076A A3 E4              SUBD    0,S
076C ED E4              STD     0,S
076E AE E4      NCR     LDX     0,S         GET REMAINDER
0770 EC 64              LDD     4,S
0772 AF 64              STX     4,S
0774 ED E1              STD     0,S++       SET FLAGS, DISCARD STORE
0776 39                 RTS                 AND DISSAPEAR
                *
                *
          0777  TXTBGN  EQU     *
0777                    RMB     1000        MAKE SOME ROOM FOR PROGRAM
0B5F            DATAS   RMB     BUFEND
          0BBF  EOP     EQU     *
                        END     START

0 ERROR(S) DETECTED

SYMBOL TABLE:

ATSIGN 02CF   BUFEND 0060   BUFFER 0018   CHKIO  018A   CRLF   017D
CURRNT 0002   DATAS  0B5F   DIRECT 02F2   DIV    072D   DIV1   0738
DIV2   073E   DIV3   074E   DIV4   0758   DIV5   0736   DIVR   0748
ENDCHK 0153   EOP    0BBF   ERR0   05EE   ERROR  05C6   EXEC   01B6
EXEC1  01D0   EXECF  01DE   EXECP  01D8   EXP3EL 0253   EXP3ER 025E
EXPR   01ED   EXPR2  0227   EXPR21 0229   EXPR3  024F   EXPR4  026F
F1     04F6   FI1    0148   FIN    013B   FINE   015C   FLDCNT 0001
FNDLN  04D8   FNDLNP 04E1   FNDNXT 04EB   FOR2   0489   FOR3   0494
FOR7   04A0   FOR7L  04AF   FOR8   04BA   FORE   0476   GETLN  04FE
GL1    0508   GL3    0525   GL4    0534   GLO    053C   GOERR  0401
HIWAT  0014   HIWOK  02E7   I1     0170   IGNBLK 0172   INPERR 0397
IP1    039E   IP2    03AB   IP3    03C2   IP4    03E6   IP5    03F2
LCT    01BD   LET    03F4   LOPINC 000A   LOPLMT 000C   LOPLN  000E
LOPPT  0010   LOPVAR 0008   LS1    0303   M1     06D6   M2     06DA
MULT   06CB   NCQ    0764   NCR    076E   NEXT0  0442   NEXT2  044B
NEXT3  044D   NEXT4  0466   NEXT5  046A   NEXT6  045C   NUMOUT 0134
OCSW   0000   OK     063E   ONE    0217   OUTC   017F   PARM   02B7
PN1    0557   PN2    0559   PN3    0569   PN4    0567   PN6    0574
PN7    0580   PN8    0572   POPA   05F0   POPA0  060F   PR0    0356
PR1    0362   PR2    034C   PR3    036C   PR6    0378   PRTLN  0582
PRTNUM 0541   PRTSTG 058D   PRTSTL 058C   PU1    063A   PUSHA  0617
QHOW   012C   QSORRY 0167   QT1    05A5   QT3    05AC   QT4    05B8
QTSTG  059B   QW1    04D6   QWHAT  015E   RETURN 05C5   RLYAT  01B0
RUNNNL 0326   RUNNXL 0324   RUNSML 0332   RUNTSL 032D   S2     0703
SEED   0012   SETFLG 06F8   SETVAL 04C0   SIGN   06E4   SIZE   02A1
SPACE  0179   ST1    0641   ST2LE  06B2   ST3    065D   ST3LE  06BF
ST4    069A   START  0196   STKGOS 0006   STKINP 0006   STL    068E
STLE   068A   TABLE  009F   TMP    0491   TN1    0106   TSTNUM 0100
TSTVE  0199   TVHOW  0138   TVN    01B3   TXTBGN 0777   TXTUNF 0004
UNSIGN 070C   US2    071A   USR    072A   VARBGN FFCA   VARNXT 0006
XP18   021D   XP235  0237   XP290  023B   XP291  024D   XP43   02CD
ZER    0218
```

**end**

```
        MOV     A,M             ;Get byte containing requested bit pair
GETBP1  EQU     *               ;Loop
        RLC                     ;   Shift byte through two bits
        RLC
        DCR     C
        JNZ     GETBP1          ;Until requested pair is least significant
        ANI     X'03'           ;Reset all other bits
        RET
;
;   Instruction Length Table:
;
INLNTB  EQU     *               ;  Lengths:                 Opcodes:
        DB      X'75'           ;  1 3 1 1                  X'00'-X'03'
        DB      X'59'           ;  1 1 2 1                  X'04'-X'07'
        DB      X'15'           ;  0 1 1 1                  X'08'-X'0B'
        DB      X'59'           ;  1 1 2 1                  X'0C'-X'0F'
        DB      X'35'           ;  0 3 1 1                  X'10'-X'13'
        DB      X'59'           ;  1 1 2 1                  X'14'-X'17'
        DB      X'15'           ;  0 1 1 1                  X'18'-X'1B'
        DB      X'59'           ;  1 1 2 1                  X'1C'-X'1F'
        DB      X'7D'           ;  1 3 3 1                  X'20'-X'23'
        DB      X'59'           ;  1 1 2 1                  X'24'-X'27'
        DB      X'1D'           ;  0 1 3 1                  X'28'-X'2B'
        DB      X'59'           ;  1 1 2 1                  X'2C'-X'2F'
        DB      X'7D'           ;  1 3 3 1                  X'30'-X'33'
        DB      X'59'           ;  1 1 2 1                  X'34'-X'37'
        DB      X'1D'           ;  0 1 3 1                  X'38'-X'3B'
        DB      X'59'           ;  1 1 2 1                  X'3C'-X'3F'
        DB      X'5F'           ;  1 1 3 3                  X'C0'-X'C3'
        DB      X'D9'           ;  3 1 2 1                  X'C4'-X'C7'
        DB      X'54'           ;  1 1 1 0                  X'C8'-X'CB'
        DB      X'F9'           ;  3 3 2 1                  X'CC'-X'CF'
        DB      X'5E'           ;  1 1 3 2                  X'D0'-X'D3'
        DB      X'D9'           ;  3 1 2 1                  X'D4'-X'D7'
        DB      X'4E'           ;  1 0 3 2                  X'D8'-X'DB'
        DB      X'C9'           ;  3 0 2 1                  X'DC'-X'DF'
        DB      X'5D'           ;  1 1 3 1                  X'E0'-X'E3'
        DB      X'D9'           ;  3 1 2 1                  X'E4'-X'E7'
        DB      X'5D'           ;  1 1 3 1                  X'E8'-X'EB'
        DB      X'C9'           ;  3 0 2 1                  X'EC'-X'EF'
        DB      X'5D'           ;  1 1 3 1                  X'F0'-X'F3'
        DB      X'D9'           ;  3 1 2 1                  X'F4'-X'F7'
        DB      X'5D'           ;  1 1 3 1                  X'F8'-X'FB'
        DB      X'C9'           ;  3 0 2 1                  X'FC'-X'FF'
```

**end**

# Extra Function Keys

## Listing 1

```
                ;
                ;          K I P
                ;     Keyboard Input Patch
                ;       JOHN B. PALMER
                ;         02/16/80
                ;
                ; A NEW INPUT ROUTINE TO TRANSLATE KEYBOARD
                ; FUNCTIONS FROM A TABLE.
                ; THIS PROGRAM MUST BE LOADED BY DDT
                ; BEFORE USING THE PROGRAM 'KID.COM'

FC00 =          FRERAM  EQU     0FC00H
FD00 =          TEXT    EQU     FRERAM + 256
0001 =          KIFUN   EQU     01      ; KEYBOARD INPUT FUNCTION CODE
0009 =          TAB     EQU     09
000A =          LF      EQU     0AH
000D =          CR      EQU     0DH
001A =          EOF     EQU     01AH    ; END OF FILE

FC00                    ORG     FRERAM
                KIP:
FC00 C30000     OLDST:  JMP     0       ; TWO ADDRESS FILEDS TO BE
FC03 C30000     OLDIN:  JMP     0       ; ALTERED BY THE PROGRAM 'KID.COM'
FC06 C311FC     NEWST:  JMP     STAT
FC09 C319FC     NEWIN:  JMP     INP

00FF =          ACTIVE  EQU     0FFH
0000 =          INACTIVE EQU    0
                ; THE FOLOWING FIVE ITEMS ARE VARIABLES.,
FC0C 00         SWITCH  DB      0       ; ACTIVE WHEN STRING IS BEING GIVEN
FC0D 00         EOSTR:  DB      0       ; END OF STRING CHARACTER
FC0E 0000       STRPTR: DW      0       ; STRING POINTER
FC10 00         SAVCHR: DB      0       ; NEW CHARACTER FROM KEYBOARD

FC11 3A0CFC     STAT:   LDA     SWITCH
FC14 B7                 ORA     A       ; TEST FOR ZERO
FC15 C0                 RNZ
FC16 C300FC             JMP     OLDST   ; JUMP TO OLD STATUS
                        ;
FC19 3A0CFC     INP:    LDA     SWITCH
FC1C B7                 ORA     A       ; TEST FOR ZERO
FC1D C2C2FC             JNZ     STRING  ; YES, GIVE THE STRING
                        ;
FC20 CD03FC             CALL    OLDIN
FC23 FE01               CPI     KIFUN   ; KEYBOARD FUNCTION

FC25 C0                 RNZ     ; NO
                ; THE ABOVE 'RET' IS THE NORMAL RETURN FOR CONSOLE INPUT

FC26 CD03FC             CALL    OLDIN   ; GET NEXT CHARACTER
FC29 3210FC             STA     SAVCHR ; SAVE NEW CHARACTER
                        ; THE LEAR SIEGLER GIVES A CR AFTER THE FUNCTION KEY,
                        ; CALL THE BIOS TO IGNORE IT.
FC2C F620               ORI     20H     ; CONVERT TO LOWER CASE
FC2E D660               SUI     'A'     ; TEST IF IN RANGE OF 'A' TO 'A'+15

FC81 E1                 POP     H
FC82 D1                 POP     D
FC83 C9                 RET
                        ;
                VIAHL:
FC84 E9                 PCHL
                        ;

                FOUND:
FC85 7E                 MOV     A,M     ; THE STRING DELIMITER
FC86 320DFC             STA     EOSTR   ; SAVE IT
FC89 23                 INX     H       ; POINT TO FIRST CHARACTER IN STRING
FC8A 220EFC             SHLD    STRPTR  ; SAVE POINTER
FC8D 3EFF               MVI     A,ACTIVE
FC8F 320CFC     FIN:    STA     SWITCH
FC92 E1                 POP     H       ; RESTORE REGISTERS NOW
FC93 C1                 POP     B
FC94 C319FC             JMP     INP     ; GO BACK AND GIVE A STRING
                        ;
                SKIP:   ; NOW SKIP OVER THE STRING
FC97 7E                 MOV     A,M     ; GET DELIMITER
FC98 CDBAFC             CALL    EOFCHR  ; TEST FOR END OF FILE
FC9B D8                 RC
FC9C 4F                 MOV     C,A     ; SAVE DELIMITER
FC9D 23                 INX     H       ; HOP OVER IT.
FC9E 7E         TSTCHR: MOV     A,M     ; GET A  CHARACTER
FC9F 23                 INX     H       ; ALWAYS POINT AHEAD
FCA0 CDBAFC             CALL    EOFCHR
FCA3 D8                 RC
FCA4 B9                 CMP     C       ; AT THE END?
FCA5 C8                 RZ              ; YES, WE WENT PAST STRING AND DELIMITER
FCA6 C39EFC             JMP     TSTCHR
                        ;

                IGNORE: ; IGNORE SPACES, TABS, COMMAS, CR OR LF
                        ; ZERO FLAG MEANS MATCH, CARRY MEANS EOF
FCA9 7E                 MOV     A,M
FCAA 23                 INX     H       ; ALWAYS POINT AHEAD
FCAB FE20               CPI     ' '     ; SPACE
FCAD C8                 RZ
FCAE FE09               CPI     9       ; TAB
FCB0 C8                 RZ
FCB1 FE2C               CPI     ','     ; COMMA
FCB3 C8                 RZ
FCB4 FE0D               CPI     CR
FCB6 C8                 RZ
FCB7 FE0A               CPI     LF
FCB9 C8                 RZ

                EOFCHR: ; IF EOF, SET CARRY
                        ; ( THERE SHOULD NOT BE ANY OTHER CONTROL
                        ;    CHARACTERS IN THE FILE, BUT JUST IN CASE. )
FCBA FE1A               CPI     EOF
FCBC 37                 STC
FCBD 3F                 CMC             ; NO CARRY
FCBE C0                 RNZ             ; RETURN NZ, NC BECAUSE IT WAS NOT EOF
                        ;
                ; YES, IT WAS THE EOF CHARACTER
```

```
FC30 D610           SUI     16      ; FORCE CARRY IF <15
FC32 DC03FC         CC      OLDIN   ; EXPECT AN UNWANTED CHARACTER
FC35 3A10FC         LDA     SAVCHR  ; RESTORE PRIOR CHARACTER
                ; PRIOR CHARACTER WAS KIFUN, NOW SEARCH DEFINITION TEXT
                SEARCH:
FC38 C5             PUSH    B       ; SAVE REGISTERS
FC39 E5             PUSH    H
FC3A FE01           CPI     KIFUN   ; IS THIS ONE ALSO KIFUN?
FC3C CA5BFC         JZ      PEEK    ; HE WANTS TO PEEK
FC3F 47             MOV     B,A     ; SAVE
FC40 2100FD         LXI     H,TEXT

                TRY:
                                    ; SKIP SPACES, TABS ETC.
FC43 CDA9FC         CALL    IGNORE  ; MOV A,M  INX H
                                    ; IF EOF, THE CARRY IS SET

FC46 DA56FC         JC      EXIT
FC49 CA43FC         JZ      TRY     ; IGNORE SPACES, TABS
FC4C B8             CMP     B       ; FOUND?
FC4D CA85FC         JZ      FOUND   ; EXIT TO FOUND
FC50 CD97FC         CALL    SKIP
FC53 D243FC         JNC     TRY     ; IF CARRY, EXIT

                    ;
                EXIT:   ; COULD NOT FIND STRING
FC56 3E3F           MVI     A,'?'   ; GIVE A QUESTION MARK
FC58 E1             POP     H       ; RESTORE REGISTERS NOW
FC59 C1             POP     B
FC5A C9             RET             ; RETURN WITH '?'
                    ;
                PEEK:

                    ; SHOW WHAT IS NOW IN DEFINITION BLOCK
FC5B 3E0D           MVI     A,CR
FC5D CD74FC         CALL    DIO
FC60 3E0A           MVI     A,LF
FC62 2100FD         LXI     H,TEXT
                PRINT:
FC65 CD74FC         CALL    DIO
FC68 7E             MOV     A,M     ; GET CHAR.
FC69 23             INX     H

FC6A FE1A           CPI     EOF     ; AT END?
FC6C C265FC         JNZ     PRINT   ; REPEAT TILL DONE
FC6F E1             POP     H
FC70 C1             POP     B
FC71 3E0D           MVI     A,CR
FC73 C9             RET             ; BACK TO CPM WITH JUST CR
                    ;
                DIO:    ; CALL DIRECT I/O FOR CONSOLE OUTPUT
FC74 D5             PUSH    D
FC75 E5             PUSH    H
FC76 2A0100         LHLD    01
FC79 010900         LXI     B,9
FC7C 09             DAD     B
FC7D 4F             MOV     C,A
FC7E CD84FC         CALL    VIAHL

FCBF B7             ORA     A       ; NZ

FCC0 37             STC             ; CARRY
FCC1 C9             RET             ; RETURN NZ, C
                    ;
            STRING: ; WHEN SWITCH IS ACTIVE, RETURN CHARACTERS FROM STRING
FCC2 CDD3FC         CALL    GETCHR
FCC5 F8             RM              ; QUIT, SOMETHING WENT WRONG
FCC6 FE5E           CPI     '^'     ; IS NEXT CHARACTER TO BE A CONTROL?
FCC8 C0             RNZ             ; NO
FCC9 CDD3FC         CALL    GETCHR  ; GET NEXT ONE
FCCC F8             RM              ; QUIT IF SIGN SET
FCCD FE5E           CPI     '^'
FCCF C8             RZ              ; HE REALLY WANTS LITERALLY
            ; OTHERWISE TRANSLATE TO A CONTROL CODE
FCD0 E61F           ANI     01FH
FCD2 C9             RET
                    ;
FCD3 E5     GETCHR: PUSH    H
FCD4 2A0EFC         LHLD    STRPTR  ; GET CURRENT POINTER
FCD7 3E1A           MVI     A,EOF   ; END OF FILE?
FCD9 BE             CMP     M
FCDA CAF6FC         JZ      WIERD
FCDD 3A0DFC         LDA     EOSTR   ; GET END OF STRING CHARACTER
FCE0 BE             CMP     M

FCE1 CAF6FC         JZ      WIERD
FCE4 23             INX     H       ; UP POINTER
FCE5 220EFC         SHLD    STRPTR
FCE8 BE             CMP     M       ; TEST NEXT CHARACTER FOR END OF STRING
FCE9 2B             DCX     H       ; GO BACK
FCEA 6E             MOV     L,M     ; THIS IS THE CURRENT CHARACTER
FCEB C2F2FC         JNZ     READY
FCEE 97             SUB     A       ; ZERO
FCEF 320CFC         STA     SWITCH  ; TURN OFF SWITCH
                READY:
FCF2 7D             MOV     A,L
FCF3 E1             POP     H
FCF4 B7             ORA     A       ; CLEAR SIGN
FCF5 C9             RET             ; BACK TO CALLER
                    ;

                WIERD:  ; SOMETHING IS WRONG
FCF6 F1             POP     PSW     ; FIX STACK DUE TO
                                    ; ABNORMAL EXIT FROM ABOVE SUBROUTINE
FCF7 97             SUB     A       ; ZERO
FCF8 320CFC         STA     SWITCH  ; TURN OFF THE SWITCH
FCFB 3D             DCR     A       ; SET SIGN
FCFC 3E3F           MVI     A,'?'
FCFE E1             POP     H
FCFF C9             RET             ; BACK WITH '?' AND SIGN
                    ;
FD00                END     KIP
```

**end**

*(Continued on top of page 34)*

# Extra Function Keys (Continued)

## Listing 2

```
                        ; * * * *
                        ; *
                        ; *     K I D   -Keyboard Input Definition
                        ; *
                        ; * A PROGRAM TO DEFINE THE  SPECIAL KEYBOAD
                        ; * FUNCTIONS FOR MOST COMMON CRT'S. THIS
                        ; * PROGRAM EXPECTS THE DEFINTIONS TO BE IN A
                        ; * FILE OF TYPE '.KID', THE USER GIVING THE
                        ; * FIRST EIGHT CHARACTERS OF THE FILENAME
                        ; * WHEN THIS PROGRAM IS CALLED BY CP/M.
                        ; * BEFORE USING THIS PROGRAM, YOU MUST
                        ; * LOAD THE PROGRAM 'KIP.HEX' WITH DDT
                        ; * INTO YOUR MEMORY AREA ABOVE THE BIOS.
                        ; *
                        ; *     JOHN B. PALMER
                        ; *       02/16/80
                        ; *
                        ; * * * *
                        ;
                        ; Example:      the user gives the command
                        ; KID BASIC
                        ; cpm will load the file KID.COM into tpa and execute
                        ; 0100h. This program then loads the file BASIC.KID
                        ; into an area above the cpm bios, and modifys the
                        ; cosole status and input vectors of the bios.
                        ; From then on if the user hits control 'A' followed by
                        ; any number or letter, a word or phrase is generated.

0100                            ORG     0100H
                        KID:
0100 C30C01                     JMP     BEGIN
                        ; THE FRERAM AREA WILL CONTAIN A NEW INPUT CODE
FC00 =                  FRERAM  EQU     0FC00H  ; USER'S FREE RAM AREA

0103 00FC               FREPTR: DW      FRERAM  ; POINT TO FREE RAM
0105                    CSVPTR: DS      2       ; POINT TO CS VECTOR
0107                    CIVPTR: DS      2       ; POINT TO CI VECTOR

0109 0000               DMATMP: DW      0       ; DMA TEMPORARY
010B 00                 COUNT:  DB      0       ; NUMBER OF SECTORS READ

0004 =                  MAXI    EQU     4       ; DON'T READ MORE THAN FOUR

                        ; CPM ENTRY POINTS

0000 =                  BOOT    EQU     0       ; WARM BOOT
0005 =                  BDOS    EQU     5       ; CALL TO CPM
005C =                  FCBL    EQU     05CH    ; FIRST FILE CONTROL BLOCK
0065 =                  FTYP    EQU     FCBL + 9        ; FILE TYPE
007C =                  FCBCR   EQU     FCBL + 32       ; CURRENT RECORD
                        ; BDOS FUNCTIONS. C HAS FUNCTION NUMBER, DE IS POINTER, A HAS RETURN VALUE
0009 =                  PRINTF  EQU     9       ; PRINT BUFFER ( END IS '$')
000F =                  OPENF   EQU     15      ; OPEN FILE
0010 =                  CLOSEF  EQU     16      ; CLOASE FILE
0014 =                  READF   EQU     20      ; READ SEQUENTIAL
001A =                  SETDMA  EQU     26      ; SET ADDRESS FOR DISK BUFFER
                        ;
                    CLOSE:
0171 115C00                 LXI     D,FCBL
0174 0E10                   MVI     C,CLOSEF
0176 CD0500                 CALL    BDOS
0179 C30902                 JMP     MODIFY  ; NOW COMES THE HARD PART
                            ;
                    MESNOF:
017C 4E4F20                 DB      'NO '
                    KIDFIL:
017F 4B49442046             DB      'KID FILE $'
                    MESBIG:
0189 4B49442046             DB      'KID FILE IS TOO BIG $'
                    NOKIP:
019E 4B49502041             DB      'KIP ABSENT $'
                    ALLOK:
01AA 4B49442052             DB      'KID READ OK $'
                            ;


                        ; * * * * * * * * * *
                        ;
                        ; WARNING - THE CODE THAT FOLLOWS IS VERY RISKY.
                        ; BE VERY CAREFULL IF YOU HAVE TO CHANGE  EITHER
                        ; 'PATCH' OR 'MODIFY'.
                        ;
                        ; * * * * * * * * * *

                        PATCH:  ; PROBLEM:
                                ;       GET ADDRESS THAT IS LOCATED AT THE ADDRESS POINTED
                                ; BY CSVPTR AND PUT IT INTO ADDRESS POINTED BY FREPTR +1
                                ;       DO SAME FOR CIVPTR AND  FREPTR +4
                                ;
                                ;       GET ADDRESS POINTED BY FREPTR + 6  AND
                                ; PUT IT INTO ADDRESS POINTED BY CSVPTR.
                                ;       DO SAME FOR  FREPTR +9 AND CSVPTR.
                                ;
                                ; ( don't feel bad if you don't understand this, I don't
                                ;  think I am sure myself. )


                        ;   GET AN ADDRESS THAT POINTS TO BIOS CONSOLE STATUS VECTOR FIELD
01B7 2A0501                     LHLD    CSVPTR
                        ;   GET ADDRESS AT THAT LOCATION
01BA CDF101                     CALL    LOADDE  ; LOAD DE VIA HL
                        ;   GET ADDRESS OF FREE RAM
01BD 2A0301                     LHLD    FREPTR
01C0 23                         INX     H       ; ADD 1
                        ;   STORE ADDRESS
01C1 CDF501                     CALL    STORDE  ; STORE DE VIA HL

                        ; AGAIN FOR AN ADDRESS THAT POINTS TO BIOS CONSOLE INPUT VECTOR FIELD
01C4 2A0701                     LHLD    CIVPTR
01C7 CDF101                     CALL    LOADDE
01CA 2A0301                     LHLD    FREPTR
01CD 010400                     LXI     B,4
01D0 09                         DAD     B       ; ADD 4
```

```
 59:
 60:  000A =          LF      EQU     0AH
 61:  000D =          CR      EQU     0DH
 62:
 63:                  BEGIN:
 64:                          ;  PUT 'KID' INTO FILE TYPE
 65:  010C 116500           LXI     D,FTYP
 66:  010F 010300           LXI     B,03
 67:  0112 217F01           LXI     H,KIDFIL
 68:  0115 CD1B01           CALL    MOVIT
 69:  0118 C32601           JMP     OPEN
 70:                  MOVIT:
 71:  011B 78               MOV     A,B
 72:  011C B1               ORA     C
 73:  011D C8               RZ              ; ALL DONE
 74:  011E 7E               MOV     A,M
 75:  011F 12               STAX    D
 76:  0120 23               INX     H
 77:  0121 13               INX     D
 78:  0122 0B               DCX     B
 79:  0123 C31B01           JMP     MOVIT
 80:                        ;
 81:                  OPEN:
 82:  0126 2A0301           LHLD    FREPTR  ; GET ADDRES OF THE FREE RAM AREA
 83:  0129 110001           LXI     D,256   ; PUT TEXT ABOVE THE NEW INPUT CODE AREA
 84:  012C 19               DAD     D
 85:  012D 220901           SHLD    DMATMP  ; SAVE
 86:  0130 EB               XCHG
 87:  0131 0E1A             MVI     C,SETDMA
 88:  0133 CD0500           CALL    BDOS    ; SET DMA
 89:
 90:  0136 115C00           LXI     D,FCBL  ; POINT TO FILE CONTROL BLOCK
 91:  0139 0E0F             MVI     C,OPENF ; OPEN FILE
 92:  013B CD0500           CALL    BDOS
 93:  013E 3C               INR     A       ; TEST FOR 0FFH
 94:  013F 117C01           LXI     D,MESNOF

 95:  0142 CA2902           JZ      FINISH  ; COULD NOT FIND FILE
 96:                        ;
 97:                  READ:
 98:  0145 115C00           LXI     D,FCBL
 99:  0148 0E14             MVI     C,READF ; SEQUENTIAL READ
100:  014A CD0500           CALL    BDOS
101:  014D B7               ORA     A       ; TEST FOR ZERO
102:  014E C27101           JNZ     CLOSE   ; END OF READ
103:                        ;
104:  0151 210B01           LXI     H,COUNT
105:  0154 34               INR     M
106:  0155 3E04             MVI     A,MAXI
107:  0157 BE               CMP     M
108:  0158 118901           LXI     D,MESBIG
109:  015B DA2902           JC      FINISH  ; FILE IS TOO LARGE
110:  015E 2A0901           LHLD    DMATMP
111:  0161 118000           LXI     D,128   ; SECTOR SIZE
112:  0164 19               DAD     D
113:  0165 220901           SHLD    DMATMP
114:  0168 EB               XCHG
115:  0169 0E1A             MVI     C,SETDMA
116:  016B CD0500           CALL    BDOS    ; SET DMA
117:  016E C34501           JMP     READ
```

*(Continued on top of page 36)*

```
172:  01D1 CDF501           CALL    STORDE
173:
174:                  ; NOW THE SECOND HALF OF THE PROBLEM.
175:  01D4 2A0301           LHLD    FREPTR
176:  01D7 010600           LXI     B,6
177:  01DA 09               DAD     B       ; ADD 6
178:  01DB EB               XCHG
179:  01DC 2A0501           LHLD    CSVPTR
180:  01DF CDF501           CALL    STORDE
181:
182:                  ; AGAIN FOR (FREPTR)+9, CIVPTR
183:  01E2 2A0301           LHLD    FREPTR
184:  01E5 010900           LXI     B,9
185:  01E8 09               DAD     B       ; ADD 9
186:  01E9 EB               XCHG
187:  01EA 2A0701           LHLD    CIVPTR
188:  01ED CDF501           CALL    STORDE

189:  01F0 C9               RET             ; ALL DONE WITH PATCH
190:                        ;
191:                  LOADDE:
192:  01F1 5E               MOV     E,M
193:  01F2 23               INX     H
194:  01F3 56               MOV     D,M
195:  01F4 C9               RET
196:                        ;
197:                  STORDE:
198:  01F5 73               MOV     M,E
199:  01F6 23               INX     H
200:  01F7 72               MOV     M,D
201:  01F8 C9               RET
202:                        ;

203:                  TSTJMP: ; TEST IF 'JMP 0' IS AT ADDRESS POINTED BY FREPTR
204:  01F9 2A0301           LHLD    FREPTR
205:  01FC 3EC3             MVI     A,0C3H  ; IF NO JMP, KIP IS ABSENT
206:  01FE BE               CMP     M
207:  01FF 37               STC
208:  0200 C0               RNZ             ; RETURN WITH CARRY SET
209:  0201 23               INX     H

210:                  ; IF NEXT EITHER NEXT TWO NOT ZERO, DON'T PATCH
211:  0202 7E               MOV     A,M
212:  0203 B7               ORA     A       ; TEST FOR ZERO, CLEAR CARRY
213:  0204 C0               RNZ
214:  0205 23               INX     H
215:  0206 7E               MOV     A,M
216:  0207 B7               ORA     A
217:  0208 C9               RET
218:                        ;

219:                  MODIFY:
220:  0209 2A0100           LHLD    01      ; GET ADDRESS OF WARM BOOT
221:  020C 110300           LXI     D,3     ; UP BY THREE
222:  020F 19               DAD     D       ; COMPUTE CS VECTOR
223:  0210 23               INX     H       ; POINT TO THE ADDRESS CONSOLE STATUS
224:  0211 220501           SHLD    CSVPTR  ; SAVE FOR LATER USE
```

*(Continued on next page)*

# Extra Function Keys (Continued)

```
225:  0214 23              INX     H
226:  0215 23              INX     H
227:  0216 23              INX     H       ; POINT TO ADDRESS OF CONSOLE INPUT
228:  0217 220701          SHLD    CIVPTR
229:                ; NOW HAS THE BIOS VECTOR ALREADY BEEN MODIFIED?
230:                ; IS 'KIP' PRESENT?
231:  021A CDF901            CALL    TSTJMP  ; IF ZERO FLAG AND NO CARRY, SAFE TO PATCH
232:  021D 119E01            LXI     D,NOKIP
233:  0220 DA2902            JC      FINISH  ; KIP WAS ABSENT
234:  0223 CCB701            CZ      PATCH
235:  0226 11AA01            LXI     D,ALLOK



236:                       ;
237:                FINISH:
238:  0229 0E09              MVI     C,PRINTF
239:  022B CD0500            CALL    BDOS    ; PRINT A MESSAGE
240:  022E C30000            JMP     BOOT
241:  0231                   END     KID

A>DDT KIP.HEX
DDT VERS 2.2
NEXT  PC
FD00 FC00
-GO

A>TYPE CPM.KID
@"STAT "  A"ASM "  B"BASIC "  C"CON:"
D"DIR "   E"ERA "  F"PIP "    G"PRN:"
H"RDR:"   I"PUN:"  J"SUB JOB" K"KID "
L"LOAD "  M"MAC "  N"NULL:"   O"TYPE "
1"PIP^M"  2"DIR^M" 3"B:STAT^M"
8"^>"     9"^^"    0"^C"      q"^A"
`"STAT *.*^M" a"A:^M" b"B:^M"


A>
A>TYPE BASIC.KID
1¦LOAD "¦,  2¦SAVE "¦,  3¦FILES ¦,  4¦RUN^M¦,
5¦LIST^M¦,  6¦FILE$¦"¦, 7¦RUN FILE$^M¦
8¦SYSTEM^M¦,  9¦BASIC^M¦
@¦PRINT ¦  A¦PRINT USING ¦
B¦TAB(¦
C¦FOR ¦  D¦NEXT ¦
E¦IF ¦  F¦THEN ¦  G¦ELSE ¦
H¦INPUT ¦
I¦GOTO ¦
J¦GOSUB ¦  K¦RETURN ¦
L¦LEFT$(¦  M¦RIGHT$(¦  N¦MID$(¦  O¦LEN¦

A>
A>KID CPM
KID READ OK
A>KID WHO
NO KID FILE
A>
```

**end**

```
385 REM
390       T1=45E-9 :T2=8   :R1=10E3 :R2=25E3
395       C1=T1/0.32/R2/(1+0.7/R2) :C2=1E-3
400       GOTO 615
405 REM
410       T1=72E-9 :T2=8   :R1=10E3 :R2=25E3
415       C1=T1/0.31/R2/(1+0.7/R2) :C2=1E-3
420       GOTO 615
425 REM
430       T1=30E-9 :T2=14  :R1=2.8E3 :R2=20E3
435       C1=T1/0.693/R2 :C2=1E-3
440       GOTO 615
445 REM
450       T1=40E-9 :T2=8   :R1=10E3 :R2=25E3
455       C1=T1/0.32/R2/(1+0.7/R2) :C2=1E-3
460       GOTO 615
465 REM
470       T1=40E-9 :T2=8   :R1=10E3 :R2=25E3
475       C1=T1/0.32/R2/(1+0.7/R2) :C2=1E-3
480       GOTO 615
485 REM
490       T1=30E-9 :T2=70  :R1=2.8E3 :R2=50E3
495       C1=SQRT(T1/R2↑2/3.03) :C2=1E-3
500       GOTO 615
505 REM
510       T1=50E-9 :T2=175 :R1=20E3 :R2=175E3
515       C1=T1/R2 :C2=1E-3
520       GOTO 615
525 REM
530     #""
535     #"THE PULSE WIDTH FOR THE CMOS CHIP YOU CHOSE DEPENDS"
540     #"UPON THE SUPPLY VOLTAGE."
545     INPUT"WHAT SUPPLY VOLTAGE WILL YOU USE ? (5 - 15) ",A$
550       V=VAL(A$)
555       IF(V<5) OR (V>15) THEN 545
560 REM
565     IF N <> 9 THEN 590
570       T1=(-5*V + 100)*1E-9 :T2=270 :R1=10E3 :R2=500E3
575       C1=T1/0.2/R2/LOG(V) :C2=1E-3
580       GOTO 615
585 REM
590       T1=(-15*V + 315)*1E-9 :T2=270 :R1=10E3 :R2=500E3
595       C1=T1/0.2/R2/LOG(V) :C2=1E-3
600 REM
605 REM - - - - - INPUT AND TEST WANTED PULSE WIDTH - - - - -
610 REM
615     #""
620     INPUT"WHAT LENGTH PULSE IN MICROSECONDS ?  ",T
625     T=T*1E-6                    :REM  CONVERT TO SECONDS
630       IF T => T1 THEN 645
635         #"PULSE MUST BE MORE THAN ";T1*1E6;" MICROSECONDS"
640         GOTO 615
645       IF T <= T2 THEN 675
650         #"PULSE MUST BE LESS THAN ";T2*1E6;" MICROSECONDS"
655         GOTO 615
660 REM
665 REM - - - - INPUT AND TEST CAPACITOR'S VALUE - - - - - -
670 REM
675     #""
680     INPUT"WHAT STANDARD VALUE CAPACITOR IN MICROFARADS ?  ",C
685     C=C*1E-6                    :REM  CONVERT TO FARADS
690       IF C => C1 THEN 705
695         #"USE A LARGER VALUE THAN ";C1*1E6;" MICROFARADS"
```

# CAD for Time Delay Circuits

```
LIST

100 REM  ONESHT     13 AUGUST 79     HAL KNIPPENBERG
105 REM
110 REM  PROGRAM HELPS DESIGN TIMING COMPONENTS FOR ONE-SHOT
115 REM  MULTIVIBRATORS.
120 REM
125 REM  REFERENCE POPULAR ELECTRONICS, SEPT 79, PAGE 69, FOR
130 REM  LIMITATIONS AND TIMING FORMULAS
135 REM
140 REM  VARIABLES:
145 REM     A$.....ANSWERS TO MISC MACHINE QUESTIONS
150 REM     C......CAPACITORS VALUE
155 REM     C1.....MINIMUM VALUE FOR CAPACITOR
160 REM     C2.....MAXIMUM VALUE FOR CAPACITOR
165 REM     N......NUMBER OF CHOICE OF CHIP
170 REM     R......RESISTORS VALUE
175 REM     R1.....MINIMUM VALUE FOR RESISTOR
180 REM     R2.....MAXIMUM VALUE FOR RESISTOR
185 REM     T......TIME PERIOD
190 REM     T1.....MINIMUM TIME PERIOD
195 REM     T2.....MAXIMUM TIME PERIOD
200 REM     V......SUPPLY VOLTAGE
205 REM     Z......CALLS MACHINE LANGUAGE SUBROUTINE
210 REM
215 REM     # IS THE ABBREVIATION FOR 'PRINT'
220 REM     THIS PROGRAM WILL FUNCTION WITHOUT REM STATEMENTS
225 REM
230 REM * * * * * * * * * * * * * * * * * * * * * * * * * *
235 REM                 START MAIN ROUTINE
240 REM
245     Z=CALL(12762)              :REM  CLEAR TV SCREEN
250 REM
255 REM - - - - - CHOOSE CHIP TO BE USED - - - - - - - - - -
260 REM
265     #"COMMON ONE-SHOT CHIPS    (POSSIBLE PULSE WIDTHS)"
270     #""
275     #"1.  9600  (74 NS - 8 SEC)        6.    74123 (40 NS - 8 SEC)"
280     #""
285     #"2.  9601  (45 NS - 8 SEC)        7.  74LS221 (30 NS - 70 SEC)"
290     #""
295     #"3.  9602  (72 NS - 8 SEC)        8.   74C221 (50 NS - 175 SEC)"
300     #""
305     #"4. 74121  (30 NS - 14 SEC)       9.    4098B (30 NS - 270 SEC)"
310     #""
315     #"5. 74122  (40 NS - 8 SEC)       10.   4528B/ (100 NS -270 SEC)"
320     #"                                      MC14528C"
325     #""
330     INPUT"WHICH CHIP WILL YOU USE ? (1 - 10)  ",N
335       IF N < 1 THEN 330
340       IF N > 10 THEN 330
345 REM
350 REM - - - - - SET DESIGN LIMITS - - - - - - - - - - - -
355 REM
360     ON N GOTO 370,390,410,430,450,470,490,510,530,530
365 REM
370       T1=74E-9 :T2=8   :R1=10E3 :R2=25E3
375       C1=T1/0.32/R2/(1+0.7/R2) :C2=1E-3
380       GOTO 615
```

*(Continued on top of page 38)*

```
700         GOTO 675
705       IF C <= C2 THEN 735
710         #"USE A SMALLER VALUE THAN ";C2*1E6;" MICROFARADS"
715         GOTO 675
720 REM
725 REM - - - - - COMPUTE AND TEST RESISTOR VALUE - - - - -
730 REM
735     #""
740     ON N GOTO 745,745,750,755,745,745,760,765,770,770
745     R=(T/0.32/C) - 0.7   :GOTO 775
750     R=(T/0.31/C) - 0.7   :GOTO 775
755     R=T/0.693/C          :GOTO 775
760     R=SQRT(T/3.03/C^2)   :GOTO 775
765     R=T/C                :GOTO 775
770     R=T/0.2/C/LOG(V)
775       IF R<R1 THEN #"USE A SMALLER VALUE C"  :GOTO 675
780       IF R>R2 THEN #"USE A LARGER VALUE C"   :GOTO 675
785 REM
790 REM - - - - - - PRINT RESULTS - - - - - - - - - - - - -
795 REM
800     Z=CALL(12762)              :REM  CLEAR TV SCREEN
805 REM
810     #"FOR A PULSE WIDTH OF ";T*1E6;" MICROSECONDS, USE:"
815     #""
820     #"      CAPACITOR = ";C*1E6;" MICROFARADS"
825     #"      RESISTOR APPROX = ";INT(R+.5);" OHMS"
830 REM
835 REM - - - - - - GET COMMON VALUES - - - - - - - - - - -
840 REM
845     #""
850     #""
855     #"WHAT IS THE CLOSEST STANDARD VALUE RESISTOR YOU"
860     INPUT"CAN USE ?  ",R
865       IF R => R1 THEN 880
870         #"RESISTOR MUST BE GREATER THAN ";R1;" OHMS"
875         GOTO 850
880       IF R <= R2 THEN 910
885         #"RESISTOR MUST BE LESS THAN ";R2;" OHMS"
890         GOTO 850
895 REM
900 REM - - - - COMPUTE AND PRINT PRACTICAL RESULTS - - - -
905 REM
910     ON N GOTO 915,915,920,925,915,915,930,935,940,940
915     T=0.32*R*C*(1+0.7/R)     :GOTO 950
920     T=0.31*R*C*(1+0.7/R)     :GOTO 950
925     T=0.693*R*C              :GOTO 950
930     T=R*C*(3.03*R*C)         :GOTO 950
935     T=R*C                    :GOTO 950
940     T=0.2*R*C*LOG(V)
945 REM
950     #"" :#""
955     #"USING A CAPACITOR OF ";C*1E6;" MICROFARADS AND A"
960     #"RESISTOR OF ";R;" OHMS WILL RESULT IN A PULSE"
965     #"WIDTH OF ";T*1E6;" MICROSECONDS."
970 REM
975 REM - - - - - CHECK FOR ANOTHER DESIGN - - - - - - - - -
980 REM
985     #"" :#""
990     INPUT"   ANOTHER DESIGN ?  ",A$
995     IF A$(1,1) = "Y" THEN 245 :REM MICROSOFTS USE LEFT$(A$,1)
1000     #"" :#""
1005 REM
1010 REM - - - - - - - - - - - - - - - - - - - - - - - - - -
1015                           END
READY
```

**end**

# Software Review

**SCRIBE IX**
Published by **Friedman Ads/Promotions**
**P.O. Box 234, Shiba**
**Tokyo, Japan 105-91**
Price: **$200**
Reviewed by **Gene Head**

Well friends, the software from Japan is here! I don't know if the package I received is representative of a coming trend, but if it is, all micro enthusiasts stand to benefit greatly.

SCRIBE IX, (I presume pronounced scribe-nine) is the Asian answer to the Electric Pencil. From my experience with an older version of Pencil, I can say SCRIBE IX is a direct replacement with added features, at less than two-thirds the price of Pencil, and is delivered as a SOURCE file. This source file is intended to be assembled with a CP/M assembler and operated in the CP/M environment. While certain modifications to the source code would allow operation on other host systems, I doubt if this would be within the realm of most programmers.

Dave Friedman, whose article on a video slot machine you may have read in the November/December issue of *DDJ*, is the author of this fine package. Frankly, I question Mr. Friedman's decision to supply the source code for this sophisticated program at such a low price. I would suspect the unscrupulous might steal his product and re-sell it. There is no need to even sign a license agreement to secure the source disk! However, the price is quite reasonable and this may encourage others to follow suit if Friedman doesn't get swindled out of business before the idea catches on. I can only imagine the chagrin of established software producers when they witness the action taken by Mr. Friedman.

## An Overview

SCRIBE IX is a complete word processing package with features built in to enable extensive mailing list functions. These mailing functions include selected auto insert from appended files, complete with letter salutation and other keyed-text insertions. There are two subsystems, one for printing and one for disk activities. The screen editing mode has all the features expected in a high-quality, professional word processor. The command mode functions similarly to the ED editor supplied with CP/M.

The manual is complete and clear. I It took more than one reading to comprehend some of the more sophisticated features, but I can not imagine the manual being written any better. Best of all, it is printed and shipped as a document. Some software I have reviewed is documented on the disk and the end user is to use the system printer to generate a hard copy. Another little extra touch is the inclusion of a summary card. On one side are the basic editing commands and the other side has the print formatting commands. Once the system is learned it is easy to recall a seldom-used command without referring to the manual. This handy card is the same size as a disk and fits neatly into the disk jacket.

## Screen Editing

The top line of the screen always displays the TAB stop positions. These are easily cleared and selectively reset or set to a default value of one tab every five spaces. Forward and backward scrolling occurs between this static line and the bottom of the screen. All the expected cursor controls are available. As a matter of fact, all but two control characters operate exactly the same as in the Electric Pencil. Control-H is a BACKSPACE and control-O is the block copy command. Exiting the system is accomplished in the Disk sub-system.

There is a special user-defined key, @, that when defined will output any ASCII character. If left undefined its value defaults to 64 decimal, or @. If defined with a new decimal value, that value will be substituted. This will be especially helpful for printers requiring special characters for custom operation.

The search mode provides another feature not found on my early version of Pencil. The SEARCH command can be initiated as expected with the standard search, or search and replace. Control-C is used to continue additional searching after a string is located. However, here is where SCRIBE IX begins to show real secretary-type know-how!

There are five sub-commands available in the search mode. A brief description follows. I didn't appreciate the value of these when I read their description in the manual. It's one of those operations you have to experience first hand to fully appreciate.

U This command toggles the search mode from precision to non-precision. In non-precision, no distinction is made between upper- and lower-case characters. Wild-card, or don't-care characters are valid and held by the sign #.

C This is the command to continue a search.

R This is the REPLACE command. Typing this will allow selective use of the search and replace function. Upon viewing each occurrence of the searched-for string, individual decisions can be made to replace or leave the existing string intact.

D This command will delete the original string. Again, this is a selective function as strings are found.

T This is a tabulated activity report! This will display all locations, replacements and deletions issued in this edit session.

ESCape can be issued at any time, returning to the screen edit mode.

There is less than one page in the manual to describe these functions. After several weeks of using SCRIBE IX to write reports, articles, and source code, I am very impressed with this unique feature.

Two additional features, which I can not take advantage of due to the antique nature of my selectric printer, are BOLD MARKING and UNDERLINING. Special characters in the text command the printer to perform underlining and bold typing when hard copy is finally issued through the print sub-system.

One final note about the screen editing. When the control-N command is issued to place the cursor at the end of the current file, the screen is cleared *except* for the last line of text in the file. This last text line becomes the first line on the display with the cursor in the home position. When using Pencil this last line is not displayed and that has always made me a bit nervous. In fact, I habitually had to scroll back down a few lines to maintain my train of thought when using Pencil.

## Printing Sub-System

The most obvious improvement in printing is the ability to dynamically change formatting parameters while printing. Essentially, any record (a record is defined as all the characters between line feed characters) that begins with a period is not printed through the print subsystem. These records can contain non-printing comments as well as recognizable commands. Basic commands found on most quality formatters are available, and then some. For example the entire operations manual is printed using the OUTDENTING function. Instead of each record (paragraph) being indented on the

first line it is outdented. I found this format visually pleasing. The Emit command allows any ASCII decimal value to be sent to the printer. I use this to ring the bell several times when a page of copy is finished printing. Auto insert will be outlined later in the mail list description. Parameters may be strung together to present a MACRO command list that can be saved as a file for standard operations. Page look ahead allows new pages to be started if a complete record, or optionally, any number of lines of a record, cannot fit on the current page. Form stops are set or re-set depending on continuous-fed paper or individual sheets.

There are driver routines for standard ASCII teletype type printers, Qume Sprint 5, two versions of NEC Spinwriters and two versions of Diablo HyTerm. The source code for these routines are very complete and are all present in the assembled object module. They can be selected through a printer ID command in the print sub-system. If your printer supports proportional spacing this value is also dynamically selectable.

The nicest feature of the printing sub-system that my Pencil lacks is the ability to view on the screen exactly what will be sent to the printer. This view feature saves paper and frustration.

With my old Selectric it is impossible to properly comment on the printing capabilities of the sub-system. When combined with one of the many newer printers SCRIBE IX supports, I can only imagine (dream?) the full capacity of SCRIBE IX.

## Disk File Sub-System

Control-K exits the screen editor and enters the disk sub-system. An ESCape will return to the screen editor. In the sub-system, several functions are available and two important parameters are displayed. Both the percent of disk space currently used by existing files and the percent of RAM buffer currently occupied by text is displayed. This lets you know when your disk is filling up and when you are running out of RAM for the current file. My 54K CP/M system used only 31 percent of the available space for this entire article.

Control-D displays the directory, listing only files with the .PCL extension. In addition, the disk usage parameter is updated. Disks can be loaded and exchanged without any problems. The only possible exception is operator error when a bad disk (wrong density, wrong format, no disk) is interrogated. This causes a CP/M system error and returns to CP/M when it is corrected. This mistake will destroy your current efforts that have not been previously committed to disk. I think there may be a way around this potential problem, but I have not researched it yet. I guess I will wait until I lose a thirty-page manuscript before I try to find a solution.

Files are saved and loaded from this sub-system and can be destroyed with the Kill command to reclaim needed space when required. Since .PCL is the only valid file extension recognized, existing files written with Pencil can be loaded and saved with no conversion or renaming required. However, I have found that into a long edit session, I want to permanently keep everything up to a certain point. I can save the existing file and use the Rename command to protect it from further tampering by careless saves and loads. I wish the Save command would first issue a control-B to put the cursor at the beginning of the text BEFORE EVERY SAVE command. More than once I have Saved before homing the cursor to the beginning of text. I have never lost anything because I have always noticed the mistake early enough to fix the damage. My pilot friends tell me there are two types of retractable gear flyers: those who have landed with their gear up and those who are going to. I suspect an applicable corollary exists here.

Clearing the text buffer and clearing before and after the cursor are both sub-system commands. The "X" command begins an exit to CP/M with the prompt "All text saved ? (Y/N) ". Fortunately, I've never missed here. It seems foolproof.

## Supporting A Mailing List

Using the auto insert command, two functions are available that will make standardized letters and mass mailings quite easy.

First, standard records or paragraphs can be given pseudo names. Then a complete customized letter or other document can be generated by stringing together specific records with very simple commands. The manual has step-by-step instructions for generating standardized mailings and even has a sample file written to demonstrate the capabilities. It was very easy to follow.

Second, lists of names and addresses can be generated and coded to provide selective processing. Up to 99 different codes can be spread throughout a file and selectively pulled for printing. This is *not* a sophisticated mailing list program with sorts and searches and the like, but it is a solid system linked to a remarkable word processor that will meet the needs of small- to medium-sized lists, say 400 to 600 names. With the search and replace function City, State, and Zip can be a single character when entering a number of names and quickly replaced when the job is completed by the actual string prior to printing. This can conserve a lot of disk space. Memory size is important in this area, as inserted names and addresses must be appended at the end of the form letter. They are not retrieved from disk as needed, but rather reside in memory at the end of the standard text.

## What About This Source Code?

There were only two lines of code that needed to be adjusted (*) to make the object work on my machine. The code looks something like Figure 1.

Of the three screen sizes, only one can be selected TRUE. Second, the VIDEO SCREEN ADDRESS equate had to be set to 0F00H for my IMSAI VIO-C card. These values are tested during conditional assembly and the proper object code is generated, depending on the system configuration. The code assembled flawlessly and the object module worked the first time out.

There is a second source file to make the Northstar Horizon vector interrupts operate the SCRIBE IX at maximum speed. My non-Horizon Northstar system uses an old George Risk keyboard and a very slow strobe, and I see no sacrifice in speed without the interrupt feature. Maybe if I were to use a system with an inter-

**Figure 1.**

```
* VID8024  EQU  TRU      ;SET TRUE FOR 80 X 24 SCREEN
VID6420  EQU  FLASE     ;SET TRUE FOR 64 X 20 SCREEN
VID6416  EQU  FLASE     ;SET TRUE FOR 64 X 16 SCREEN
  * VIDADD  EQU  0F00H      ;VIDEO SCREEN ADDRESS
```

rupted keyboard I would notice a change in speed. However, the keyboard buffering is such that I never lose characters, even on the longest of wrap-around words being copied to the next line. I always had to wait with the Pencil, and had to frequently backspace over mistakes. The program, by the way, was delivered to operate without modification on a Horizon II DD system.

Delivering source is the best thing that an experimenter/hacker could hope for. It took several weeks to locate the line feed followed by a carriage return in Pencil and swap these two characters to drive my selectric without a lot of I/O handling in CP/M. With SCRIBE IX, the CR and LF are defined in the first ten lines of code. Locating the LF followed by CR routines was simple. Customizing specific areas should prove relatively simple for those with special needs. While the code is not commented, the labels and defined equates are very easy to follow and seem quite logical. Modules can be spotted and understood as to function without much head scratching. I have written to Mr. Friedman twice and received prompt replies both times. I suspect it is his intention to keep the customers supplied with whatever information they seek. His licensing agreement is quite liberal. It basically says not to rip him off and sell the program, or make money off it by putting it in a product you do sell. Pretty simple. You don't sign anything, you agree to abide by the agreement when you accept the software.

## Conclusion

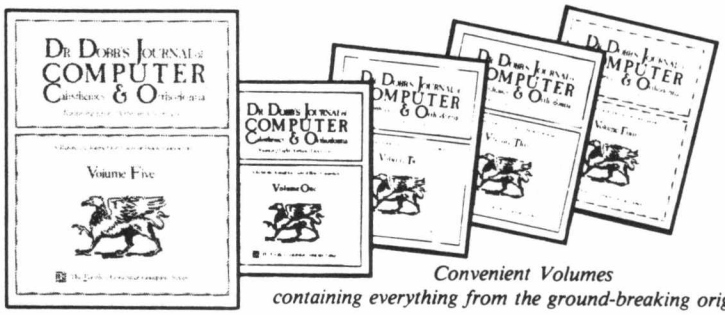
There is not enough room to expand properly about the features of SCRIBE IX. I am no expert. I have used Pencil extensively, and have researched every word processor article and advertisement during 1980 in most of the computer mags. Twice I asked the folks at Pencil for information and never heard from them. Basically, SCRIBE IX delivers a product superior to Pencil at less than two-thirds the cost. I can't directly compare with Word Pro and the other higher-priced packages. However, from the articles and the ads I am confident SCRIBE IX will hold its own, especially in the basic screen editing functions and disk I/O. The auto inserting for mailing is a bit primitive by some standards, but for overall operation and comparative cost the buyer of SCRIBE IX is getting a bargain.

■■


For Your Personal

# REFERENCE COLLECTION

## Thousands of dollars' worth of microcomputer information and listings!




*Convenient Volumes*
*containing everything from the ground-breaking originals*

**Dr. Dobb's Journal Vol. V** – All the ground-breaking issues from 1980 in one volume! Systems software reached a new level with the advent of CP/M, chronicled herein by Gary Kildall and others (*DDJ*'s all-CP/M issue sold out within weeks of publication). Software portability became a topic of greater import, and *DDJ* published Ron Cain's immediately famous Small-C compiler – reprinted here in full!

Contents include: The Evolution of CP/M, a CP/M-Flavored C Interpreter, Ron Cain's C Compiler for the 8080, Further with Tiny BASIC, a Syntax-Oriented Compiler Writing Language, CP/M to UCSD Pascal File Conversion, Runtime Library for the Small-C Compiler, and, as always, even more!

**Dr. Dobb's Journal Vol. I** – The material brought together in this volume chronicles the development in 1976 of Tiny BASIC as an alternative to the "finger blistering" front-panel machine-language programming which was then the only way to do things. This is always pertinent for bit crunching and byte saving, language design theory, homebrew computer construction and the technical history of personal computing.

Topics include: Tiny BASIC, the (very) first word on CP/M, Speech Synthesis, Floating Point Routines, Timer Routines, Building an IMSAI, and more.

**Dr. Dobb's Journal Vol. II** – 1977 found *DDJ* still on the forefront. These issues offer refinements of Tiny BASIC, plus then state-of-the-art utilities, the advent of PILOT for microcomputers and a great deal of material centering around the Intel 8080, including a complete operating system. Products just becoming available for reviews were the H-8, KIM-1, MITS BASIC, Poly Basic and NIBL.

Articles about Lawrence Livermore Lab's BASIC, Alpha-Micro, String Handling, Cyphers, High Speed Interaction, I/O, Tiny Pilot & Turtle Graphics, many utilities, and even more.

**Dr. Dobb's Journal Vol. III** – The microcomputer industry entered its adolescence in 1978. This volume brings together the issues which began dealing with the 6502, with mass-market machines and languages to match. The authors began speaking more in terms of technique, rather than of specific implementations; because of this they were able to continue laying the groundwork industry would follow. These articles relate very closely to what is generally available today.

Languages covered in depth were SAM76, Pilot, Pascal and Lisp, in addition to RAM Testers, S-100 Bus Standard Proposal, Disassemblers, Editors and much, much more.

**Dr. Dobb's Journal Vol. IV** – This volume heralds a wider interest in telecommunications, in algorithms, and in faster, more powerful utilities and languages. Innovation is still present in every page, and more attention is paid to the best ways to use processors which have proven longevity – primarily the 8080/Z80, 6502, and 6800. The subject matter is invaluable both as a learning tool and as a frequent source of reference.

Main subjects include: Programming Problems/Solutions, Pascal, Information Network Proposal, Floating Point Arithmetic, 8-bit to 16-bit conversion, Pseudorandom Sequences, and Interfacing a Micro to a Mainframe – more than ever!

**YES!** ☐ Please send me the following Volumes of *Dr. Dobb's Journal* for $20.75 each.
☐ ALL 5 for ONLY $88, a savings of over 15%!

Please charge my: ☐ Visa ☐ MasterCard ☐ American Express
I enclose ☐ Check/money order

Card # _____ Expiration Date _____

Signature _____

Name _____ Address _____

City _____ State _____ Zip _____

*Prices subject to change.*

Mail to: Dr. Dobb's Journal, P.O. Box E, Menlo Park, CA 94025
Allow 6-9 weeks for delivery.

Vol. I ____ x $20.75 = ____
Vol II ____ x $20.75 = ____
Vol. III ____ x $20.75 = ____
Vol IV ____ x $20.75 = ____
Vol. V ____ x $20.75 = ____
ALL 5 ____ x $88.00 = ____
Sub-total $____

**Postage & Handling** ____
**Must be included with order.**
*Please add $1.25 per book in U.S. ($1.75 each outside U.S.)*

**TOTAL** $____

*(Continued from page 6)*

```
*BM-CSbb↑Zb↑Z  (remove multiple
                 blanks)
*B#Sb↑Z↑L↑I↑Lbb↑Z
               (blanks to form-feed,
                 tab, form-feed,
                 2 blanks)
*E             (end edit and close
                 file)
```

Now use PIP as follows:

```
A> PIP
* CON:=FILE.TXT[P2T255]
```

At 9600 baud, the [P2T255] modifier causes single word display at the top of the screen at a rate of over 200 words per minute. Halving the tab expansion modifier roughly doubles the word rate, e.g.

```
A> PIP
* CON:=FILE.TXT[P2T128]
```

. . . displays words at about 400 words per minute.

One can continue to speed up the display rate by making the tab expansion modifier smaller, until the display capabilities of the terminal are exceeded.

Note that if the baud rate is less than 9600, the speed of word display is proportionately less. But even when using this scheme with a low baud rate terminal respectable speeds can be achieved by using lower values for the tab expansion modifier.

Finally, if the flashing cursor is a distraction, a cardboard or tape mask could be fashioned to cover the portion of the video screen where this appears.

David Ecklein
Tesseract Associates
Stinson Lake Road
Rumney, NH 03266

## Datalyzer Users Unite!

Dear *DDJ*,

A couple of years ago, I purchased a S-100 logic analyzer called "Datalyzer." It was made by Databyte of Middleton, WI, who went out of business about 6 months later.

I have been trying to modify the Datalyzer's software to work under CP/M with a couple of extra commands added, but have only had limited success. Could you please publish this letter with the hope that those who read it and have a Datalyzer might drop me a line about their experience? I'd be glad to compile a Datalyzer user group list if I get enough responses.

Bob Marsolais
16121 Atglen St.
Hacienda Hts., CA 91745

## Santa Barbara Tiny Basic

*(continued from page 23)*

BASIC. The statement with the error is printed out with a questin mark inserted at the point where the error is detected.

(1) WHAT? means it does not understand you. Example:

```
WHAT?
210 P?TINT "THIS"          where PRINT is mistyped
WHAT?
260 LET A=B+3, C=(3+4?, X=4
```

(2) HOW? means it understands you but does not know how to do it.

```
HOW?
310 LET A=B*C?+2           where B*C is greater than 32767
HOW?
380 GOT 412?               where 412 does not exist
```

(3) SORRY means it understands you and knows how to do it but there is not enough memory to do it.

**Error Corrections.** If you notice an error in typing before you hit the CR, you can delete the last character by the Rub-Out key or delete the entire line by the Alt-Mode key. Tiny BASIC will echo a back-slash for each Rub-Out. Echo for Alt-Mode consists of a LF, a CR, and an up-arrow.

To correct a statement, you can retype the statement number and the correct commands. Tiny BASIC will replace the old statement with the new one.

To delete a statement, type the statement number and a CR only.

Verify the corrections by "LIST nnnn" and hit the Control-C key while the line is being printed.

■■

**Engineers**
**Scientists**
**Systems Programmers**
**Business Owners**
All these read *Dr. Dobb's Journal.*
Reach them with cost-effective advertising.
(415) 323-3111.

$2.50 ($3.00 In Canada)

# DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

**FOR USERS OF SMALL COMPUTER SYSTEMS**
Number 56 June 1981 Volume 6, Issue 6

**A Cross-Assembler for the 8080/Z80**
*by Jonathan Mills*

**Comal-80: A New Language?**
*by Susan Sonderstrup and Mogens Pelle*

**A Tiny File System**
*by William A. Gale*

**Re-Thinking Program Editors**
*by William B. Brogden*

**H8 Text Editor to CP/M File Conversion**
*by Jim Scott*

**. . . . and even more than before!**



P.O. Box E
Menlo Park, CA
94025

# DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Volume 6, Issue 6
Number 56
June 1981

# Contents

## Articles

**Cross Assembler for the 8080 or Z80** ..... 9
*by Jonathan Mills*

**Comal-80, A New Language?** ..... 14
*by Susan Sonderstrup, Mogens Pelle*

**H8 Text Editor to CP/M File Conversion** ..... 18
*by Jim Scott*

**A Tiny File System** ..... 20
*by William A. Gale*

**Installing a Printer Toggle in North Star DOS 5.2** ..... 50
*by Bert Smart*

*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025.

© 1981 by People's Computer Company unless otherwise noted on specific articles. All rights reserved.

## Departments

Letters ..... 4
Guest Essay ..... 16
Book Reviews ..... 48
Dr. Dobb's Clinic ..... 52
Of Interest ..... 56

**Publisher**
Michael Madaj
**Editor**
Marlin Ouverson
**Contributing Editors**
Dave Caulkins
Dave Cortesi
**Business Manager**
Maureen Christine
**Art Director**
Clifford West
**Artist**
Barbara Ruzgerian
**Typesetters**
Renny Wiggins
Michelle Moonstone
**Proofreader**
Nancy Heubach
**Marketing Director**
Craig S. Harper
**Advertising Sales**
Kent Safford
**Circulation Manager**
Peter Clark
**Circulation Assistant**
Leah Dansby

# Editorial

The 6th West Coast Computer Faire is over (sigh of relief and aching feet). Director Jim Warren's roller skates have been hung up for another year, or at least until his rumoured mini-faires get under way. This year's was even bigger than expected, as the thirty thousand nametags were exhausted even before the exhibitors.

Perhaps the biggest surprise to commercial concerns was the profile of the "average" attendee. All the sales people had been coached well in advance on how to spill forth a technical-sounding pitch; none seemed able to answer basic questions like, "Why should I have a computer?" "What can I use it for?" "I don't know anything about computers – where do I start?" Would-be consumers wandered from booth to booth in search of someone who could still remember how to speak their lingo and give a down-to-earth reply. They filled halls designated for "introductory" sessions, and then expressed frustration when they couldn't understand the speakers.

Of the hundreds of booths, only a small handful really stood out in terms of newsworthiness. The star of the show was the Osborne 1 computer, which *DDJ* will be reviewing in an upcoming issue. Its dual floppies, 5" monitor and full-size keyboard comes in one case which will fit under an airplane seat – but the real breakthrough is the price. It lists for $1795 including CP/M, Wordstar, Supercalc, CBASIC, and MBASIC.

Less of a crowdstopper, but nonetheless important, was the Stratos by Symbiotic Systems, Inc. Two things stand out that set it apart from other CP/M compatible systems. First, a clock/calendar is not only integrated into the hardware, but into the CP/M system as well, allowing REMINDER to call and execute any program at the specified, appropriate time. Second, the BIOS (vendor-written hardware support code) makes use of bank-switched memory to do full-track disk buffering, which has to make a significant performance improvement.

Two other booths at the Faire should be noted as among the Most Popular. Epson printers sold out everywhere, receiving general, deserved acceptance as a good quality dot matrix printer at a reasonable price. The Sinclair booth was also well populated. All those computer consumers were there, wanting to know more about this less-than-$200 computer and what it can do for them.

If this year's Faire is a valid indicator, and I believe it is, then all the prophecies of the original homebraumeisters, of the *ComputerTown, USA!* folks and of others involved in computer literacy, are coming true. It appears the circle of microcomputerists is opening outward to include not only technicians and hobbyists, but result-oriented application-minded laymen as well.



Marlin Ouverson
Editor

# Letters

## Mincemeat And Minced Words

Dear Mr. Ouverson,

Sigh. I apologize to all readers who will advise me to get a copy of Harbrace or Turabian and read it. I have copies of them already, and I have read them, but I just didn't take too much time when I wrote the article. I don't usually nest parentheses unless I'm writing LISP code.

The most glaring errors are in the description of the command set for MINCE. I made two big blunders, one in the inclusion of some EMACS commands that are not existent in MINCE, and in another place an entire sentence is missing, and the one that should follow it is simply wrong.

My keyboard does not generate Meta characters. Mark of the Unicorn's does. It is a custom one off design. That's a Mode Line at the bottom of the screen, not a Model Line. C-X K prompts you for a buffer name, and does not kill the current one by default.

There are now plans in the wings for a user's group of some sort for Amethyst. I hear that there is already a C mode floating about someplace. I will advise *DDJ* as soon as something materializes.

I apologize to anyone who feels that I severely maligned his editor or text processor. Send me a copy and I will review it too, and will try to be as objective as I can. It's hard for me to be objective when I love a product as much as I do MINCE.

Yours,
Barry A. Dobyns
1635 Royal Crest #1128
Austin, TX 78741

## FCT NMC

To the Editor:

The modifications to the CP/M DUMP utility proposed by Tom Cage (*DDJ* #53) kept me busy a whole Sunday. While his algorithm was fairly simple, implementation with such fictional mnemonics as DCR LX, DCR HX, MVI LX,data, or MVI HX,data was naturally impossible.

One possible alternative is figure 1 where [*] is used in the comment field to indicate new lines of source code to insert in the original DUMP.ASM file.

Since most DUMPs are of .COM files, I also suggest that the LXI H,0 line in OPENOK: be changed to LXI H,0100H. This way the DUMP listing shows the actual addresses for each line.

And finally, for those whose DUMP fails to display a sign-on line even though it is included in the .ASM source file, add the following code immediately following the File Control Block definitions:

```
LXI   D,SIGNON;[*]
CALL  ERR      ;[*]
CALL  CRLF     ;[*]
```

The net result of these changes will be figure 2.

Sincerely,
Robert G. Savage
98-719 Iho Place #1101
Aiea, Hawaii 96701

**Figure 1.**

```
...
OPENOK:   MVI     A,80H
          STA     IBP          ;set buffer pointer to 80H
          LXI     B,204H       ;[*] set up byte counters
...
GLOOP:    PUSH    H            ;save line position
...
          JC      FINIS        ;carry set by GNB if end file
          PUSH    B            ;[*] save byte counters
...
          CALL    PHEX
          MVI     A,' '        ;[*] dbl space after line number
          CALL    PCHAR        ;[*]
...
NONUM:    INX     H            ;to next line number
...
          POP     B            ;[*] restore byte counters
          DCR     C            ;[*] decrement 4th byte counter
          JNZ     GLOOP        ;[*]
          MVI     A,' '        ;[*] dbl space after bytes 4 & 12
          CALL    PCHAR        ;[*]
          MVI     C,4          ;[*] restore 4th byte
          DCR     B            ;[*] decrement 8th byte counter
          JNZ     GLOOP        ;[*]
          MVI     A,' '        ;[*] triple space after 8th byte
          CALL    PCHAR        ;[*]
          MVI     B,2          ;[*] restore 8th byte
...
```

**Figure 2.**

```
A>FILEDUMP TEST.123
FILEDUMP VERSION 1.5

0100  00 01 02 03  00 01 02 03  00 01 02 03  00 01 02 03
0110  00 01 02 03  00 01 02 03  00 01 02 03  00 01 02 03
0120  00 01 02 03  00 01 02 03  00 01 02 03  00 01 02 03
0130  00 01 02 03  00 01 02 03  00 01 02 03  00 01 02 03
0140  00 01 02 03  00 01 02 03  00 01 02 03  00 01 02 03
0150  00 01 02 03  00 01 02 03  00 01 02 03  00 01 02 03
0160  00 01 02 03  00 01 02 03  00 01 02 03  00 01 02 03
0170  00 01 02 03  00 01 02 03  00 01 02 03  00 01 02 03

A>
```

## Brace Yourself

Dear *DDJ*,

I first read your magazine in my orthodontist's office in Singapore. I don't wear braces any more.

Brad Rubenstein
2415 Dwight Way #4
Berkeley, CA 94704

## Small-c Compiler

Dear *DDJ*,

This note is to report two bug fixes and an enhancement to the small-c compiler. The first fix has to do wtih the way small-c handles comments in the form:

```
/*
** comment
*/
```

The problem is that the comment terminator "*/" being at the beginning of the line is not recognized because the function "inchar" (used to skip over comment characters) always calls "gch" after acquiring a new line. "Gch" bumps "lptr" causing it to point to the second character. Since "inchar" is only used by "preprocess" while skipping over comments and only one of its calls requires the full features of "inchar." I chose to delete "inchar" and modify the part of "preprocess" which handles comments as indicated below:

```
else if((ch()=='/')&(nch()=='*'))
 [lptr=lptr+2;
 while(((ch()=='*')&(nch()=='/'))==0)
  [if(ch()==0) inline(); else lptr++;
  if(eof)break;
  ]
 lptr=lptr+2;
 ]
```

The second bug is a problem in P. L. Woods' patch reported in *DDJ* #52. His fix will not handle the following code properly:

```
func(a, b, c) int b, a; int c; [
```

The problem is that the arguments are typed in two statements rather than one. The culprit is the location of the statement "argtop=argstk;" in "getarg." That statement should be moved from "getarg" to "newfunc" just before the statement "while(argstk)...." Doing that involves deleting the "argtop" declaration in "getarg" and declaring it as an external variable at the beginning of the program.

Then enhancement implements the backslash escape sequences for character and string literals. I find that these are very handy. This is easy to install by defining a new function "litchar" in figure 3.

**Figure 3.**

```
/* return current literal char & bump lptr */
litchar() [
   int i, oct;
   if((ch()!=92)¦ (nch()==0)) return gch();
   gch();
   if(ch()=='n') [++lptr; return 13;] /* CR */
   if(ch()=='t') [++lptr; return  9;] /* HT */
   if(ch()=='b') [++lptr; return  8;] /* BS */
   if(ch()=='f') [++lptr; return 12;] /* FF */
   i=3; oct=0;
   while(((i--)>0)&(ch()>='0')&(ch()<='7')) oct=(oct<<3)+gch()-'0';
   if(i==2) return gch(); else return oct;
   ]
```

The statement "litq[litptr++]=gch();" in "qstr" should be changed to "litq[litptr++] =litchar();". The function "pstr" should be modified as indicated below:

```
pstr(val)
   int val[];
   [int k;
   k=0
   if (match("'")==0) return 0;
   while(ch()!=39)
      k=(k&255)*256 + (litchar()&127);
   ++lptr;
   val[0]=k;
   return 1;
   ]
```

Finally "preprocess" must be modified to permit \" and \' sequences. Change "while(ch()!='"')" to "while ((ch()!='"')¦ ((line[lptr-1]==92)&(line [lptr-2] !=92)))" and change "while (ch()!=39)" to "while((ch()!=39)¦ ((line [lptr-1] ==92)&(line[lptr-2]!=92)))".

I hope this information will be helpful to the small-c enthusiasts among your readers.

Sincerely,
J. E. Hendrix
Rt. 1, Box 74-B-1
Oxford, MS 38655

## Not Complaining?

Dear Mr. Ouverson,

Your new "Ex Machina" column is a long awaited, sorely needed and very welcome addition to *Dr. Dobb's*. It's high time someone stepped in to protect us, the home computer hobbyists, from the high-handed and (sometimes) dishonest treatment of manufacturers and vendors.

A case in point is my Heathkit H-19 terminal. This terminal has a front panel cutout which accepts a 5.25" disk drive. (The cabinet is the same as in the H-89 "all-in-one" computer.) All that is needed for the installation are a few bucks' worth of hardware parts: sheet metal brackets and the like. Imagine my surprise when the local Heath Center could not provide the part numbers of the required components.... Imagine my further surprise when, over a call to the factory, they refused to sell me the parts!

Why does Heath behave like that? I do not know. Perhaps they simply can't be bothered with little me. Even with their inflated price structure, the parts could not be more than $20.

This is not a complaint. Heath is free to sell or not to sell spare parts. I feel, however, that anyone contemplating a Heath product should be aware of the quality of their so-called service. I have told this story to anyone who would listen, and have been successful in preventing the purchase of a printer. Not much, I agree, but I feel I have done my little something to improve this world....

Sincerely yours,
R. A. Hoffman
9 Greenway Court
Brookline, MA 02146

## A Poke at MIT BASIC

Dr. Dobbs:

I, too, ran into the problem of not being able to PEEK or POKE above 32767 (7FFFH) in the old MITS BASICs, 8K 3.1 and 8K 3.2.

May I offer a much more simple patch which, however, doesn't allow

using positive numbers as does T. J. Van Iderstine's patch in volume 52?

8K 3.1 addr 646 was "JM FCERR" change to NOP NOP NOP

8K 3.2 addr 649 was "JM FCERR" change to NOP NOP NOP

To compute the proper address, use x-65536, or just directly compute it: A VDM or VTI or VIO (etc.) at F000H is at address -4096. Just say VB=-4096, then forget negative numbers, just referencing VB+nn for your poke/peek addresses.

Ward Christensen
Dolton, IL

## Coming to Terms

Dear *Doctor Dobb's Journal,*

I read with interest the letter from reader Langdon Procter of Pasadena, CA. After giving the matter some thought I think that Mr. Procter and myself can agree on the following points on the Sinclair ZX80.

The ZX80 has educational value for everyone. It is an engineering feat both in hardware and software. No doubt it is an excellent choice for someone who has had no prior training. And even seasoned professionals wonder at the power of this little machine.

The video quality of the ZX80 is satisfactory for its purpose. You need a good quality Monochrome Television set of recent design. There are some TV sets that do not perform good with any VHF video computer.

Cassette tapes made on any ZX80 should play back on any other ZX80. For best results both cassette machines must be in good repair. A few cassette machines may not work if poor cassette tape is used.

If you wish to expand your ZX80, use only the items recommended by Sinclair. Do not try to modify the ZX80. The printed circuit card was not built for user experimentation.

I do not wish to go into any more detail about a few other topics mentioned by Mr. Proctor in his letter. Instead I would like to draw attention to a very important point that has not yet been mentioned.

Many vendors of new items expect the consumer to inform them if any problems appear after the consumer has used the product for a time. If we consumers will write or call either the vendor or maker we may expect help. We should not try to repair new equipment ourselves. Neither should we suppose a product is perfect only because it is new.

Any owners of the ZX80 who are having trouble with the video display, cassette interface, power supply or any other feature should not suppose that something must be wrong with the TV set, the cassette recorder or the electric company. Read the manual over again. If that does not help, then contact Sinclair Research. Their address is given as:

Sinclair Research
50 Stanford Street
Boston, MA 02114

If your ZX80 is healthy and happy, but you wish to learn more about it you should also consider a subscription to SYNC. Their address is:

SYNC
39 East Hanover Avenue
Morris Plains, NJ 07950

I hope this letter will be helpful to future ZX80 owners, or any present owners who may have had some problems with this very delightful and very, very small personal computer.

John B. Palmer
PO Box 23
Boonville, CA 95415

■■

Why wait
to start
increasing your sales?
Call now
to reserve
advertising space
in the next
*Dr. Dobb's Journal*
(415) 323-3111

**YOU CAN HELP YOUR COMMUNITY LEARN ABOUT COMPUTERS**

**SUPPORT**

ComputerTown™

*ComputerTown*, a computer literacy project sponsored by People's Computer Company and later funded by the National Science Foundation, began operations in 1979. The project's ongoing goal is to design, publish and disseminate materials for the development of community-based computer literacy projects around the world.

**You too can help!** Your tax-deductible donations of new and used computer hardware and software will enable us to bring "hands-on" experience to people everywhere. These items can mean the difference between a person just hearing about computer technology and actually being able to sit down and use it. A *ComputerTown* can be set up in a public library, museum, school, business or a number of other places. What's needed, along with the information, resources and experience we provide, is just a few machines and some software to get started.

Already there are more than eighty *ComputerTowns* in the U.S., Canada, England and other countries. Your generous donations can help this number grow.

For more information about *ComputerTown*, and for details on how to make contributions, contact:

Fritzi Lareau, c/o *ComputerTown*
1263 El Camino Real, P.O. Box E, Menlo Park, CA 94025
**(415) 323-3111**

# The Z88 Project
# Cross Assembler for 8080 or Z80

In this article, which may be the first in a series, you will find a major piece of software and a new central processing unit (CPU) card described. The first installment of the software source listing is offered.

Although most magazines publish software tid-bits, large software packages are seldom seen. Exceptions exist. *Dr. Dobb's* several version of Tiny BASIC are notable.

This condition exists for several reasons. First, relatively few hobbyists write complex systems software. Most computer enthusiasts today expect to find the systems software as a reliable part of the package. Next, those who *do* write this type of software intend to sell it. I am no stranger to these motives, but have another one, too. Some form of low-cost development tool is needed to get the 16-bit hobbyist market off the ground. I hope that XASM-86 will become that tool. Finally, large complex software tends to be difficult to design and debug. A commitment to completion and unflagging enthusiasm are *sine qua non* software!

XASM-86 bridges the gap between the popular 8080/Z80 and the powerful new 8086. XASM-86 will run on an 8080 or Z80 CPU, but since it is a cross-assembler, it will generate object code for any 8086 or 8088 processor. XASM-86 is fast, efficient, and innovative. Techniques used in compiler design have been applied to create a reliable, state-of-the-art program. The source listing is fully documented. It requires over 500,000 bytes on disk, and lists on almost 200 pages.

XASM-86 was a natural outgrowth of the Z-88 project begun in 1979. Yet unnamed, the project was a response to the spate of announcements of 16-bit microprocessors that occurred in 1978 through 1979. Examination of these machines led me to suspect that the 16-bit macroprocessor market would be a new one, or at least a slowly developing one for hobbyist retrofits. Then Intel announced a clever product, the 8088. Essentially an 8086 with an 8-bit bus interface unit, the 8088 looked like an excellent product for an S-100 (IEEE-696) CPU.

Thus in September 1971, the Z-88 project began. It had one objective:

> "Upgrade existing S-100 computer systems to 16-bit capability without causing hardware obsolescence, or hardware or software incompatibilities."

The target device to meet this objective was a special CPU. There were seven requirements that it had to meet. They were:

1. Meet the (then preliminary) IEEE-696 S-100 bus standard,
2. Be selectable for non-IEEE-696 S-100 buses (Altair or IMSAI),
3. Maintain compatibility with 8-bit peripherals in use on the bus selected,
4. Develop all timing using logical combinations of existing signals instead of monostable pulse generators (one-shots),
5. Include provision for multi-tasking, i.e, priority vectored interrupts,
6. Accept all 8080/8085/Z80 software, with the exception of that software using RIM or SIM instructions on the 8085, or that used software to generate highly critical timing loops that could not be modified,
7. Provide for an Altair or IMSAI front panel, and use as many of the front panel control functions for each processor as possible.

As became obvious, this special CPU used two processors: a Z80A and an 8088. A 8259A priority interrupt controller was added as the heart of a circuit that generated Z80 mode 2 or 8088 interrupt vectors, based on the active processor. A special control circuit was developed to allow control to be handed back and forth between processors in any of several ways. (See Figure 1.)

On November 30, 1979, the Z-88 passed its final tests, and was put into action as the heart of the Z-88 project's development system. It has since completed over 1,500 hours of operation without error.

Attempts were made to market the Z-88, but met with lukewarm results. In one instance, perhaps the most surprising, a case of parallel development was uncovered. Mark Garetz, working for Bill Godbout, had independently come to the same conclusions, but selected an 8085/8088 processor combination. Godbout is marketing this CPU with, I hope, great success. A press release on the Z-88 directed to *BYTE* magazine popped up in their August 1980 issue. This caused a deluge of several thousand requests for information. To those of you who did write, please contact Brent Olsen of SSM (formerly Solid State Music), and tell *him* how much you want a Z-88. Brent came closest to closing a manufacturing agreement with us. He is a great guy, and a hard-headed realist.

However, no matter what a great CPU the Z-88 was, it suffered from a simple problem. To quote an unnamed marketing vice-president, "Who cares what kind of a CPU is in there? It could be a chipmunk, as long as it works." This inspired the observation on my part, that a 16-bit CPU with no software was a dead chipmunk. Phase two of the Z-88 project was to begin.

Phase two, later named XASM-86, had as its objective to, "Design, program, and debug a software tool for the 8086." After deciding on a cross-assembler, which suited the Z-88 CPU well, nine requirements were established for it. They were:

1. Run on 8080, 8085, or Z80 CPUs and produce 8086/8088 object code,
2. Operate on disk files to produce disk files (disk-to-disk),
3. Use consistent and rational syntax and mnemonics for 8086/8088 source code (not necessarily matching Intel's),
4. Encourage documentation and structured design practices,
5. Attempt to be independent of any operating system,
6. Assemble mnemonics for AMD 9511 Arithmetic Processor (this was decided before the 8087 coprocessor was announced),
7. Use a modular design to allow maintainability and reliability,
8. Provide for future enhancements, and
9. Provide fast cross-assembly.

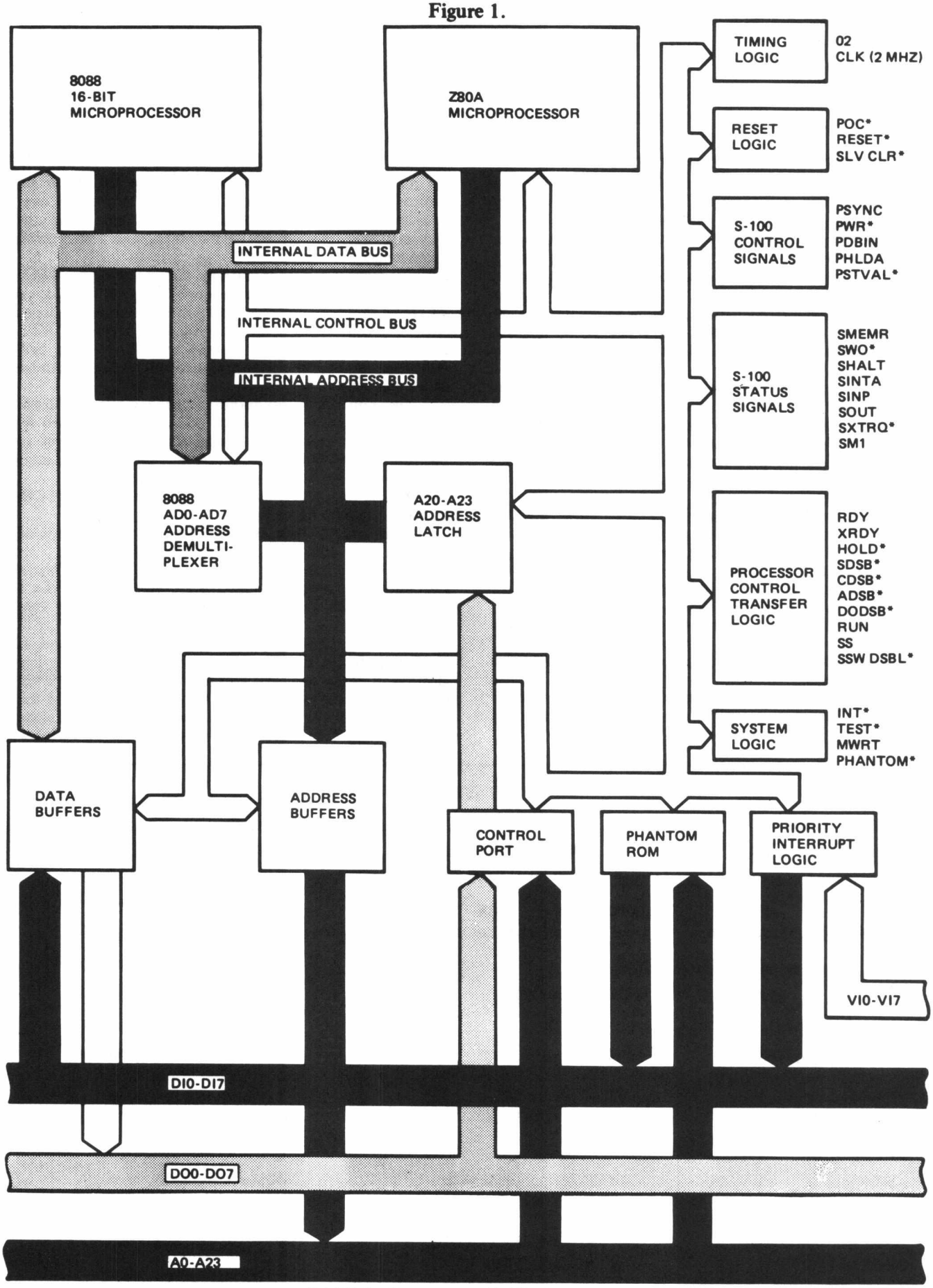
The first version of XASM-86 met these goals by cross-assembling 300 lines of 8086 source mnemonics to 8086 object code, disk-to-disk, in 55 seconds. The system is, of course, limited by the speed of the floppy disk drive.


## by Jonathan Mills

*Jonathan W. Mills, President of Programmers Publishing Co., Box 2571, Kalamazoo, MI 49003*


© 1981 by Programmers Publishing Company

**Figure 1.**




© 1981 by Programmers Publishing Company

Some of the exciting features of XASM-86 will be described below. The mnemonic set and syntax will be examined, and the subroutine library will be outlined.

XASM-86 is approximately 17,000 bytes long, excluding the symbol table. It is written in structured assembly language accompanied by extensive internal documentation. Two utility programs are provided with XASM-86: a relocating loader, and a symbol table compiler. It is possible to reconfigure the symbol table to allow XASM-86 to produce 6800, 6502, Z80 or other machine's object code. It is not a project for the beginner, though.

High-level, application-oriented input-output routines are a part of XASM-86. These routines should make it easy to "hook" the program into any operating system. Since all XASM-86 input-output (I/O) is done by transferring records between devices, all the user need do is change those routines or addresses of routines that read or write a character from or to a specific device.

XASM-86 mnemonics are loosely based on those provided by Intel for the 8086, but are structured to resemble those of the DEC LSI-11 in their syntax. For example, to move a byte from memory to the BH register:

Intel mnemonics: MOV BH,LOCATION
XASM-86 mnemonics: MOVB LOCATION,BH

XASM-86 specifies in the opcode whether or not a byte is the operand, and also (finally) uses a logical left-to-right orientation for source-to-destination ordering of the operands. This is consistent, always.

XASM-86 addressing modes are also similar in syntax to the LSI-11. The 25 possible 16-bit source/destination operands are in Table 1.

As may be noted, the variable names in XASM-86 are not limited to six letters in length. In fact, they may be over 200 characters long, although only 15 will be printed on a listing. The rest are stored in the symbol table, and differentiate the more lengthy names selected.

Fourteen pseudo-operations are included in XASM-86. One of them, the "ditto" operator (") serves to reduce repeated use of the same pseudo-operation to a simple double quote. Since the psuedo-operations are all descriptive (i.e., BYTE instead of DB, ALLOCATE instead of DS), the ditto operator increases programmer effectiveness and program readability at the same time.

Two kinds of macro definitions are included in XASM-86. The first is the simple macro that is expanded into several XASM-86 mnemonics. The second is of some interest. Invoking the OPMACRO pseudo-operation associates a name with an operand expression. An example would be:

```
ACCOUNTBALANCE OPMACRO ACCOUNTOFFSET@BX
```

In use, a program could say:

```
LABEL: ADD CX,ACCOUNTBALANCE  instead of
       ADD CX,ACCOUNTOFFSET@BX.
```

Listing I on page 22 shows the XASM-86 mnemonics, operands, and pseudo-operations. It also includes the AMD 9511 Arithmetic Processor mnemonics assembled by this program.

Borrowing from compiler design, XASM-86 uses extensive table-driven routines to accomplish expression parsing, expression evaluation, and to implement an internal interpreter which reduces each type of assembly for any 8086 instruction to a single function. These innovative methods allow for speed and flexibility. An added benefit is the ease with which

**Table 1.**

| **16-bit Registers:** | | | | | | | | **Example:** |
|---|---|---|---|---|---|---|---|---|
| AX | CX | DX | BX | SP | BP | SI | DI | MOV AX,SP |
| **Indirect:** | | | | | | | | |
| @SI | | @DI | | 00000H@BP * | | @BX | | MOV AX,@DI |
| **Direct (Effective Address equals Offset):** | | | | | | | | |
| 03EF2H | | LABEL | | DISPLACEMENT | | | | MOV AX,RESULT |
| **Based:** | | | | | | | | |
| DISPLACEMENT@BX | | | | DISPLACEMENT@BP | | | | MOV AX,EMPLOYEENO@BX |
| **Indexed:** | | | | | | | | |
| DISPLACEMENT@SI | | | | DISPLACEMENT@DI | | | | MOV AX,STARTARRAY@SI |
| **Base Indexed:** | | | | | | | | |
| @BX+SI | | @BX+DI | | @BP+SI | | @BP+DI | | MOV AX,@BX+SI |
| **Base Indexed Displaced:** | | | | | | | | |
| DISPLACEMENT@BX+SI | | | | DISPLACEMENT@BX+DI | | | | MOV AX,PARAMETERS@BX+SI |
| DISPLACEMENT@BP+SI | | | | DISPLACEMENT@BP+DI | | | | |

* The @BP operand must be given an offset of 0 to avoid generating the opcode for effective address equal to the offset alone.

a fix may be made to any incorrectly assembled instruction: simply change the assembly function!

At the heart of the process is a symbol table based on the binary tree organization. Although not as fast as hashing (generating a unique code and thus an address based on some function of the symbol's text, the binary tree is simple to implement, and has the added benefit of producing ordered output on completion by simply traversing it.

A subroutine library is used to keep the general purpose routines that comprise or are used by the various XASM-86 modules. Extensive documentation allows new modules to be written in terms of these "primitives" in a fairly short time. Figure 2 lists the routines in the library.

In closing the description of XASM-86, I would like to say that it is a virtually "bullet-proof" piece of software. Over 100 error messages, stored internally as text, signal conditions that range from I/O errors to "OPCODE IS INVALID OR INCORRECTLY SPELLED". In fact, there are even messages that signify that XASM-86 has reached a "should-never-get-here" part of the code! Even on the first run, it was possible to debug XASM-86 by observing the error messages it produced. Only once did it crash the system – and that was due to my failure to include a necessary module. Listing II includes the current version's error messages.

Finally, for all the hackers who will write in to tell me how this or that portion of a module or routine could be re-

## Figure 2.

**Routines included in the XASM-86 subroutine library. Some of these routines are listed in this installment in Listing 3.**

XASM-86 Utility Routines

Logical Input/Output (Record-oriented I/O)

- Transfer logical record
- Create logical file
- Delete logical file
- Open logical file
- Close logical file

Binary Tree Manipulation (Symbol Table Utilities)

- Search tree for record
- Insert record at root
- Insert record at left child
- Insert record at right child
- Sort records in tree (pre-order traversal)

Symbol Table Record Manipulation

- Set record index to symbol (count,type)
- Set record index to value (count,type)
- Set record index to left child pointer
- Set record index to right child pointer
- Set record index to symbol text
- Set record index to value

Text Manipulation

- Find field of variable length
- Move field of variable length
- Move word (2-byte field)
- Compare field of variable length
- Fill field with blanks (ASCII 20H)
- Fill field with 8086 NOPs (ASCII 90H)
- Fill field with any character

Program Execution Control

- Case of condition code (JMPXI)
- Execute indirect subroutine (CALLI)

Data Conversion

- Convert ASCII field into binary (hexadecimal interpretation)
- Convert ASCII field into binary (decimal interpretation)
- Convert binary field into ASCII (hexidecimal interpretation)

Parameter Manipulation

- Extract nibble from byte parameter
- Load logical I/O parameters
- Save operand parameters
- Load operand parameters

Table and Array Manipulation

- Calculate entry pointer = Table base address + ( index * items per entry)

Arithmetic

- Subtract two words in CPU registers
- Compare two words in CPU registers
- Set zero and carry flags based on word add or subtract

Physical Input/Output (Character-oriented I/O)

| | |
|---|---|
| Read | (disk, modem, terminal, memory) |
| Write | (disk, modem, terminal, memory, printer) |
| Open | (disk, modem, terminal, memory, printer) |
| Close | (disk, modem, terminal, memory, printer) |
| Create | (disk, modem, terminal, memory, printer) |
| Delete | (disk, modem, terminal, memory, printer) |

NOTE: Some operations may be a "no-operation" for a specific device, but are listed to preserve orthagonality of the application I/O system.

written to be:

A. faster, or
B. shorter, or
C. cryptically clever,

I would like to say that the rules of my design process are totally at odds with those of the person who counts bytes and clock cycles. If it pleases you to do so, go ahead. That might be a contributing factor to a possible dearth of software productivity on your part.

Harsh words, to be sure, but they are in keeping with one unyielding fact:

"Reliable hardware is cheap and easy to design,
reliable software is expensive and hard to design."

The Z-88 CPU was complete in three months, but the software took over four times as long. XASM-86 is *still* being improved and extended! Of course, there are a host of corollaries to this one fact. Here are four:

1. Spend time on program optimization only when necessary.

If the program is limited by the speed of a disk interface, it is stupid to spend three days to make a table look-up routine 10 microseconds faster.

2. Optimize only when you know *where* to do so.

So the program is limited by the disk? Then work on the I/O routines. Try to decrease the number of bytes in disk records. Optimize where it will do some good.

3. Clarity is more important than cryptic optimization.

If you can't understand the code, you can't fix it, revise it, upgrade it, add to it, or modify it . . . so why bother to write it? If you *do* have extra time, spend it on better documentation, not on expunging a clock cycle.

4. Don't tempt Murphy.

Basically, if it works, leave it alone. This is the hardest one for me to apply. However, if you think you have found a better way to do something, write it down and describe all the particulars. Let it set for at least 24 hours before you change anything. Then, look at the change. Reconsider it. Can you detect any unwanted side-effects? No? Okay, go ahead and try it. Run it in parallel with your old version for a while. Does the new model work after tests? After daily use? Then you have made a successful change! But, if it does not, return to the original (you did save it, didn't you?) and go back to the drawing board with the modification.

These rules, among others, have fostered the relatively rapid development of XASM-86. If the design had not been made in a structured fashion, 50% of the development time could have been expected to be spent on testing and debugging. This would correspond to a matching 800 hours for the time used to analyze, document, design and code XASM-86. Instead, only 60 hours were spent debugging XASM-86. That is about 90 percent decrease from the number of hours expected! Thus, XASM-86 is presented to you in 1981 instead of 1982. It may not be optimized, it may be somewhat redundant internally, but it is fast, it works, and it may be foolproof (depending on the fool).

I hope you enjoy it. If you would like to see the entire package here, in installments, write to the editor. ■■

**LISTING ON PAGE 22**

*JONATHAN MILLS is a renaissance man who completed his B.A. in Latin, chemistry, and mathematics. He has at various times been an officer in the U.S. Army, artist, publisher, DP manager of an IBM shop, and is currently wearing two hats, one as an instructor of data processing at the local community college, the other as president of Programmers Publishing Company.*

*Rumor has it that the prototype Z-88 was developed with only two tools: a volt-ohm meter and Jonathan's grey matter; and that it checked out afterward on an oscilliscope well within specifications.*

*Mr. Mills lives and works in Kalamazoo, Michigan, camouflaging a 128,000 byte Z-88 development system amidst the colorful orchids in his den.*

*DDJ's* responsive readership
can give you invaluable feedback
on your ideas and innovations.
Find out how you can reach them
by contacting us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# COMAL~80
# A New Language?

by Susan Sonderstrup
and Mogens Pelle

Over the years, various magazines have published articles on the advantages and disadvantages of different languages.

BASIC has been the subject of many articles indicating its limitations, and pointing out that even a simple program seems rather difficult to read and understand.

The articles often conclude by expressing the need for a language which makes it natural to build up a program of smaller, logically linked parts, which can be called one by one in an orderly way.

One such language is Pascal, frequently mentioned in contrast to BASIC, and just as often described in emotional terms.

The silent war between Pascal and BASIC fans will probably never find its end, as some believe that Pascal does not have proper editing facilities (due to the construction as a compiler) and is too difficult to learn.

At this point it will be suitable to add that most persons taking part in the continuing discussions are EDP teachers showing an ever increasing interest for Pascal to match their higher levels.

Even before this war was started, an attempt to build a bridge took its start in Denmark. The teachers expressed their dissatisfaction with the possibilities at hand, and as constuctive criticism always leads forward, a new language was implemented: COMAL-80.

This language has its family connections to both Pascal and BASIC, as it is believed to contain the best parts from both.

As mentioned above, the demand came from the educational sector asking for a better language, which is the reason for COMAL-80 having built-in programming support, and by its structure, facilitates a possible transition to more scientific languages. However, this does not mean that COMAL-80 is for education only, as it has proved valuable for administrative as well as controlling jobs, too.

It takes an extended BASIC and a good Pascal. Stir well and you get COMAL-80. At least, that is the way it may look to the users, but it certainly takes a lot more to take up the competition with established languages, benefiting from many years of adjustments to fit the market.

Rather, it takes a lot of planning, hard work, and precision to avoid loose ends to implement the language, meeting the high expectations shown in the preliminary specifications.

That the results far exceeded the expectations is proven by the fantastic response received from the more than 100 test-installations having been in use for about one year, before the current lead on the educational market in this country. The good results have spread the knowledge of COMAL-80 across the borders, and what seemed to be pioneer work within Denmark has led to many a plan for other high-level languages.

## A Bit of History

The idea behind this language is not new. It started back in 1973 when Mr. Borge Christensen, assistant professor at the teachers college in Tondern, achieved daily access to a computer equipped with a BASIC – at the time apparently a good one.

Pretty soon he found several problems in running this BASIC. First of all, the students often had problems reading programs they had written themselves some time earlier. Among the reasons, Mr. Christensen found the awkward and unnecessary limitations for variable names, which was quite normal for BASICs at that time. Also, he found that in BASIC it is extremely difficult to write structured programs.

Mr. Christensen not only had the talent, but also the touch of luck needed to make the ideas come true. Acquiring the source-code was one thing, but having two excellent people in the area, Per Christiansen and Knud Christensen, who were able to improve the code, was just what he needed to get rid of the most obvious weaknesses.

The result was instant – students found it a lot easier to understand and work with this improved BASIC, and in the years following it was distributed to colleges and high schools teaching EDP.

In 1978 low-cost microcomputers really entered the market and EDP spread into grammar schools. However, the micros all had BASIC – and now the teachers had the knowledge of the *improved* BASIC.

The demand for better language on micros became so heavy that the implementation of a new language took its start in the summer of 1978. The first version was released for use on the COMET, marketed by International Computers Limited in March 1979.

Through the spring and summer this version was tested by more than 100 teachers and their students, whose comments and suggestions mostly turned into implemented improvements. Most of this work was done by Arne Christensen, a student of the Copenhagen University for Data Science, in close cooperation with Borge Christensen. (At this point it is worth mentioning that despite the similarity of their last names, the four gents are in no way related.) The product resulting from this work is marketed under the trade name of COMAL-80.

## But What Is The Difference?

As described above, COMAL-80 appears as an improvement of BASIC. The most important aspect is that it works as an interpreter. Among the many important reasons for this, you will find that program errors can be dealt with very efficiently.

When a line is finished and the return-key is pressed, the error-message appears instantly. To continue the proprogram, the error must be corrected. The cursor indicates the error-point and in many cases COMAL-80 suggests what needs to be done.

An example to this important feature is found in figure 1.

From BASIC another feature is inherited: the line numbers. It is incorporated in COMAL-80 because the basic idea is to be able to run any BASIC program, too.

It may seem strange to incorporate a full extended BASIC in a new language – rather like putting a Ford-T engine in an MG-sport – but the intention is very clear. Even programmers are slaves of routine and on numerous occasions it has proven extremely difficult to make them start all over on a new language. That is why COMAL-80 offers the programmer a chance to work as if it was the good old BASIC, and slowly move to the expanded facilities when the need occurs. Not only can one still be a BASIC-freak and run COMAL-80, but even the valuable programs already made in BASIC can run in COMAL-80. For the sake of honesty, BASIC has a lot of dialects and some of them are in fact contradictive; therefore some minor editorial correc-

tions may be needed, but certainly no routines need reprogramming.

Though the line-numbers are incorporated there is no need to use them, as all references to a program can be done by labels. There is no need to keep track of absolute references such as

```
30 GOTO 5000
```

although it can be done, it could as well be

```
30 GOTO start
```

for which "start" has been defined in another line, such as

```
100 LABEL start
```

Another significant difference which actually makes this language so powerful is the structured part which, as already mentioned, is taken in from Pascal.

First of all the procedure concept: this instruction is roughly an expansion of GOSUB in BASIC, but significantly more powerful. The instruction has the form of:

```
PROC name
:
ENDPROC name
```

name which may be called from any part of the program by the instruction

```
EXEC name
```

Parameters may be transferred to a procedure, which may even be declared closed. This gives the advantage of allowing the same variable names to be used both inside and outside a procedure. Such closing may again be opened by using the GLOBAL-instruction. Obscure procedure names are no longer needed, as all names may include up to 16 characters. Meaningful names improve the readability and the understanding of a program.

COMAL-80 includes conditional instructions. The BASIC instruction IF . . . THEN is of course included, but it has been heavily expanded to the combined IF-instruction:

```
IF . . . (THEN) . . . (ELIF) . . .
   (ELSE) . . . (ENDIF)
```

In the same way

```
ON i GOTO k,l,m
```

may be used or substituted by a stronger term:

```
CASE . . . (OF) . . . WHEN . . .
   (OTHERWISE) . . . ENDCASE
```

Repeating instructions known from BASIC:

```
FOR . . . TO . . . NEXT
```

may even be used backwards in COMAL-80, replacing TO by DOWNTO. Further it includes 3 powerful instructions:

```
WHILE . . . ENDWHILE
REPEAT . . . UNTIL
LOOP . . . EXIT . . . ENDLOOP
```

– instructions which certainly make working a lot easier to the programmer.

The above examples can give but an impression of the significant benefits included in COMAL-80. Hopefully, the program in Figure 2 will prove more than this brief description.

However, one more advantage deserves mentioning: the PRINT USING function has been expanded to have a certain resemblance to the FORMAT-instruction used in other languages, and it is allowed to write:

```
100 a$:="##.###"

200 PRINT USING a$: variable
```

## How Does It Work?

As mentioned COMAL-80 has been implemented as an interpreter, the advantages of which are numerous, but leaving us with the drawback of lower speed. However, there are ways to compensate for this and increase the speed.

This program has been divided into three parts, each doing its own job.

Part one consists of two sub-modules: the editor and the syntax control working while keying in the program. The editor allows writing, deleting and inserting anywhere on the line, and upon completion, indicated by pressing RETURN, the line is transferred to the syntax-control, which checks the line against COMAL-80 specifications. If a syntax-error is found, the line is redisplayed, the cursor indicating the error-location. Also, an error-code and one of more than 200 error-messages, which include over 3,000 letters, explains what is wrong. At the same time it returns to the edit-mode, ready for corrections. This repeats until the line is accepted, upon which the line is transferred to internal format, ready for later execution.

Module 2 – the prepass – takes over when the program is to be executed. This module concludes the translation into internal format. Among others, it extends referring lines by the absolute memory address for the line referred to.

The program is now brought to a format in which the 3rd module can work very rapidly. In fact, the runtime module of COMAL-80 has proved in several cases to be faster than BASIC on a 16-bit minicomputer.

## Availability

In March 1981, COMAL-80 was known to be available for different micros running the CP/M operating system, and by now it is probably available for other systems too.

## Conclusion

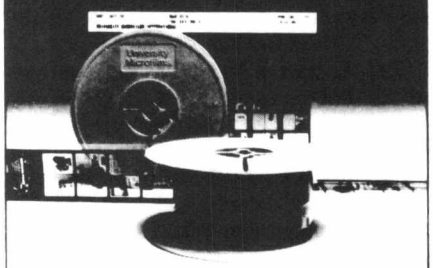
The implementation of COMAL-80 has demonstrated that a great idea may as well come from a small country, and that even the complicated task it is to turn the idea into a new EDP-language can be done to the extent of gaining international recognition.

For readers wanting further details on COMAL-80, they should contact Metanic ApS, Kongevejen 177, DK-2830 Virum, Denmark.

■■

**LISTING ON PAGE 46**

This publication is available in microform.



**University Microfilms International**

300 North Zeeb Road
Dept. P.R.
Ann Arbor, Mi. 48106
U.S.A.

30-32 Mortimer Street
Dept. P.R.
London W1N 7RA
England

# Guest Essay

# Re~Thinking Program Editors

## The Problem

It seems to me that a great obstacle to better programming is the lack of an editor that is as well adapted to its purpose as VisiCalc is adapted to calculation in rows and columns. We are writing programs, essentially, in the order and style required by a compiler or interpreter, not that best adapted to thinking about programs. This is, I think, the root of many objections to Pascal and FORTH (for instance *DDJ* #48, #51 letters.)

The order in which the programs have to be written does not seem to suit the way in which people think about programming. I don't propose to change the way the languages work, but the way we edit programs. If you will bear with me for a little hand waving, perhaps I can explain what I mean.


## by William B. Brogden

*William B. Brogden, c/o Island Cybernetics, P. O. Box 208, Port Aransas, TX 78373.*


## Possible Solution

Suppose we have an "electronic blackboard," a two dimensional surface about which we could move freely. On this surface we can create and modify "text boxes" which have labels and text inside. On the extreme right we may have a set of boxes representing the primitive operators of the language we are using. Progressively higher-order routines go across with the final main program on the extreme left. The rule would be that a routine can only refer to itself or to a routine/box/operator to the right. This is similar to the Warnier-Orr diagramming technique.

Now for the third dimension. At the closest level of approach, we would just "see" and work in one "text box." It would look just like a regular editor. If we "back off" one step, the display would show two or three adjacent boxes, each with a label and the first few lines of text (Figure 1). Backing off another step might show only the labels (Figure 2), and a third would just show a dot for each box (Figure 3). It would be like a variable zoom; we could zoom back, move over to a new text box, zoom in and work on it.

We could also move directly to another text box by name. Naturally, the name of each box is the name of the subroutine, or whatever, inside it.

Whenever we wanted to compile the program, the editor would know how to "unravel" the two-dimensional layout into the linear file required by the compiler or interpreter. Furthermore, the editor would be able to check each text box on demand to ensure that all routines or variables or operators called for exist as labels of boxes to the right.

**FIGURE 1. Editor display, one "step back." This is not intended to be a specific language.**

```
*-----------------------------------------------------------*
:                                                           :
:                                   * OPEN ----------------:
:    * MAIN ---------------*        :  CALL GET.FILE.NAME   :
:    :  CALL INIT          :        :    (etc)              :
:    :  CALL OPEN          :        :                       :
:    :  WHILE NOT( FOUND ) :        *----------------------:
:    :  CALL READ          :                                :
:    :  CALL SEARCH        :                                :
:    :  WEND               :        * READ ----------------:
:    :  CALL OUTPUT        :        :  READ #1 IN$          :
:    :  (etc)              :        :  IF EOF THEN FOUND =  :
:    *---------------------*        :     (etc)             :
:                                   :     (etc. off screen  :
*-----------------------------------------------------------*
```

**FIGURE 2. Editor display, two "steps back."**

```
*-----------------------------------------------------------*
:                                                           :
:                   * INIT ------*     * IN$ -------*       :
:                                                           :
:                   * OPEN ------*     (other variables)    :
: * MAIN ------*                                            :
:                   * READ ------*     ( etc.)              :
:                                                           :
:                   * SEARCH ----*                          :
:                                                           :
:                   * OUTPUT ----*                          :
:                                                           :
:                                                           :
*-----------------------------------------------------------*
```

**FIGURE 3. Editor display, three "steps back."**

```
*-----------------------------------------------------------*
:                       *                                   :
:               *       *                                   :
:                                                           :
:      *        *       *                                   :
:                       *                                   :
:               *                                           :
:                                                           :
:                       *                                   :
:               *       *                                   :
:               *       *                                   :
:                       *                                   :
:                                                           :
*-----------------------------------------------------------*
```


*Dr. Dobb's* readers have been aiding the growth and development of the micro field for years. If you would like to make a contribution, contact us at:

Dr. Dobb's Journal
P.O. Box E
Menlo Park, CA 94025
(415) 323-3111

The editing program would only need to keep in memory a list of the boxes and some buffers for the current and recently active boxes. The actual text content of most boxes would be out on disk storage. This would mean that working on the largest program would not slow down editing. Compare that with the problems of moving backward and forward long distances with a linear editor. It might be feasible to have a split screen showing the text of two boxes at once, depending on the "width" of the boxes.

## The Hardware

Certainly this editor would require a fairly fast disk and a rather large amount of memory. It is also obvious that only high-speed terminals with cursor addressing or memory-mapped displays could be used, but I don't think there are too many people doing serious programming on teletypes anymore.

## The Advantages

Now, what do we gain by this? Mainly, the order which the language wants no longer dominates the order of program writing. The overall structure of the program can be created first, as a layout of boxes (with comments), in a direct parallel with a Warnier-Orr diagram. Without changing technique, we can start to code, at any "level" in the program. If we want to check (for instance) how parameters are passed to a subroutine, we can jump right to it. Anytime we want an overview of the program, just zoom back one or two steps and there you are.

## The References

Precedents for this sort of editor would appear to be some of the "actor" languages and two-dimensional data base layouts. The "screen" editor of FORTH is close but is not flexible enough. The article which most served to crystallize my thoughts on this, and which I recommend to all, was "Interactive Systems & Virtuality" by Ted Nelson, in *Creative Computing*, Nov. & Dec., 1980. Warnier-Orr diagrams are described in *BYTE*, Dec. 1977 & Jan. 1978.

## The Next Step

By presenting this idea in its present, half-baked form, I hope to stimulate the brilliant/unconventional/hardworking *Dr. Dobb's* community to start thinking in the direction of what a really good program editor should do.

If you come up with something good, please write.

■■

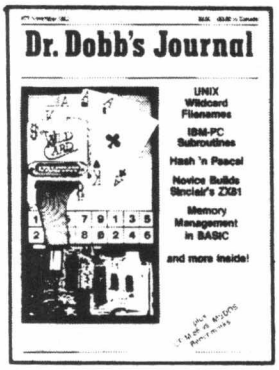

# Dr. Dobb's Journal

For Users of Small Computer Systems

Each issue includes:
- valuable software tools
- algorithms & problem solving
- industry news
- important product reviews

With in-depth coverage of:
- telecommunications
- systems programming
- language development
- machine independent programs

and much, much more!




Yes! Please enter my subscription for
☐ 1 yr. (12 issues) $25 ☐ 2 yrs. $47 (save $13 off newsstand)
☐ Please bill me ☐ I enclose check/money order

Name_____

Address _____

City_____ State_____ Zip_____

**Dr. Dobb's Journal, A Publication of People's Computer Company**
**1263 El Camino Real, P.O. Box E, Menlo Park, CA 94025**

## Are you ready?

*DDJ*, the world's foremost microcomputer publication, has been working for years to prepare its readers to be innovators, to lead the wave of breakthroughs in our changing technology.

Every issue of *Dr. Dobb's Journal* helps one to understand the nuts and bolts of small computer systems. We offer entire listings of valuable software: our pages have included compilers, cross-assemblers, editors, new languages, hardware interfaces and more – usually before anyone else thinks of them!

## Even more important!

Our subscribers share insights, correspond, and contribute to one another's work, more than any other group we know. They treat *Dr. Dobb's Journal* as a "hands-on" publication.

This warm cooperation has done more to refine software products, and generally to advance the state of microcomputer technology, than perhaps any other resource. And it is available to you through our pages!

## For the straight facts . . . . . .

If you are a serious computing professional or enthusiast, then you should take a very close look at what *DDJ* offers you. We've been on the cutting edge since 1976.

# H8 Text Editor to CP/M File Conversion

For almost three years after I got my H8 computer I put up with the increasingly frustrating vicissitudes of cassette I/O. During this time I built up a large collection of tape files: assembler source, Tiny Pascal source, and general data, as well as BASIC source and "memory image" (load-execute) files. When I finally bought two disk drives and CP/M, I wanted to be able to transfer some of these files to disk without having to key them in by hand.

I wrote the TAPECOPY program, whose source is included here, to do this job for me. In planning the program, I realized that I really only needed to convert files which had been created by the Text Editor (TED-8). This included assembler source, Pascal source, and data files. Converting BASIC source and memory image files involves special problems, and I did not feel that I needed this capability. (I was not planning to get a disk BASIC.) *So TAPECOPY converts only Text Editor files.* Anyone interested in converting other file types should study the "Tape Files" section in the H8 "Software Reference Manual," and use TAPECOPY as a basis for their own program.

Instructions for the use of the program are included in the initial comments in the source. The program has the option of viewing the data on the screen as it is being transferred. This can be helpful since there is no indicator of the progress of the conversion other than the movement of the tape, but it can slow down execution, particularly if your screen has a low baud rate.

If the disk file specified already exists, you are given the chance to erase the file or abort the program. In the latter case, you would simply rerun the program, specifying a different, non-existing file name.

In opening the tape file, the program checks the first complete record to see if it is a TED-8 file with record sequence number zero. If both of these conditions are not true, it is assumed that you positioned the tape in the recorder ahead of the desired file; the tape is then searched for the first occurrence of the correct record type.

If successive records do not increase in sequence number by 1 until the end of the file is reached (the last record has the EOF bit on), the program terminates with the message "Tape Sequence Error." Other error messages which can occur have to do with disk problems, and are mostly self-explanatory.

As mentioned in the program comments, a checksum error in a tape record will cause the program to stop with no message. The only indication of a problem will be when you eventually notice that the recorder is not moving and no other activity is taking place. I have not attempted to fix this inconvenient characteristic of the program, mainly because I consider it a temporary utility; I will not be using cassette much after I get my files converted. However, the following comments may be of help to anyone who is inclined to try to clean up this detail: routines in the Panel Monitor (PAM-8) ROM are invoked to find the beginning of a record, to read the next byte from tape, and to verify the tape checksum. The PAM-8 code would ordinarily sound the H8 audible alarm when a checksum error is detected; however, the code to do this depends on the 2mS clock interrupt. CP/M seems to run with interrupts disabled much of the time, so the error routine probably gets into an endless "disabled" loop. The problem could probably be eliminated by replacing the PAM-8 routines with your own similar routines which could assure that interrupts are enabled at the proper times by using the EI (Enable Interrupts) and DI (Disable Interrupts) instructions. The source for PAM-8 is in the H8 "Software Reference Manual."

The main loop of the program simply calls GETCHAR and PUTCHAR repeatedly until the tape file is finished. The first time GETCHAR is called, it finds the beginning of the tape file and reads the header record and the first data record. Each time it is called, it returns the next data byte from the input buffer, and it reads a new record into the buffer whenever the previous record is exhausted. In two cases (see below), the byte returned is a translation of, not identical to, what was on the tape. The first time PUTCHAR is called, it opens the disk file. Each time it is called, it adds the new byte to the output buffer and, when necessary, writes a disk record and clears the output buffer.

The tape record format is described in the "Tape Files" section of the H8 "Software Reference Manual," which refers to the TED-8 format as "compressed text." Each string of spaces is compressed by TED-8 into a single byte consisting of the hex value 80 plus n, where n is the number of spaces. Also, there are no carriage return or line-feed characters in the tape records; the end of each line is represented by a byte with a value of zero. TAPECOPY expands bytes of hex value 81 and higher into spaces, and it expands a zero byte into a carriage return/line-feed pair.

This program was developed under CP/M for Heath H8, Version 1.43. For Standard CP/M (zero origin), change the BASE EQU from 4200H to 0H. I later installed Heath's "Extended Configuration" option. I found that the program did not work. I have not yet determined why this is true, but I was able to fix the problem by copying the needed routines from the Panel Monitor into my source program. This also eliminated the problem of system death when a checksum error occurs on the input tape.

The details of the fix are as follows:

1. Copy the routines CTC, SRS, RNP, RNB, and CRC.

2. Drop the equates for CTC, SRS, and RNB in the original TAPECOPY.

3. Add equates for ASYN, ASTX, UCIRO, UCIER, UCIRE, USRRXR, IPTPD, and IPTPC.

4. In CTC, change the MVI A,1 instruction to:

```
XRA   A
OUT   OPTPC ;TURN OFF TAPE.
CALL  MSGOUT
DB    'TAPE CHECKSUM
        ERROR',CR,LF,0
JMP   BOOT
```

5. In RNB, change the CALL TPXIT instruction to:

```
IN    IPTPC
```

6. Include the following DS:

```
CRCSUM  DS  2  ;CRC-16
                CHECKSUM
```

7. Change JE and JNE opcodes to JZ and JNZ, respectively, in the SRS routine. Otherwise, the CP/M assembler will do terrible things.

*(Continued on page 50)*

**(LISTING ON PAGE 40)**

**by Jim Scott**

*Jim Scott, 1724 Blakemore Road, Richmond, VA 23225.*

# A Tiny File System

by William A. Gale

How can we describe the machine-dependent parts of a portable system? I've been working on a portable operating system which now includes a small editor, a small compiler, and a compiler generator based on MetaII. But they all rest on file input and output which is very machine-dependent. This is an attempt to specify a simple file system that I hope I can replicate on my next computer. It works on my current Apple II and has supported further development of a single user bootstrap operating system.

This "tiny" file system is to a complete file system as tiny BASIC is to BASIC. That is, it is severly truncated. Space allocation is sequential, which is too simple for a complete file system. Directory maintenance, along with file entry and removal, rest entirely in human hands. However, these minimal files can be used to implement a more elaborate file system which need not see the machine dependencies of this tiny file system.

Since the file system is much simpler than Apple's DOS, one might ask why I went to the trouble. First, my intention was to provide a basis for portability to future machines I might have. So I didn't want to tie into DOS except possibly at a buried level. Second, I'd rather not have to pay the space penalty for the entire DOS. This simple file system used only Apple's RWTS routine (Read Write Track Sector) from DOS and leaves a net gain of 8K bytes. Using the entire resources of a small computer one always faces a trade-off between complexity in one part of a system for complexity in other parts. A choice in complexity and size of file system is worth having. Third, the buffer sizes required by this file system are half the size required by Apple's file format. I have sketches for (machine independent) assignment programs, and for a portable directory system that extends this system.

*William A. Gale, 439 S. Orange Ave., S. Orange, NJ 07079.*

*The Author works in statistics and economics at Bell Telephone Laboratories.*

© 1981 by William A. Gale.
All Rights Reserved.

In planning for an assembly language version of this simple file system, I thought about it by writing high level "programs." These are not in any known programming language, but tend to use C lexical conventions and should be easily readable. Writing these high level specs let me identify precisely what I wanted the outside to look like and let me define a set of internal variables and procedures to accomplish that. Having it written probably eased the assembly langauge programming. I found when I was done that I had made my usual error per twenty assembly lines, but only one error at the high level, so I didn't have any massive revisions to carry out. I expect a third benefit that lies in the future when I do it again on another machine. I hope it makes the assembly language programming easier there. This basic file system represents about 80% of the machine language support for the portable code, so easing it means easing a hard part of transporting my code.

My goal was to write simple and easily repaired code. Since input and output take a lot of time, I was not worried about time very much. As it works out, the system charges an overhead of about 170 microseconds per byte read with this code. That corresponds to about 2 seconds in reading the entire disk, or about 6% overhead on the 30 seconds this seems to take on an Apple.

The description has three levels of increasing machine dependence. The outermost layer is as machine independent as possible. It embodies the purpose of this software, to provide sequential files with single character access. At this level, the storage device consists of a numbered sequence of abstract BUCKETS (sectors, blocks, . . .), numbered from zero to MAXBUCK. MAXBUCK is a necessary, machine-dependent parameter. Each bucket can store MAXCHAR characters, after allowing for a little overhead for lower level variables, each field that is opened requires the assignment of a BUFFER, which holds the data, some lower-level variables, and any machine-dependent data required. The size of this buffer is thus machine-dependent. The system returns an error code to the special location ER along with each operation. The values that ER can take are

0 no error
1 end of file or end of medium
2 bad transaction requested
3 hardware error

When end of file is encountered on reading, the character EOF is returned.

In thinking about the scale of the parameters MAXBUCK and MAXCHAR, I had in mind three possible storage configurations for an Apple:

1. The single disk I now have. It has 35 tracks of 13 sectors, each with 256 characters.
2. Four Apple disks as a unit. Two slots, two drives, 35 tracks, 13 or 16 sectors, 256 characters.
3. A hard disk holding 30 megabytes in 60K sectors of 512 bytes each.

For these I planned on

| | 1 | 2 | 3 |
|---|---|---|---|
| MAXBUCK | 454 | 1819 | 60000 |
| MAXCHAR | 252 | 252 | 508 |
| buffersize | 175 | 275 | ? |

The specific operations the file system supports are:

OPEN a BUFFER for READ or WRITE at a given BUCKET. The file extends from a given bucket onward.

CLOSE a BUFFER. If writing the buffer will be flushed and the last bucket marked as the end of the file.

PUT a character into a BUFFER. The buffer is emptied to disk automatically when it is full of data.

GET a character from a BUFFER. The buffer is everflowing, being refilled from hardware as required.

It turns out that it might as well support the more primitive

READ a BUCKET into BUFFER.
WRITE a BUFFER into BUCKET.

since PUT and GET need them. Also, they look useful for implementing random access files, including directories.

The second layer of the interface adopts a particular means of supporting the outer layer, by describing the buffer contents. The buffer contains, in the first place, the image of a bucket, with data from the zeroth to MAXCHAR positions, followed by BYTESPRESENT and NEXTBUCK. BYTESPRESENT is a variable which shows the number of characters entered in the bucket. To accommodate a possible 508 MAXCHAR bucket, I made it two bytes. NEXTBUCK is a variable which contains the bucket number of the next following bucket in the file. Here this is just the current bucket +1, but a more elaborate assignment system will need to record the bucket. To accommodate even 455 buckets, it needs to be two bytes, which is also enough for the 30 megabyte disk. The last bucket in a file is shown by setting NEXTBUCK to

zero.

The buffer also contains space beyond the bucket image for

CURRBUCK – the bucket from which the data was read or to which it will be written.

NEXTCHAR – the next character to be read or written.

TRANSACT – the purpose for which the buffer may be used. It has values

CLOSED
OPENREAD
OPENWRITE

Finally, the buffer has space for any machine-dependent variables required (Apple's IOB table in my case).

The innermost layer of the file system gives the mapping from buckets to local coordinates – blocks or sectors or what have you. The mapping should make the "next" sequential bucket an easy one to get from the previous one. For one Apple drive, I specified

Slot = 6
Drive = 1
Track = Bucket/13
Sector = Bucket mod 13.

If Apple had not interleaved the sectors of a track already, this would have been a good place to do so. This layer also contains the details of transferring a bucket image form memory to the hardware. I used Apple's RWTS routine. This is the fundamental capability that the high level routines assume. It seems reasonable to stop here, because any further down is hopelessly machine-dependent.

### High Level Specs

```
Open BUFFER for TRANS at BUCKET
   ER=0
   if(TRANSACT!=CLOSED)
      {ER=2; return}
   TRANSACT=OPEN-TRANS
   CURRBUCK=BUCKET
   NEXTBUCK=0
   NEXTCHAR=0
   BYTESPRESENT=0
   fill data space with EOF

Close BUFFER
   ER=0
   if(TRANSACT==CLOSED)
      {ER=2; return}
   if(TRANSACT==OPENWRITE)
      {NEXTBUCK=0
       write BUFFER
       }
   TRANSACT=CLOSED

Read BUFFER
   ER=0
   if(TRANSACT!=OPENREAD)
      {ER=2; return}
   if(CURRBUCK=0)
      {ER=1; return}
   input bucket image from CURRBUCK
   if(read error)
      {ER=3; return}
   CURRBUCK=NEXTBUCK
   NEXTCHAR=0

Get CHAR from BUFFER
   ER=0
   if(TRANSACT!=OPENREAD)
      {ER=2; return}
   if(NEXTCHAR==BYTESPRESENT)
      {if(CURRBUCK=0)
         {CHAR=EOF; ER=1; return}
       else
         read BUFFER
       }
   if(ER!=0) return;
   CHAR=BUFFER[NEXTCHAR]
   NEXTCHAR++

Write BUFFER
   ER=0
   if(TRANSACT!=OPENWRITE)
      {ER=2; return}
   if(CURRBUCK>MAXBUCK
      or NEXTBUCK>MAXBUCK)
      {ER=1; return}
   transfer bucket image to CURRBUCK
            on hardware
   BYTESPRESENT=0
   NEXTCHAR=0
   fill data space with EOF

   Put CHAR into BUFFER
      ER=0
      if(TRANSACT!=OPENWRITE)
         {ER=2; return}
      if(NEXTCHAR==MAXCHAR)
         {assign NEXTBUCK
                 (=CURRBUCK+1)
          write BUFFER
          CURRBUCK=NEXTBUCK
          }
      if(ER!=0) return
      BUFFER[NEXTCHAR]=CHAR
      NEXTCHAR++
      BYTESPRESENT=NEXTCHAR
```

**(LISTING ON PAGE 40)**

# A Cross Assembler for the 8080 or Z80

## Listing I

MNE-CODE is the source file from which the XASM-86 symbol table is compiled. The symbol table compiler (MAKETABLE) will be listed in a future installment if reader interest warrants it.

```
*
0000 ********************************************************************************
0010 *
0020           TAB    10,17,24
0030 *
0040 * MNE-CODE (MNEMONIC - CODE TABLE) IS THE CONFIGURATION FILE FOR
0050 * THE Z-88 CROSS-ASSEMBLER, XASM.  MNE-CODE IS PROCESSED BY
0060 * MAKETABLE TO YIELD AN XASM-USABLE SYMBOL TABLE.  SINCE THE USER
0070 * MAY MODIFY MNE-CODE TO INCLUDE ANY OPCODE OR REGISTER, IT
0080 * IS FEASIBLE TO CREATE ASSEMBLERS FOR FUTURE PROCESSORS SIMPLY BY
0090 * MODIFYING MNE-CODE.
0100 *
0110 * MNE-CODE PRODUCES 8088/8086 CODE, BUT DOES NOT INCLUDE THE
0120 * FOLLOWING OPCODES:
0130 *
0140 * ANY SHORT FORM REGISTER OPERATIONS
0150 * MOVE,ARITHMETIC, & LOGIC ACCUMULATOR-ONLY OPERATIONS
0160 * LAHF, SAHF
0170 * JMP DIRECT SHORT
0180 * ESC
0190 * LOCK
0200 *
0210 * IT IS FELT THAT THESE "SPECIAL" OPERATIONS COULD BE MOSTLY
0220 * REPRODUCED BY THE MORE SYMMETRICAL, ELEGANT REGISTER/MEMORY 8086
0230 * OPCODES.  THIS WAS DEEMED AN ADVANTAGE, SINCE IT OFFERED THE
0240 * END-USER A SIMPLER INSTRUCTION SET WITH FEWER "TRAPS".
0250 *
0260 * FINALLY, A SET OF AMD-9511 OPCODES IS INCLUDED FOR USERS WHO
0270 * ARE FORTUNATE ENOUGH TO HAVE ACCESS TO THIS CHIP IN THEIR
0280 * SYSTEM.  COMBINED WITH A SHORT INTERPRETER, THESE CODES ALLOW
0290 * THE FORMATION OF POWERFUL AND FAST AITHMETIC ROUTINES.
0300 *
0310 * NOTE THAT "ESC" FAMILY OPERATIONS (11011XXX MODXXXR/M)
0320 * OR D8XX - DFXX OPCODES MAY BE INCLUDED AT ANY FUTURE DATE.
0330 * THIS IS ADVANTAGEOUS SINCE INTEL'S COPROCESSOR FAMILY WILL
0340 * PROBABLY MAKE USE OF THESE "UNDEFINED" INSTRUCTIONS.
0350 *
0360 * THE TYPE COMMAND SETS SYMTYPE=1ST OP, VALTYPE=2ND OP
0370 *
0380                          ;
0390                          ;
0400                          ;
0410                          ;
0420                          ;
0430 *
0440 * 16-BIT REGISTERS
0450 *
0460                          ;
0470 TYPE      4      4
0480                          ;
0490 AX        00
1030 * 8088/8086 OPCODES
1040 *
1050                          ;
1060 TYPE      0      0
1070                          ;
1080 MOV       2302   8900    ;MOVE <OPERAND1> TO <OPERAND2>
1090 MOVB      0102   8800
1100 MVI       9202   C700    ;MOVE <IMMEDIATE DATA> TO <OPERAND2>
1110 MVIB      8002   C600
1120 MVS       4402   8C00    ;MOVE <OPERAND1> TO <OPERAND2>
1130                          ;ONE OPERAND MUST BE SEGMENT REGISTER
1140                          ;
1150 PUSH      2B01   FF30    ;PUSH <OPERAND1>
1160 PUSHS     5B01   06      ;PUSH <SEGMENT REGISTER>
1170 PUSHF     BB00   9C      ;PUSH <FLAGS>
1180                          ;
1190 POP       2B01   8F00    ;POP <OPERAND1>
1200 POPS      5B01   07      ;POP <SEGMENT REGISTER>
1210 POPF      BB00   9D      ;POP <FLAGS>
1220                          ;
1230 XCHG      2202   8700    ;EXCHANGE <OPERAND1> WITH <OPERAND2>
1240 XCHGB     0002   8600
1250                          ;
1260 IN        8B01   E5      ;INPUT DATA FROM <PORT 0 - 255>
1270 INB       8B01   E4
1280 INW       BB00   ED      ;INPUT DATA FROM PORT 0 - 65536 INDIRECT IN DX
1290 INWB      BB00   EC
1300                          ;
1310 OUT       8B01   E7      ;OUTPUT DATA TO <PORT 0 - 255>
1320 OUTB      8B01   E6
1330 OTW       BB00   EF      ;OUTPUT DATA TO PORT 0 - 65536 INDIRECT IN DX
1340 OTWB      BB00   EE
1350                          ;
1360 USE       5B01   26      ;USE <SEGMENT REGISTER> IN NEXT OPCODE
1370                          ;
1380 XLAT      BB00   D7      ;TRANSLATE AL = BYTE AT (BX + AL)
1390 LEA       2202   8D00    ;LOAD OFFSET ADDRESS  OF <OPERAND1> TO <OPERAND2>
1400                          ;
1410 LDS       2202   C500    ;LOAD SEGMENT ADDRESS OF <OPERAND1> TO DS  &
1420                          ;LOAD OFFSET ADDRESS  OF <OPERAND1> TO <OPERAND2>
1430                          ;
1440 LES       2202   C400    ;LOAD SEGMENT ADDRESS OF <OPERAND1> TO ES  &
1450                          ;LOAD OFFSET ADDRESS  OF <OPERAND1> TO <OPERAND2>
1460                          ;
1470 ADD       2302   0100    ;ADD <OPERAND1> TO <OPERAND2>
1480 ADDB      0102   0000
1490 ADI       9202   8100    ;ADD <IMMEDIATE DATA> TO <OPERAND2>
1500 ADIB      8002   8000
1510                          ;
1520 ADC       2302   1100    ;ADD <OPERAND1> + CARRY TO <OPERAND2>
1530 ADCB      0102   1000
1540 ACI       9202   8110    ;ADD <IMMEDIATE DATA> + CARRY TO <OPERAND2>
1550 ACIB      8002   8010
1560                          ;
1570 INC       2B01   FF00    ;INCREMENT <OPERAND1> BY 1
```

```
0500 CX         09
0510 DX         12
0520 BX         1B
0530 SP         24
0540 BP         2D
0550 SI         36
0560 DI         3F
0570                            ;
0580 *
0590 * 8-BIT REGISTERS
0600 *
0610                            ;
0620 TYPE       5       4
0630                            ;
0640 AL         00
0650 CL         09
0660 DL         12
0670 BL         1B
0680 AH         24
0690 CH         2D
0700 DH         36
0710 BH         3F
0720                            ;
0730 *
0740 * INDIRECT MEMORY REFERENCES (@M)
0750 *
0760                            ;
0770 TYPE       6       4
0780                            ;
0790 @BX+SI     00
0800 @BX+DI     09
0810 @BP+SI     12
0820 @BP+DI     1B
0830 @SI        24
0840 @DI        2D
0850 @BP        36
0860 @BX        3F
0870                            ;
0880 *
0890 * SEGMENT REGISTERS
0900 *
0910                            ;
0920 TYPE       7       4
0930                            ;
0940 ES         00
0950 CS         09
0960 SS         12
0970 DS         1B
0980                            ;
0990                            ;
1000                            ;
1010                            ;
1020 *
```

*(Continued on top of page 22)*

```
1580 INCB       0B01    FE00
1590                            ;
1600 AAAD       BB00    37      ;POST-ADJUST RESULT OF ASCII ADD
1610 DAAD       BB00    47      ;POST-ADJUST RESULT OF  BCD ADD
1620                            ;
1630 SUB        2302    2900    ;SUBTRACT <OPERAND1> FROM <OPERAND2>
1640 SUBB       0102    2800
1650 SUI        9202    8128    ;SUBTRACT <IMMEDIATE DATA> FROM <OPERAND2>
1660 SUIB       8002    8028
1670                            ;
1680 SBB        2302    1900    ;SUBTRACT <OPERAND1> + CARRY FROM <OPERAND2>
1690 SBBB       0102    1800
1700 SBI        9202    8118    ;SUBTRACT <IMMEDIATE DATA> + CARRY FROM <OPERAND2>
1710 SBIB       8002    8018
1720                            ;
1730 DEC        2B01    FF08    ;DECREMENT <OPERAND1> BY 1
1740 DECB       0B01    FE08
1750                            ;
1760 NEG        2B01    F718    ;NEGATE <OPERAND1>
1770 NEGB       0B01    F618
1780                            ;
1790 CMP        2302    3900    ;COMPARE <OPERAND1> TO <OPERAND2>
1800 CMPB       0102    3800
1810 CPI        9202    8138    ;COMPARE <IMMEDIATE DATA> TO <OPERAND2>
1820 CPIB       8002    8038
1830                            ;
1840 AASU       BB00    3F      ;POST-ADJUST RESULT OF ASCII SUBTRACT
1850 DASU       BB00    4F      ;POST-ADJUST RESULT OF BCD SUBTRACT
1860                            ;
1870 MUL        2B01    F720    ;UNSIGNED MULTIPLY AX BY <OPERAND1>
1880 MULB       0B01    F620    ;UNSIGNED MULTIPLY AL BY <OPERAND1>
1890 IMUL       2B01    F728    ;SIGNED   MULTIPLY AX BY <OPERAND1>
1900 IMULB      0B01    F628    ;SIGNED   MULTIPLY AL BY <OPERAND1>
1910                            ;
1920 AAMU       BB00    D40A    ;POST-ADJUST RESULT OF ASCII MULTIPLY
1930                            ;
1940 DIV        2B01    F730    ;UNSIGNED DIVIDE AX:DX BY <OPERAND1>
1950 DIVB       0B01    F630    ;UNSIGNED DIVIDE AX    BY <OPERAND1>
1960 IDIV       2B01    F738    ;SIGNED   DIVIDE AX:DX BY <OPERAND1>
1970 IDIVB      0B01    F638    ;SIGNED   DIVIDE AX    BY <OPERAND1>
1980                            ;
1990 AADU       BB00    D50A    ;PRE-ADJUST FOR SUBSEQUENT ASCII DIVIDE
2000 CBW        BB00    98      ;CONVERT BYTE TO WORD (SIGN-EXTEND)
2010 CWP        BB00    99      ;CONVERT WORD TO PAIR (SIGN-EXTEND)
2020                            ;
2030 NOT        2B01    F710    ;1'S-COMPLEMENT <OPERAND1>
2040 NOTB       0B01    F610
2050                            ;
2060 SHL        2602    D120    ;SHIFT <OPERAND1> LEFT INSERTING ZERO(S)
2070 SHLB       0602    D020
2080                            ;
2090 SAL        2602    D120    ;SHIFT <OPERAND1> LEFT INSERTING ZERO(S)
2100 SALB       0602    D020
```

*(Continued on next page)*

# A Cross Assembler for the 8080 or Z80 (Continued)

```
2110                                    ;
2120 SHR        2602    D128    ;SHIFT <OPERAND1> RIGHT INSERTING ZERO(S)
2130 SHRB       0602    D028
2140                                    ;
2150 SAR        2602    D138    ;SHIFT <OPERAND1> RIGHT INSERTING HI-BIT(S)
2160 SARB       0602    D038
2170                                    ;
2180 ROL        2602    D100    ;ROTATE <OPERAND1> LEFT
2190 ROLB       0602    D000
2200                                    ;
2210 ROR        2602    D108    ;ROTATE <OPERAND1> RIGHT
2220 RORB       0602    D008
2230                                    ;
2240 RLC        2602    D110    ;ROTATE <OPERAND1> LEFT THROUGH CARRY
2250 RLCB       0602    D010
2260                                    ;
2270 RRC        2602    D118    ;ROTATE <OPERAND1> RIGHT THROUGH CARRY
2280 RRCB       0602    D018
2290                                    ;
2300 AND        2302    4100    ;AND <OPERAND1> TO <OPERAND2>
2310 ANDB       0102    4000
2320 ANI        9202    8120    ;AND <IMMEDIATE DATA> TO <OPERAND2>
2330 ANIB       8002    8020
2340                                    ;
2350 TEST       2202    8500    ;AND-TO-SET-FLAGS <OPERAND1> TO <OPERAND2>
2360 TESTB      0002    8400
2370 TSTI       9202    F700    ;AND-TO-SET-FLAGS <IMMEDIATE DATA> TO <OPERAND2>
2380 TSTIB      8002    F600
2390                                    ;
2400 OR         2302    0900    ;OR <OPERAND1> TO <OPERAND2>
2410 ORB        0102    0800
2420 ORI        9202    8108    ;OR <IMMEDIATE DATA> TO <OPERAND2>
2430 ORIB       8002    8008
2440                                    ;
2450 XOR        2302    3100    ;EXCLUSIVE-OR <OPERAND1> TO <OPERAND2>
2460 XORB       0102    3000
2470 XRI        9202    8130    ;EXCLUSIVE-OR <IMMEDIATE DATA> TO <OPERAND2>
2480 XRIB       8002    8030
2490                                    ;
2500 REPZ       BB00    F3      ;REPEAT NEXT INSTRUCTION WHILE ZERO FLAG SET
2510 REPNZ      BB00    F2      ;REPEAT NEXT INSTRUCTION WHILE ZERO FLAG RESET
2520                                    ;
2530 MOVC       BB00    A5      ;MOVE @SI TO @DI
2540 MOVCB      BB00    A4
2550                                    ;
2560 CMPC       BB00    A7      ;COMPARE @SI TO @DI
2570 CMPCB      BB00    A6
2580                                    ;
2590 SCAC       BB00    AF      ;COMPARE AX TO @DI
2600 SCACB      BB00    AE
2610                                    ;
2620 LODC       BB00    AD      ;LOAD AX FROM @SI

2630 LODCB      BB00    AC
2640                                    ;
```

```
3180 LOOP       AB01    E2      ;LOOP BY <DISPLACEMENT> CX TIMES
3190 LOOPZ      AB01    E1      ;LOOP BY <DISPLACEMENT> WHILE ZERO SET
3200 LOOPNZ     AB01    E0      ;LOOP BY <DISPLACEMENT> WHILE ZERO RESET
3210 JCXZ       AB01    E3      ;JUMP BY <DISPLACEMENT> IF CX = 0
3220                            ;
3230 LOOPE      AB01    E1      ;LOOP BY <DISPLACEMENT> WHILE EQUAL
3240 LOOPNE     AB01    E0      ;LOOP BY <DISPLACEMENT> WHILE NOT EQUAL
3250                            ;
3260 INT        8B01    CD      ;EXECUTE INTERRUPT <OPERAND1>
3270 INTB       BB00    CC      ;EXECUTE INTERRUPT 3 (BREAKPOINT INTERRUPT)
3280 INTO       BB00    CE      ;EXECUTE INTERRUPT 4 ON OVERFLOW
3290                            ;
3300 CLCF       BB00    F8      ;CLEAR CARRY FLAG
3310 CMCF       BB00    F5      ;COMPLEMENT CARRY FLAG
3320 STCF       BB00    F9      ;SET CARRY FLAG
3330 CLDF       BB00    FC      ;CLEAR DIRECTION FLAG (SET AUTO-INCREMENT)
3340 STDF       BB00    FD      ;SET DIRECTION FLAG (SET AUTO-DECREMENT)
3350 CLIF       BB00    FA      ;CLEAR INTERRUPT FLAG (DISABLE INTERRUPTS)
3360 STIF       BB00    FB      ;SET INTERRUPT FLAG (ENABLE INTERRUPTS)
3370 HLT        BB00    F4      ;HALT PROCESSOR
3380 WAIT       BB00    9B      ;WAIT FOR TEST INPUT
3390 NOP        BB00    90      ;NO OPERATION (EXCHANGE AX WITH AX)
3400                            ;
3410                            ;
3420                            ;
3430                            ;
3440                            ;
3450 *
3460 * AMD-9511 OPCODES
3470 * CODES SET TO GENERATE AMD-9511 INTERRUPT,
3480 * WHICH MUST BE MASKED OUT IF NOT DESIRED.
3490 *
3500                            ;
3510 SADD       BB00    EC      ;16-BIT INTEGER ARITHMETIC
3520 SSUB       BB00    ED
3530 SMUL       BB00    EE
3540 SDIV       BB00    EF
3550                            ;
3560 DADD       BB00    AC      ;32-BIT INTEGER ARITHMETIC
3570 DSUB       BB00    AD
3580 DMUL       BB00    AE
3590 DDIV       BB00    AF
3600                            ;
3610 FADD       BB00    90      ;32-BIT FLOATING POINT ARITHMETIC
3620 FSUB       BB00    91
3630 FMUL       BB00    92
3640 FDIV       BB00    93
3650                            ;
3660 SQRT       BB00    81      ;32-BIT FLOATING POINT FUNCTIONS
3670 SQR        BB00    92
3680 SIN        BB00    82
3690 COS        BB00    83
3700 TAN        BB00    84
3710 ASIN       BB00    85
3720 ACOS       BB00    86
```

```
2650 STOC       BB00    AB      ;STORE AX INTO @DI
2660 STOCB      BB00    AA
2670                            ;
2680 CALL       7B01    E8      ;CALL ROUTINE AT <DISPLACEMENT>
2690 CALL@      2B01    FF10    ;CALL ROUTINE POINTED TO BY <OPERAND1>
2700 CALLS      9902    9A      ;CALL ROUTINE AT <DISPLACEMENT> <SEGMENT>
2710 CALL@S     2B01    FF18    ;CALL INTERSEG ROUTINE POINTED TO BY <OPERAND1>
2720                            ;
2730 JMP        7B01    E9      ;JUMP TO ROUTINE AT <DISPLACEMENT>
2740 JMP@       2B01    FF20    ;JUMP TO ROUTINE POINTED TO BY <OPERAND1>
2750 JMPS       9902    EA      ;JUMP TO ROUTINE AT <DISPLACEMENT> <SEGMENT>
2760 JMP@S      2B01    FF28    ;JUMP INTERSEG TO ROUTINE POINTED TO BY <OPERAND1>
2770                            ;
2780 RET        BB00    C3      ;RETURN FROM SUBROUTINE
2790 RSP        9B01    C2      ;RETURN ADDING <IMMEDIATE DATA> TO SP
2800 RETS       BB00    CB      ;RETURN FROM SUBROUTINE INTERSEG
2810 RSPS       9B01    CA      ;RETURN ADDING <IMMEDIATE DATA> TO SP INTERSEG
2820 IRET       BB00    CF      ;RETURN FROM INTERRUPT
2830                            ;
2840 JE         AB01    74      ;JUMP BY <DISPLACEMENT> IF EQUAL
2850 JL         AB01    7C      ;JUMP BY <DISPLACEMENT> IF LESS
2860 JLE        AB01    7E      ;JUMP BY <DISPLACEMENT> IF LESS OR EQUAL
2870 JB         AB01    72      ;JUMP BY <DISPLACEMENT> IF BELOW
2880 JBE        AB01    76      ;JUMP BY <DISPLACEMENT> IF BELOW OR EQUAL
2890 JP         AB01    7A      ;JUMP BY <DISPLACEMENT> IF PARITY
2900 JO         AB01    70      ;JUMP BY <DISPLACEMENT> IF OVERFLOW
2910 JS         AB01    78      ;JUMP BY <DISPLACEMENT> IF SIGN
2920                            ;
2930 JZ         AB01    74      ;JUMP BY <DISPLACEMENT> IF ZERO
2940 JNGE       AB01    7C      ;JUMP BY <DISPLACEMENT> IF NOT GREATER OR EQUAL
2950 JNG        AB01    7E      ;JUMP BY <DISPLACEMENT> IF NOT GREATER
2960 JNAE       AB01    72      ;JUMP BY <DISPLACEMENT> IF NOT ABOVE OR EQUAL
2970 JNA        AB01    76      ;JUMP BY <DISPLACEMENT> IF NOT ABOVE
2980 JPE        AB01    7A      ;JUMP BY <DISPLACEMENT> IF PARITY EVEN
2990 JNEG       AB01    78      ;JUMP BY <DISPLACEMENT> IF NEGATIVE
3000                            ;
3010 JNE        AB01    75      ;JUMP BY <DISPLACEMENT> IF NOT EQUAL
3020 JNL        AB01    7D      ;JUMP BY <DISPLACEMENT> IF NOT LESS
3030 JNLE       AB01    7F      ;JUMP BY <DISPLACEMENT> IF NOT LESS OR EQUAL
3040 JNB        AB01    73      ;JUMP BY <DISPLACEMENT> IF NOT BELOW
3050 JNBE       AB01    77      ;JUMP BY <DISPLACEMENT> IF NOT BELOW OR EQUAL
3060 JNP        AB01    7B      ;JUMP BY <DISPLACEMENT> IF NOT PARITY
3070 JNO        AB01    71      ;JUMP BY <DISPLACEMENT> IF NOT OVERFLOW
3080 JNS        AB01    79      ;JUMP BY <DISPLACEMENT> IF NOT SIGN
3090                            ;
3100 JNZ        AB01    75      ;JUMP BY <DISPLACEMENT> IF NOT ZERO
3110 JGE        AB01    7D      ;JUMP BY <DISPLACEMENT> IF GREATER OR EQUAL
3120 JG         AB01    7F      ;JUMP BY <DISPLACEMENT> IF GREATER
3130 JAE        AB01    73      ;JUMP BY <DISPLACEMENT> IF ABOVE OR EQUAL
3140 JA         AB01    77      ;JUMP BY <DISPLACEMENT> IF ABOVE
3150 JPO        AB01    7B      ;JUMP BY <DISPLACEMENT> IF PARITY ODD
3160 JPOS       AB01    79      ;JUMP BY <DISPLACEMENT> IF POSITIVE
3170                            ;
```

*(Continued on top of page 24)*

```
3730 ATAN       BB00    87
3740 LOG        BB00    88
3750 LN         BB00    89
3760 EXP        BB00    8A
3770 PWR        BB00    8B
3780                            ;
3790 FIXS       BB00    9F      ;CONVERT 32-BIT FLOATING POINT TO 16-BIT INTEGER
3800 FIXD       BB00    9E      ;CONVERT 32-BIT FLOATING POINT TO 32-BIT INTEGER
3810                            ;
3820 FLTS       BB00    9D      ;CONVERT 16-BIT INTEGER TO 32-BIT FLOATING POINT
3830 FLTD       BB00    9C      ;CONVERT 32-BIT INTEGER TO 32-BIT FLOATING POINT
3840                            ;
3850 CHSS       BB00    F4      ;CHANGE SIGN OF 16-BIT INTEGER
3860 CHSD       BB00    B4      ;CHANGE SIGN OF 32-BIT INTEGER
3870 CHSF       BB00    95      ;CHANGE SIGN OF 32-BIT FLOATING POINT
3880                            ;
3890 PSHS       BB00    F7      ;PUSH DATA ITEM ONTO AMD 9511 STACK
3900 PSHD       BB00    B7
3910 PSHF       BB00    97
3920                            ;
3930 PULS       BB00    F8      ;PULL DATA ITEM FROM AMD 9511 STACK
3940 PULD       BB00    B8



3940 PULD       BB00    B8
3950 PULF       BB00    98
3960                            ;
3970 XCHS       BB00    F9      ;EXCHANGE TOP TWO DATA ITEMS ON AMD 9511 STACK
3980 XCHD       BB00    B9
3990 XCHF       BB00    99
4000                            ;
4010 PI         BB00    9A      ;PUSH PI ONTO AMD 9511 STACK
4020                            ;
4030 *
4040 * PSEUDO-OPERATIONS (PSEUDOPS)
4050 *
4060 * NOTE THE UNIQUE PSEUDOP ", OR "DITTO". THIS PSEUDOP DUPLICATES
4070 * THE OPERATION OF THE LAST PSEUDO SPECIFICALLY IDENTIFIED (WORD,
4080 * BYTE, ETC.)  THUS, THE LONGER - BUT MORE ANNOYING TO TYPE - PSEUDOS
4090 * CAN BE COMBINED WITH A CONCISE OPERATOR FOR EASY TO UNDERSTAND
4100 * PROGRAMS.
4110 *
4120                            ;
4130 TYPE       1       1
4140                            ;
4150 "          00              ;REPEAT LAST PSEUDO WITH NEW VALUE
4160 END        01              ;END ASSEMBLY
4170 ORIGIN     02              ;DEFINE NEW ASSEMBLY COUNTER
4180 CODEMACRO  03              ;DEFINE MACRO
4190 ENDMACRO   04              ;END MACRO DEFINITION
4200 EQUATE     05              ;DEFINE SYMBOL
4210 OPMACRO    06              ;DEFINE OPERAND MACRO
4220 ALLOCATE   07              ;RESERVE STORAGE
```

*(Continued on next page)*

# A Cross Assembler for the 8080 or Z80 (Continued)

```
4230 BYTE       08              ;DEFINE BYTE
4240 PAIR       09              ;DEFINE PAIR OF WORDS
4250 FILL       0A              ;FILL MEMORY WITH VALUE
4260 TEXT       0B              ;DEFINE TEXT
4270 WORD       0C              ;DEFINE WORD
4280 LINK       0D              ;LINK TO ASSEMBLE PROGRAM
4290                            ;
4300                            ;
4310                            ;
4320                            ;
4330
```

**end**

## Listing II

The first installment of the XASM-86 Z80-to-8086 Cross-Assembler. The XASM-86 error messages are included in this listing. The modules, identifiable by the sequential line numbers, in order of appearance, are:

| | |
|---|---|
| .ERRORS | XASM-86 error handling |
| .PRIMITIVS | First group of routines in library. |

It is mandatory to call these modules by the assigned names, since they are called by the linker for the assembly of XASM-86.

```
** . .
0000 ********************************************************************************
0010 *
0020 *  8 JAN 81 - NOTE THAT THE ERROR BUFFER CONTAINS UP TO 17 ERROR
0030 *             CODES FOR A SINGLE TEXT LINE.  TO PRINT A LIST OF
0040 *             ERRORS AND LINE NUMBERS - ONLY - WOULD REQUIRE A PASS
0050 *             THROUGH THE WORK FILE, TRANSLATING ERROR CODES
0060 *             TO ERROR MESSAGES.
0070 *  1 FEB 81 - ADD MODEM ERRORS.
0080 *
0090 * ERRORS ADDS AN ERROR CODE TO THE SEQUENTIAL ERROR RECORD BUFFER,
0100 * AND INCREMENTS THE BUFFER COUNT.  ERROR CODES ARE PROVIDED
0110 * FOR ALL PHASES AND FUNCTIONS OF THE ASSEMBLY, THUS ALLOWING USERS
0120 * TO PIN-POINT THE FAULTS IN THEIR ASSEMBLY LANGUAGE PROGRAMS.
0130 * OF MAJOR IMPORTANCE ARE THE ERROR MESSAGES.  ALTHOUGH THEY ADD
0140 * TO THE LENGTH OF XASM, THE MESSAGES GIVE INSTANTLY UNDERSTANDABLE
0150 * EXPLANATIONS OF ERRORS.
0160 *
0170 * ERRORS IS RECURSIVE: IT CALLS ITSELF WHEN ONLY 1 BYTE IS LEFT IN
0180 * THE ERROR BUFFER, & FILLS IT WITH A MESSAGE INDEX THAT DENOTES THE
0190 * BUFFER AS BEING FULL.
0200 *
0210 * ERRORS EXPECTS: ERRFLAG     = 0 = DEFAULT TO NO ERRORS FOUND
0220 *                 A           = ERROR MESSAGE INDEX
0230 *
0240 * ERRORS RETURNS: ERRBUFFER   = ERRBUFFER PLUS NEW ERROR CODE
0250 *                 ERRBUFPTR   = ERRBUFPTR + 1
0260 *                 ERRCNT      = ERRCNT + 1
0270 *                 ERRFLAG     = 1 = ERROR FOUND
0280 *
0290 * ERRORS SAVES ALL REGISTERS AND FLAGS
0300 *
0310
```

```
0860 MS201      DTH     'LINENUMBER OR OPCODE INVALID'
0870 MS202      DTH     'OPERAND(S) OR PARAMETER(S) INVALID'
0880 MS203      DTH     'NO MORE THAN 8 PARAMETERS ALLOWED IN LINE'
0890 MS204      DTH     'OPCODE INVALID'
0900                    ;
0910 MS301      DTH     'OPCODE NOT IN SYMBOL TABLE'
0920 MS302      DTH     'OPCODE FIELD MUST BE OPCODE, MACRO OR PSEUDO-OP'
0930 MS303      DTH     'OPCODE IS INVALID OR INCORRECTLY-SPELLED'
0940 MS304      DTH     'OPCODE FIELD MUST BE OPCODE OR MACRO'
0950                    ;
0960 MS402      DTH     'DUPLICATE LABEL'
0970                    ;
0980 MS501      DTH     'CODE ASSEMBLED BUT OPERAND(S) MISSING'
0990 MS502      DTH     'CODE ASSEMBLED BUT EXTRA OPERAND(S) IGNORED'
1000 MS503      DTH     'OPERAND SYNTAX IS INVALID'
1010 MS504      DTH     'OPERAND MAY NOT INCLUDE @M'
1020 MS505      DTH     'OPERAND MUST BE @M'
1030 MS506      DTH     'OPERAND MUST BE DISP@M'
1040 MS507      DTH     'OPERAND MUST BE DISP OR DATA'
1050 MS508      DTH     'COMPLEX EXPRESSION OPERATOR MUST BE + OR -'
1060 MS509      DTH     'OPCODE OR PSEUDO-OP MAY NOT BE OPERAND'
1070 MS510      DTH     'OPERAND MUST BE DATA'
1080 MS511      DTH     'OPERAND MUST BE DISP'
1090 MS512      DTH     'OPERAND MUST BE CL OR 1'
1100 MS513      DTH     'OPERAND MUST BE SEG'
1110 MS514      DTH     'OPERAND MUST BE REG16,SEG,DISP,OR @M'
1120 MS515      DTH     'OPERAND MUST BE REG16,DISP OR @M'
1130 MS516      DTH     'OPERAND MUST BE REG8,DISP OR @M'
1140 MS517      DTH     'ONLY ONE OPERAND MAY INCLUDE OR BE DISP'
1150 MS518      DTH     'DISP < -126    OR    > +129'
1160 MS521      DTH     'TOO MANY PARAMETERS FOR OPCODE'
1170 MS523      DTH     'OPERAND CHOSEN NOT VALID AS DISP'
1180                    ;
1190 MS601      DTH     '2ND OPERAND MAY NOT BE SEG'
1200 MS602      DTH     '2ND OPERAND MAY NOT BE @M'
1210 MS603      DTH     '2ND OPERAND MAY NOT BE DISP'
1220 MS604      DTH     '2ND OPERAND MAY NOT BE DISP@M'
1230 MS605      DTH     '2ND OPERAND MAY NOT BE DATA'
1240 MS606      DTH     '2ND OPERAND INVALID'
1250 MS607      DTH     'DISP OR DATA VALUE NOT FOUND'
1260 MS608      DTH     'DISP OR DATA LENGTH EXCEEDS BYTE(S) ALLOTTED'
1270                    ;
1280 MS701      DT      'XASM COMMAND FORMAT INCORRECT, MUST MATCH:'
1290            DB      [CR]
1300            DB      [LF]
1310            DT      'XASM "[drive:]sourcefilename"'
1320            DTH     '"[drive:]objectfilename" "[C][E][W]"'
1330                    ;

1340 MS702      DTH     'UNIT SPECIFIED MUST BE 0 OR 1'
1350 MS703      DTH     'CHARACTERS ALLOWED IN OPTION LIST ARE C,E, AND W'
1360                    ;
1370 MS801      DTH     'SYMBOL TABLE FULL'
1380 MS802      DTH     'SYMBOL TABLE EMPTY'
```

```
0320                                    ;
0330                                    ;
0340                                    ;
0350                                    ;
0360                                    ;
0370 ERRORS         PUSH    H           ;SAVE USED REGS & FLAGS
0380                PUSH    PSW
0390                                    ;
0400                LDA     ERRCNT
0410                CPI     17          ;IF ERRCNT = 17, THEN BUFFER IS FULL
0420                JZ      EXITERRORS  ;DO NOT ADD ERROR CODE TO RECORD
0430                                    ;
0440                                    ;
0450                LHLD    ERRBUFPTR
0460                POP     PSW         ;GET ERROR CODE
0470                PUSH    PSW         ;RESTORE STACK FOR NORMAL EXIT
0480                MOV     M,A         ;ADD CODE TO ERROR RECORD IN ERRBUFFER
0490                INX     H           ;POINT TO NEXT BYTE IN ERROR BUFFER
0500                SHLD    ERRBUFPTR   ;SAVE POINTER TO NEXT ERROR CODE
0510                                    ;
0520                LDA     ERRCNT      ;INCREMENT ERRBUFFER LENGTH
0530                INR     A
0540                STA     ERRCNT
0550                                    ;
0560                CPI     16          ;IF ERRCNT = 16, ONLY 1 BYTE FREE
0570                CZ      ERR811      ;SO ERRBUFFER FULL FOR THIS LINE
0580                                    ;
0590                MVI     A,1         ;INDICATE ERROR FOUND
0600                STA     ERRFLAG
0610                                    ;
0620 EXITERRORS     POP     PSW
0630                POP     H
0640                RET
0650                                    ;
0660                                    ;
0670                                    ;
0680                                    ;;
0690                                    ;
0700 ERR811         MVI     A,E811      ;ERROR - ERRBUFFER FULL
0710                CALL    ERRORS
0720                RET
0730                                    ;
0740                                    ;
0750                                    ;
0760                                    ;
0770 *
0780 * TABLE OF ERROR MESSAGES
0790 *
0800                                    ;
0810 MS000          DTH     'Z-88 XASM (C) 1980 BY PROGRAMMERS PUBLISHING CO.'
0820                                    ;
0830 MS101          DTH     'PREMATURE END-OF-SOURCEFILE'
0840 MS102          DTH     'PREMATURE END OF MACRO EXPANSION TEXT'
0850                                    ;
```

*(Continued on top of page 26)*

```
1390 MS803          DTH     'ERROR IN .PRIMEVAL LOGIC'
1400 MS804          DTH     'ERROR IN OPCODE SYMBOL TABLE RECORD'
1410 MS805          DTH     'ERROR IN .PARSETEXT LOGIC'
1420 MS806          DTH     'ERROR IN ASSEMBLY CONTROL WORD'
1430 MS807          DTH     'PARAMETER INDEX <1 OR >8'
1440 MS808          DTH     'NO MASK BYTES IN OPCODE SYMBOL TABLE RECORD'
1450 MS809          DTH     'ERROR TRAP FAILURE FOUND DURING .BUILDCODE'
1460 MS810          DTH     'ERROR TRAP FAILURE FOUND DURING .OPERANDS'
1470 MS811          DTH     'ERROR BUFFER FULL'
1480 MS812          DTH     'PRINTER MAY NOT BE  INPUT DEVICE'
1490 MS813          DTH     'TRANSIENT MEMORY ERROR DURING IO'
1500 MS814          DTH     'MEMORY IS ROM, DEFECTIVE, OR NON-EXISTENT'
1510                                    ;
1520 MS901          DTH     'ILLEGAL FILENUMBER - SOURCE FILE'
1530 MS902          DTH     'ILLEGAL FILETYPE - SOURCE FILE'
1540 MS903          DTH     'INPUT ERROR - MEMORY - TEXTBUFFER'
1550 MS904          DTH     'OUTPUT ERROR - DISK - SOURCE FILE'
1560 MS905          DTH     'ILLEGAL FILENUMBER - WORK FILE'
1570 MS906          DTH     'ILLEGAL FILETYPE - WORK FILE'
1580 MS907          DTH     'INPUT ERROR - MEMORY - WORKBUFFER'
1590 MS908          DTH     'OUTPUT ERROR - DISK - WORK FILE'
1600 MS909          DTH     'INPUT ERROR - DISK - WORK FILE'
1610 MS910          DTH     'OUTPUT ERROR - MEMORY - WORKBUFFER'
1620 MS911          DTH     'INPUT ERROR - DISK - SOURCE FILE'
1630 MS912          DTH     'OUTPUT ERROR - MEMORY - TEXTBUFFER'
1640 MS913          DTH     'ILLEGAL FILENUMBER - OBJECT FILE'
1650 MS914          DTH     'ILLEGAL FILETYPE - OBJECT FILE'
1660 MS915          DTH     'INPUT ERROR - MEMORY - OBJECTBUFFER'
1670 MS916          DTH     'OUTPUT ERROR - DISK - OBJECT FILE'
1680 MS917          DTH     'ILLEGAL FILENUMBER - PRINT FILE'
1690 MS918          DTH     'ILLEGAL FILETYPE - PRINT FILE'
1700 MS919          DTH     'OUTPUT ERROR - PRINTER - PRINT FILE'
1710 MS920          DTH     'CREATE ERROR - DISK - WORK FILE'
1720 MS921          DTH     'CREATE ERROR - DISK - OBJECT FILE'
1730 MS922          DTH     'ERROR - XASM MAY NOT CREATE DISK SOURCE FILE'
1740 MS923          DTH     'OPEN ERROR - DISK - WORK FILE'
1750 MS924          DTH     'OPEN ERROR - DISK - OBJECT FILE'
1760 MS925          DTH     'OPEN ERROR - DISK - SOURCE FILE'
1770 MS926          DTH     'CLOSE ERROR - DISK - WORK FILE'
1780 MS927          DTH     'CLOSE ERROR - DISK - OBJECT FILE'
1790 MS928          DTH     'CLOSE ERROR - DISK - SOURCE FILE'
1800 MS929          DTH     'DELETE ERROR - DISK - WORK FILE'
1810 MS930          DTH     'DELETE ERROR - DISK - OBJECT FILE'
1820 MS931          DTH     'ERROR - XASM MAY NOT DELETE DISK SOURCE FILE'
1830 MS932          DTH     'ERROR IN CREATE UTILITY LOGIC'
1840 MS933          DTH     'ERROR IN OPEN UTILITY LOGIG'
1850 MS934          DTH     'ERROR IN CLOSE UTILITY LOGIC'
1860 MS935          DTH     'ERROR IN DELETE UTILITY LOGIC'
1870 MS936          DTH     'ILLEGAL FILENUMBER - SYMBOL TABLE FILE'
1880 MS937          DTH     'ILLEGAL FILETYPE - SYMBOL TABLE FILE'
1890 MS938          DTH     'CREATE ERROR - DISK - SYMBOL TABLE FILE'
1900 MS939          DTH     'OPEN ERROR - DISK - SYMBOL TABLE FILE'
1910 MS940          DTH     'CLOSE ERROR - DISK - SYMBOL TABLE FILE'
```

*(Continued on next page)*

# A Cross Assembler for the 8080 or Z80 (Continued)

```
1920 MS941          DTH     'DELETE ERROR - DISK - SYMBOL TABLE FILE'
1930 MS942          DTH     'INPUT ERROR - DISK - SYMBOL TABLE FILE'
1940 MS943          DTH     'OUTPUT ERROR - DISK - SYMBOL TABLE FILE'
1950 MS944          DTH     'OUTPUT ERROR - MEMORY - OBJECT BUFFER'
1960 MS950          DTH     'INTERRUPTED FILE LOAD ERROR'
1970 MS951          DTH     'INTERRUPTED FILE STORE ERROR'
1980 MS960          DTH     'DIAL TONE NOT DETECTED WITHIN 15 SEC'
1990 MS961          DTH     'PHONE NUMBER MUST CONTAIN 7,8, OR 11 DIGITS'
2000 MS962          DTH     'ILLEGAL DIGIT IN PHONE NUMBER'
2010 MS963          DTH     'CONNECTION NOT ESTABLISHED WITHIN 30 SEC'
2010 MS963          DTH     'CONNECTION NOT ESTABLISHED WITHIN 30 SEC'
2020                                        ;
2030 *
2040 * TABLE OF ADDRESSES OF ERROR MESSAGES
2050 *
2060                                        ;
2070 ERRBASE        DW      MS000
2080 X101           DW      MS101
2090 X102           DW      MS102
2100                                        ;
2110 X201           DW      MS201
2120 X202           DW      MS202
2130 X203           DW      MS203
2140 X204           DW      MS204
2150                                        ;
2160 X301           DW      MS301
2170 X302           DW      MS302
2180 X303           DW      MS303
2190 X304           DW      MS304
2200                                        ;
2210 X402           DW      MS402
2220                                        ;
2230 X501           DW      MS501
2240 X502           DW      MS502
2250 X503           DW      MS503
2260 X504           DW      MS504
2270 X505           DW      MS505
2280 X506           DW      MS506
2290 X507           DW      MS507
2300 X508           DW      MS508
2310 X509           DW      MS509
2320 X510           DW      MS510
2330 X511           DW      MS511
2340 X512           DW      MS512
2350 X513           DW      MS513
2360 X514           DW      MS514
2370 X515           DW      MS515
2380 X516           DW      MS516
2390 X517           DW      MS517
2400 X518           DW      MS518
2410 X521           DW      MS521
2420 X523           DW      MS523
2430                                        ;
2440 X601           DW      MS601
2980 X927           DW      MS927
2990 X928           DW      MS928
3000 X929           DW      MS929
3010 X930           DW      MS930
3020 X931           DW      MS931
3030 X932           DW      MS932
3040 X933           DW      MS933
3050 X934           DW      MS934
3060 X935           DW      MS935
3070 X936           DW      MS936
3080 X937           DW      MS937
3090 X938           DW      MS938
3100 X939           DW      MS939
3110 X940           DW      MS940
3120 X941           DW      MS941
3130 X942           DW      MS942
3140 X943           DW      MS943
3150 X944           DW      MS944
3160 X950           DW      MS950
3170 X951           DW      MS951
3180 X960           DW      MS960
3190 X961           DW      MS961
3200 X962           DW      MS962
3210 X963           DW      MS963
3220                                        ;
3230 *
3240 * ERROR CODES EQUATED TO ERROR MESSAGE TABLE INDICES
3250 *
3260                                        ;
3270 E000           EQU     ERRBASE-ERRBASE
3280                                        ;
3290 E101           EQU     X101-ERRBASE
3300 E102           EQU     X102-ERRBASE
3310                                        ;
3320 E201           EQU     X201-ERRBASE
3330 E202           EQU     X202-ERRBASE
3340 E203           EQU     X203-ERRBASE
3350 E204           EQU     X204-ERRBASE
3360                                        ;
3370 E301           EQU     X301-ERRBASE
3380 E302           EQU     X302-ERRBASE
3390 E303           EQU     X303-ERRBASE
3400 E304           EQU     X304-ERRBASE
3410                                        ;
3420 E402           EQU     X402-ERRBASE
3430                                        ;
3440 E501           EQU     X501-ERRBASE
3450 E502           EQU     X502-ERRBASE
3460 E503           EQU     X503-ERRBASE
3470 E504           EQU     X504-ERRBASE
3480 E505           EQU     X505-ERRBASE
3490 E506           EQU     X506-ERRBASE
3500 E507           EQU     X507-ERRBASE
3510 E508           EQU     X508-ERRBASE
```

```
2450 X602       DW      MS602
2460 X603       DW      MS603
2470 X604       DW      MS604
2480 X605       DW      MS605
2490 X606       DW      MS606
2500 X607       DW      MS607
2510 X608       DW      MS608
2520
2530 X701       DW      MS701
2540 X702       DW      MS702
2550 X703       DW      MS703
2560
2570 X801       DW      MS801
2580 X802       DW      MS802
2590 X803       DW      MS803
2600 X804       DW      MS804
2610 X805       DW      MS805
2620 X806       DW      MS806
2630 X807       DW      MS807
2640 X808       DW      MS808
2650 X809       DW      MS809
2660 X810       DW      MS810
2670 X811       DW      MS811
2680 X812       DW      MS812
2690 X813       DW      MS813
2700 X814       DW      MS814
2710
2720 X901       DW      MS901
2730 X902       DW      MS902
2740 X903       DW      MS903
2750 X904       DW      MS904
2760 X905       DW      MS905
2770 X906       DW      MS906
2780 X907       DW      MS907
2790 X908       DW      MS908
2800 X909       DW      MS909
2810 X910       DW      MS910
2820 X911       DW      MS911
2830 X912       DW      MS912
2840 X913       DW      MS913
2850 X914       DW      MS914
2860 X915       DW      MS915
2870 X916       DW      MS916
2880 X917       DW      MS917
2890 X918       DW      MS918
2900 X919       DW      MS919
2910 X920       DW      MS920
2920 X921       DW      MS921
2930 X922       DW      MS922
2940 X923       DW      MS923
2950 X924       DW      MS924
2960 X925       DW      MS925
2970 X926       DW      MS926
```

*(Continued on top of page 28)*

```
3520 E509       EQU     X509-ERRBASE
3530 E510       EQU     X510-ERRBASE
3540 E511       EQU     X511-ERRBASE
3550 E512       EQU     X512-ERRBASE
3560 E513       EQU     X513-ERRBASE
3570 E514       EQU     X514-ERRBASE
3580 E515       EQU     X515-ERRBASE
3590 E516       EQU     X516-ERRBASE
3600 E517       EQU     X517-ERRBASE
3610 E518       EQU     X518-ERRBASE
3620 E521       EQU     X521-ERRBASE
3630 E523       EQU     X523-ERRBASE
3640                                    ;
3650 E601       EQU     X601-ERRBASE
3660 E602       EQU     X602-ERRBASE
3670 E603       EQU     X603-ERRBASE
3680 E604       EQU     X604-ERRBASE
3690 E605       EQU     X605-ERRBASE
3700 E606       EQU     X606-ERRBASE
3710 E607       EQU     X607-ERRBASE
3720 E608       EQU     X608-ERRBASE
3730                                    ;
3740 E701       EQU     X701-ERRBASE
3750 E702       EQU     X702-ERRBASE
3760 E703       EQU     X703-ERRBASE
3770                                    ;
3780 E801       EQU     X801-ERRBASE
3790 E802       EQU     X802-ERRBASE
3800 E803       EQU     X803-ERRBASE
3810 E804       EQU     X804-ERRBASE
3820 E805       EQU     X805-ERRBASE
3830 E806       EQU     X806-ERRBASE
3840 E807       EQU     X807-ERRBASE
3850 E808       EQU     X808-ERRBASE
3860 E809       EQU     X809-ERRBASE
3870 E810       EQU     X810-ERRBASE
3880 E811       EQU     X811-ERRBASE
3890 E812       EQU     X812-ERRBASE
3900 E813       EQU     X813-ERRBASE
3910 E814       EQU     X814-ERRBASE
3920                                    ;
3930 E901       EQU     X901-ERRBASE
3940 E902       EQU     X902-ERRBASE
3950 E903       EQU     X903-ERRBASE
3960 E904       EQU     X904-ERRBASE
3970 E905       EQU     X905-ERRBASE
3980 E906       EQU     X906-ERRBASE
3990 E907       EQU     X907-ERRBASE
4000 E908       EQU     X908-ERRBASE
4010 E909       EQU     X909-ERRBASE
4020 E910       EQU     X910-ERRBASE
4030 E911       EQU     X911-ERRBASE
4040 E912       EQU     X912-ERRBASE
```

*(Continued on next page)*

# A Cross Assembler for the 8080 or Z80 (Continued)

```
4050 E913          EQU    X913-ERRBASE
4060 E914          EQU    X914-ERRBASE
4070 E915          EQU    X915-ERRBASE
4080 E916          EQU    X916-ERRBASE
4090 E917          EQU    X917-ERRBASE
4100 E918          EQU    X918-ERRBASE
4110 E919          EQU    X919-ERRBASE
4120 E920          EQU    X920-ERRBASE
4130 E921          EQU    X921-ERRBASE
4140 E922          EQU    X922-ERRBASE
4150 E923          EQU    X923-ERRBASE
4160 E924          EQU    X924-ERRBASE
4170 E925          EQU    X925-ERRBASE
4180 E926          EQU    X926-ERRBASE
4190 E927          EQU    X927-ERRBASE
4200 E928          EQU    X928-ERRBASE
4210 E929          EQU    X929-ERRBASE
4220 E930          EQU    X930-ERRBASE
4230 E931          EQU    X931-ERRBASE
4240 E932          EQU    X932-ERRBASE
4250 E933          EQU    X933-ERRBASE
4260 E934          EQU    X934-ERRBASE
4270 E935          EQU    X935-ERRBASE
4280 E936          EQU    X936-ERRBASE
4290 E937          EQU    X937-ERRBASE
4300 E938          EQU    X938-ERRBASE
4310 E939          EQU    X939-ERRBASE
4320 E940          EQU    X940-ERRBASE
4330 E941          EQU    X941-ERRBASE
4340 E942          EQU    X942-ERRBASE
4350 E943          EQU    X943-ERRBASE
4360 E944          EQU    X944-ERRBASE
4370 E950          EQU    X950-ERRBASE
4380 E951          EQU    X951-ERRBASE
4390 E960          EQU    X960-ERRBASE
4400 E961          EQU    X961-ERRBASE
4410 E962          EQU    X962-ERRBASE
4420 E963          EQU    X963-ERRBASE
4430
4440
```

**end**

## Listing III

```
**
0000 ***********************************************************************
0010 *
0020 * GETOKEN PARSES A TEXT STRING FOR A SUB-STRING (TOKEN).
0030 * A TOKEN IS OF THE FORMAT:
0040 *
0580                RET
0590                                        ;
0600 GETOKEN2       MOV    D,H              ;SAVE POINTER TO FIRST TOKEN CHARACTER
0610                MOV    E,L
0620                                        ;
0630 GETOKEN3       MVI    A,[CR]
0640                CMP    M
0650                JZ     RESULT01         ;TOKEN TERMINATED BY [CR]
0660                                        ;
0670                LDA    MATCH1
0680                CMP    M
0690                JZ     RESULT02         ;TOKEN TERMINATED BY (MATCH1)
0700                                        ;
0710                LDA    MATCH2
0720                CMP    M
0730                JZ     RESULT03         ;TOKEN TERMINATED BY (MATCH2)
0740                                        ;
0750                INX    H                ;POINT TO NEXT CHARACTER
0760                INR    B                ;INCREMENT TOKEN COUNT
0770                DCR    C                ;DECREMENT CHARACTER COUNTER
0780                JNZ    GETOKEN3         ;SCAN TOKEN STRING UNTIL END
0790                MVI    A,05H            ;C=0 DURING TOKEN SCAN
0800                RET
0810                                        ;
0820 RESULT00       XRA    A                ;TOKEN IS COMMENT STRING
0830                DCR    C                ;COMMENT LENGTH - [CR]
0840                MOV    B,C              ;SAVE LENGTH OF COMMENT
0850                MOV    D,H              ;SAVE POINTER TO COMMENT
0860                MOV    E,L
0870                RET
0880                                        ;
0890 RESULT01       MVI    A,1              ;TOKEN TERMINATED BY [CR]
0900                INX    H
0910                DCR    C
0920                RET
0930                                        ;
0940 RESULT02       MVI    A,2              ;TOKEN TERMINATED BY (MATCH1)
0950                INX    H
0960                DCR    C
0970                RET
0980                                        ;
0990 RESULT03       MVI    A,3              ;TOKEN TERMINATED BY (MATCH2)
1000                INX    H
1010                DCR    C
1020                RET
1030                                        ;
1040 RESULT04       MVI    A,04H            ;[CR] FOUND BEFORE TOKEN
1050                INX    H
1060                DCR    C
1070                RET
1080                                        ;
1090 ***********************************************************************
1100 *
```

```
0050 *     [*].................<CHARACTERS>...........[CARRIAGE RETURN]
0060 *     [;]                                       [MATCH-CHARACTER-1]
0070 *     [NOT-SKIP-CHARACTER]                      [MATCH-CHARACTER-2]
0080 *     [CARRIAGE-RETURN] (ERROR IF FIRST CHARACTER)
0090 *
0100 * WHERE THE FIRST CHARACTER MAY BE ANY OF THOSE LISTED IN COMBINATION
0110 * WITH ANY OF THE SECOND CHARACTERS LISTED
0120 *
0130 *
0140 * GETOKEN EXPECTS: A POINTER TO A TEXT LINE IN HL
0150 *                  NUMBER OF CHARACTERS IN THE TEXTLINE IN C
0160 *                  A VALID TEXT LINE (LINEEDIT FORMAT) AT (HL)
0170 *                  A SKIP CHARACTER IN (SKIP)
0180 *                  A MATCH CHARACTER IN (MATCH1)
0190 *                  A MATCH CHARACTER IN (MATCH2)
0200 *
0210 * GETOKEN RETURNS: A POINTER TO THE FIRST CHARACTER OF THE TOKEN IN DE
0220 *                  NUMBER OF CHARACTERS IN THE TOKEN IN B
0230 *                  C = C-(NUMBER OF CHARACTERS SEARCHED)
0240 *                  HL = HL+(NUMBER OF CHARACTERS SEARCHED)
0250 *                  RESULT OF THE SEARCH IN A
0260 *
0270 * THE RESULT CODE IS: 04H  [CR] OR (C=0) BEFORE TOKEN FOUND
0280 *                     05H  (C=0) DURING TOKEN
0290 *                     00H  [*] OR [;] STARTS TOKEN (TOKEN IS COMMENT)
0300 *                     01H  [CR] TERMINATES TOKEN
0310 *                     02H  [MATCH1] TERMINATES TOKEN
0320 *                     03H  [MATCH2] TERMINATES TOKEN
0330 *
0340 *

0350                                    ;
0360 GETOKEN        MVI     B,0         ;TOKEN COUNT = 0
0370                                    ;
0380 GETOKEN1       MVI     A,[CR]
0390                CMP     M
0400                JZ      RESULT04    ;[CR] FOUND BEFORE TOKEN
0410                                    ;
0420                MVI     A,[ASTERISK]
0430                CMP     M
0440                JZ      RESULT00    ;TOKEN IS COMMENT STRING
0450                                    ;
0460                MVI     A,[SEMICOLON]
0470                CMP     M
0480                JZ      RESULT00    ;TOKEN IS COMMENT STRING
0490                                    ;
0500                LDA     SKIP
0510                CMP     M
0520                JNZ     GETOKEN2    ;TOKEN FOUND
0530                                    ;
0540                INX     H           ;POINT TO NEXT CHARACTER
0550                DCR     C           ;DECREMENT CHARACTER COUNT
0560                JNZ     GETOKEN1    ;CONTINUE SEARCH FOR TOKEN
0570                MVI     A,04H       ;C=0 BEFORE TOKEN FOUND
```

*(Continued on top of page 30)*

```
1110 * JMPXI (JUMP-INDEXED-INDIRECT) PERFORMS AN INDIRECT JUMP TO AN ADDRESS
1120 * FROM ADDRESS BLOCK INDEXED BY THE CONTENTS OF ACCUMULATOR.
1130 * JMPXI DOES NOT MODIFY ANY REGISTERS. ADDRESS BLOCK MAY BE UP TO 512
1140 * BYTES IN LENGTH (256 ADDRESS PAIRS).
1150 *
1160 * JMPXI EXPECTS: AN INDEX IN A
1170 *                A TABLE OF ADDRESS PAIRS IN LO-HI FORMAT AT ADDRESS BLOCK
1180 *                THE ADDRESS BLOCK TO IMMEDIATELY FOLLOW THE CALL TO JMPXI
1190 *
1200 * JMPXI RETURNS NOTHING.  CONTROL TRANSFERS TO THE ROUTINE LOCATED AT
1210 * (ADDRESS BLOCK+(2*ACCUMULATOR))
1220 *
1230                                    ;
1240 JMPXI          XTHL                ;ADDRESS BLOCK POINTER IN HL
1250                PUSH    D           ;HL & DE NOW SAVED
1260                MOV     E,A
1270                MVI     D,0         ;INDEX TO TABLE IN DE
1280                DAD     D
1290                DAD     D           ;HL POINTS TO ADDRESS PAIR
1300                MOV     E,M
1310                INX     H
1320                MOV     D,M         ;JUMP ADDRESS NOW IN DE
1330                POP     H           ;OLD DE NOW IN HL
1340                XCHG                ;DE RESTORED, JUMP ADDRESS IN HL
1350                XTHL                ;HL RESTORED, JUMP ADDRESS ON STACK
1360                RET                 ;JUMP TO ROUTINE
1370                                    ;

1380 ***************************************************************************
1390 *
1400 * COMPSTRINGS COMPARES TWO STRINGS TO DETERMINE WHETHER THE IN-MEMORY
1410 * (OR "HL" STRING) IS LESS, EQUAL, OR GREATER THAN THE BUFFER
1420 * (OR "DE" STRING).  THE COMPARISON PROCEEDS CHARACTER-BY-CHARACTER
1430 * UNTIL EITHER THERE IS A DIFFERENCE, OR ONE STRING ENDS.  IN THE
1440 * SECOND CASE, THE STRING WITH MORE CHARACTERS IS "GREATER". A TABLE
1450 * OF POSSIBLE STRING COMBINATIONS & RESULTS FOLLOWS:

1460 *
1470 *              LENGTH:  HL < DE      HL = DE          HL > DE
1480 * CHARACTER    ------------------------------------------------
1490 *               |                 |                |              |
1500 * HL < DE      |     LESS        |     LESS       |    LESS      |
1510 *               |                 |                |              |
1520 * HL = DE      |     LESS        |     EQUAL      |   GREATER    |
1530 *               |                 |                |              |
1540 * HL > DE      |   GREATER       |    GREATER     |   GREATER    |
1550 *               ------------------------------------------------

1560 *
1570 * COMPSTRINGS EXPECTS: B  = BUFFER STRING COUNT (DE)
1580 *                      C  = IN-MEMORY STRING COUNT (HL)
1590 *                      DE = BUFFER STRING POINTER
```

*(Continued on next page)*

# A Cross Assembler for the 8080 or Z80 (Continued)

```
1600 *                      HL = MEMORY STRING POINTER
1610 *
1620 * IT RETURNS:          A  = RESULT CODE: 0   =   HL < DE (LESS)
1630 *                                        1   =   HL = DE (EQUAL)
1640 *                                        2   =   HL > DE (GREATER)
1650 *                      B  = COUNT - # CHARACTERS COMPARED
1660 *                      C  = COUNT - # CHARACTERS COMPARED
1670 *                      DE = POINTER + CHARACTERS EQUAL TO HL STRING
1680 *                      HL = POINTER + CHARACTERS EQUAL TO DE STRING
1690 *
1700                                    ;
1710 COMPSTRINGS    LDAX   D            ;BUFFER STRING CHARACTER
1720                CMP    M            ;COMPARE MEMORY-TO-BUFFER
1730                JZ     CHAREQ       ;CHARACTERS EQUAL. CHECK STRING LENGTH
1740                JC     GREATER      ;HL > DE
1750                JMP    LESS         ;HL < DE
1760                                    ;
1770 CHAREQ         INX    H            ;POINT TO NEXT MEMORY CHARACTER
1780                INX    D            ;POINT TO NEXT BUFFER CHARACTER
1790                DCR    B
1800                JZ     LENEQ        ;CHECK FOR EQUAL STRING LENGTHS
1810                                    ;
1820                DCR    C
1830                JNZ    COMPSTRINGS  ;COMPARE REMAINING CHARACTERS
1840                                    ;
1850 LESS           XRA    A            ;HL < DE  (CHARACTERS 1ST, LENGTH 2ND)
1860                RET
1870                                    ;
1880 LENEQ          DCR    C
1890                JZ     EQUAL        ;HL = DE
1900                                    ;
1910 GREATER        MVI    A,2          ;HL = DE (CHARACTERS 1ST, LENGTH 2ND)
1920                RET
1930                                    ;
1940 EQUAL          MVI    A,1          ;HL = DE (CHARACTERS AND LENGTH)
1950                RET
1960                                    ;
1970 ********************************************************************************
1980 *

1990 * SINXSYM SETS HL = (DESTPTR) + SYMBOL INDEX
2000 *
2010 * ALL "SINX" (SET-INDEX) COMMANDS ASSUME THAT THEY
2020 * WILL INDEX A RECORD IN XASM SYMBOL TABLE FORMAT. THIS FORMAT IS:
2030 *
2040 * BYTE NUMBER              MEANING
2050 * -----------     ------------------------------
2060 *     0                 SYMBOL COUNT
2070 *     1                 SYMBOL TYPE
2080 *     2                 VALUE COUNT
2090 *     3                 VALUE TYPE
2100 *    4-5              LCHILD IN LO-HI FORMAT
2110 *    6-7              RCHILD IN LO-HI FORMAT
2120 *
2630                                    ;
2640 ********************************************************************************
2650 *
2660 * SINXL SETS HL = (DESTPTR) + LCHILD INDEX
2670 *
2680                                    ;
2690 SINXL          LHLD   DESTPTR      ;GET DESTINATION POINTER
2700                INX    H
2710                INX    H
2720                INX    H
2730                INX    H
2740                RET
2750                                    ;

2760 ********************************************************************************
2770 *
2780 * SINXR SETS HL = (DESTPTR) + RCHILD INDEX
2790 *
2800                                    ;
2810 SINXR          LHLD   DESTPTR      ;GET DESTINATION INDEX
2820                INX    H
2830                INX    H
2840                INX    H
2850                INX    H
2860                INX    H
2870                INX    H
2880                RET
2890                                    ;

2900 ********************************************************************************
2910 *
2920 * MOVEFIELD TRANSFERS A VARIABLE LENGTH FIELD FROM (SOURCEPTR)
2930 * TO (DESTPTR) FOR A COUNT OF (SOURCECNT) OR UNTIL (DESTCNT)=0.
2940 *
2950 * MOVEFIELD EXPECTS: B  = SOURCECNT
2960 *                    C  = DESTCNT
2970 *                    DE = SOURCEPTR
2980 *                    HL = DESTPTR
2990 *
3000 * IT RETURNS: B  = SOURCECNT - # CHARACTERS MOVED
3010 *             C  = DESTCNT - # CHARACTERS MOVED
3020 *             DE = SOURCECNT + # CHARACTERS MOVED
3030 *             HL = DESTCNT + # CHARACTERS MOVED
3040 *             A  = RESULT CODE: 0   =   DESTFIELD < SOURCEFIELD
3050 *                               1   =   DESTFIELD = SOURCEFIELD
3060 *                               2   =   DESTFIELD > SOURCEFIELD
3070 *
3080 *

3090                                    ;
3100 MOVEFIELD      LDAX   D
3110                MOV    M,A          ;TRANSFER CHARACTER FROM (D) TO (H)
3120                INX    D
```

```
2130 *     8                        SYMBOL
2140 *     TO
2150 * 8+(SYMBOL COUNT - 1)
2160 *
2170 *  9+(SYMBOL COUNT - 1)          VALUE
2180 *     TO
2190 *  9+(SYMBOL COUNT - 1)
2200 *  +(VALUE COUNT - 1)
2210 *
2220 *
2230                                  ;
2240 SINXSYM        LHLD   DESTPTR    ;GET DESTINATION POINTER
2250                PUSH   D          ;SAVE DE
2260                LXI    D,8        ;INDEX SYMBOL
2270                DAD    D          ;POINT TO SYMBOL
2280                POP    D          ;RESTORE DE
2290                RET
2300                                  ;

2310 ************************************************************************
2320 *
2330 * SINXVAL SETS HL = (DESTPTR) + VALUE INDEX
2340 *
2350                                  ;
2360 SINXVAL        LHLD   DESTPTR    ;GET DESTINATION POINTER
2370                PUSH   D
2380                MOV    E,M
2390                MVI    D,0        ;SET INDEX PAST SYMBOL
2400                DAD    D
2410                MVI    E,8
2420                DAD    D          ;POINT TO VALUE
2430                POP    D
2440                RET
2450                                  ;

2460 ************************************************************************
2470 *
2480 * SINXS SETS HL = (DESTPTR) + SYMBOL (COUNT,TYPE) INDEX
2490 *
2500 * NOTE:  SINXS IS THE EQUIVALENT OF "LHLD DESTPTR".
2510 *        USE THAT EXPRESSION WHENEVER SINXS IS REQUIRED.
2520 *
2530 *
2540 ************************************************************************

2550 *
2560 * SINXU SETS HL = (DESTPTR) + VALUE (COUNT,TYPE) INDEX
2570 *
2580                                  ;
2590 SINXU          LHLD   DESTPTR    ;GET DESTINATION POINTER
2600                INX    H
2610                INX    H
2620                RET
```

*(Continued on top of page 32)*

```
3130                INX    H
3140                DCR    C          ;DESTCNT = DESTCNT - 1
3150                JZ     MOVEFIELD1 ;IF DESTCNT = 0 THEN DONE
3160                                  ;
3170                DCR    B          ;ELSE SOURCECNT = SOURCECNT - 1
3180                JNZ    MOVEFIELD  ;CONTINUE IF SOURCECNT NOT ZERO
3190                                  ;
3200                MVI    A,2        ;DESTFIELD > SOURCEFIELD

3210                RET
3220 MOVEFIELD1     DCR    B
3230                JZ     MOVEFIELD2
3240                                  ;
3250                XRA    A          ;DESTFIELD > SOURCEFIELD
3260                RET
3270                                  ;
3280 MOVEFIELD2     MVI    A,1        ;DESTFIELD = SOURCEFIELD
3290                RET
3300                                  ;

3310 ************************************************************************
3320 *
3330 * MOVEWORD TRANSFERS A TWO-BYTE WORD FROM SOURCEPTR TO DESTPTR
3340 *
3350 * IT EXPECTS:  DE = SOURCEPTR
3360 *              HL = DESTPTR
3370 *              (SOURCEPTR,SOURCEPTR+1) = WORD
3380 *
3390 * IT RETURNS:  DE = SOURCEPTR + 2
3400 *              HL = DESTPTR + 2
3410 *              (DESTPTR,DESTPTR+1) = (SOURCEPTR,SOURCEPTR+1)
3420 *

3430
3440 MOVEWORD       CALL   MOVEBYTE
3450 MOVEBYTE       LDAX   D
3460                MOV    M,A
3470                INX    D
3480                INX    H
3490                RET
3500

3510 ************************************************************************
3520 *
3530 * CALLI (CALL-INDIRECT) PERFORMS A JUMP TO A SUBROUTINE SPECIFIED BY
3540 * (ROUTINEPTR).  THUS, CALLING CALLI PERFORMS A TRUE SUBROUTINE CALL.
3550 * CALLI IS TRANSPARENT.  IT RETURNS VIA THE INDIRECT ROUTINE TO THE
3560 * INSTRUCTION IMMEDIATELY FOLLOWING THE "CALL CALLI" SEQUENCE.
3570 *
3580 * CALLI EXPECTS:  (ROUTINEPTR) = SUBROUTINE ADDRESS
3590 *
3600 * IT RETURNS NOTHING.
3610 *
```

*(Continued on next page)*

# A Cross Assembler for the 8080 or Z80 (Continued)

```
3620 * IT PRESERVES PSW,BC,DE,HL.
3630                                    ;
3640 CALLI          PUSH    H           ;SAVE HL ON STACK
3650                LHLD    ROUTINEPTR  ;GET SUBROUTINE ADDRESS IN HL
3660                XTHL                ;RESTORE HL, SUBR ADDRESS ON STACK
3670                RET                 ;JUMP TO SUBROUTINE
3680                                    ;


3690 ********************************************************************************
3700 *
3710 * MAKEHEX CONVERTS AN ASCII REPRESENTATION OF A HEXADECIMAL
3720 * NUMBER INTO ITS BINARY EQUIVALENT.  THE NUMBER MUST BE OF THE FORM
3730 * 0XXXXH, WHERE XXXX IS THE ASCII FORM OF THE NUMBER, AND MAY BE ZERO
3740 * TO FOUR DIGITS LONG.
3750 * THE NUMBER MUST BE PRECEEDED BY A "0", AND TERMINATE WITH THE HEX
3760 * IDENTIFIER, "H".
3770 *
3780 * MAKEHEX EXPECTS: (SOURCEPTR)    = ASCII STRING
3790 *                  SOURCECNT      = STRING LENGTH (BUT NOT > 6)
3800 *                  STRING TO CONSIST OF 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
3810 *                  BUT NO OTHER CHARACTERS
3820 *
3830 * MAKEHEX RETURNS: HL             = VALUE OF THE STRING
3840 *                  B              = STRING TYPE = DATA16 IF EVALUATED
3850 *                  A              = RESULT INDEX
3860 *                                 = 0 = EVALUATED HEX
3870 *                                   1 = NOT HEX IDENTIFIER AT LAST CHAR
3880 *                                   2 = INVALID CHARS IN ASCII STRING
3890 *
3900 *

3910 MAKEHEX        LDA     SOURCECNT   ;GET STRING LENGTH
3920                MOV     C,A         ;SAVE STRING LENGTH
3930                CPI     7           ;IF STRING >= 7,
3940                JNC     NOTHEX      ;THEN NOT A VALID HEX NUMBER (2)
3950                                    ;
3960                LHLD    SOURCEPTR   ;POINT TO STRING
3970                MVI     A,30H
3980                CMP     M           ;IF FIRST CHAR OF STRING NOT = "0",
3990                JNZ     NOTHEX      ;THEN NOT VALID HEX NUMBER (2)
4000                                    ;
4010                MVI     B,0         ;ELSE CHECK LAST CHARACTER
4020                DCR     C
4030                DAD     B           ;POINT TO LAST CHARACTER
4040                MVI     A,48H
4050                CMP     M           ;IF LAST CHAR NOT = "H",
4060                JNZ     NOTHEXID    ;THEN NOT HEX IDENTIFIER (1)

4070                                    ;
4080                LHLD    SOURCEPTR   ;ELSE TRY TO EVALUATE HEX NUMBER
4090                INX     H           ;POINT TO FIRST CHARACTER
4100                XCHG
4110                LXI     H,0         ;VALUE PRE-SET TO ZERO

4640 * MAKEDEC RETURNS: HL              = VALUE OF THE STRING
4650 *                  B               = STRING TYPE = DATA16 IF EVALUATED
4660 *                  A               = RESULT INDEX
4670 *                                  = 0 = EVALUATED DEC
4680 *                                    1 = NOT DEC IDENTIFIER AT LAST CHAR
4690 *                                    2 = INVALID CHARS IN ASCII STRING
4700 *                                    3 = STRING > 65536 (0FFFFH)
4710 *
4720 *

4730 MAKEDEC        LDA     SOURCECNT   ;GET STRING LENGTH
4740                MOV     C,A         ;SAVE STRING LENGTH
4750                CPI     8           ;IF STRING >= 8,
4760                JNC     NOTDEC      ;THEN NOT A VALID DEC NUMBER (2)

4770                                    ;
4780                LHLD    SOURCEPTR   ;POINT TO STRING
4790                MVI     A,30H
4800                CMP     M           ;IF FIRST CHAR OF STRING NOT = 0,
4810                JNZ     NOTDEC      ;THEN NOT VALID DEC NUMBER (2)
4820                                    ;
4830                MVI     B,0         ;ELSE CHECK LAST CHARACTER
4840                DCR     C
4850                DAD     B           ;POINT TO LAST CHARACTER
4860                MVI     A,44H
4870                CMP     M           ;IF LAST CHAR NOT = "D",
4880                JNZ     NOTDECID    ;THEN NOT DEC IDENTIFIER (1)

4890                                    ;
4900                LHLD    SOURCEPTR   ;ELSE TRY TO EVALUATE DEC NUMBER
4910                INX     H           ;POINT TO FIRST CHARACTER
4920                XCHG

4930                LXI     H,0         ;VALUE PRE-SET TO ZERO
4940 MAKEDEC1       DCR     C           ;ADJUST LENGTH FOR "0","D" IDENTIFIERS
4950                JZ      EXITMAKEDEC ;IF LENGTH = 0, THEN DONE

4960                                    ;
4970                LDAX    D           ;GET CHARACTER FROM STRING
4980                CPI     30H         ;IF CHAR < "0",
4990                JC      NOTDEC      ;INVALID DEC NUMBER (2)
5000                CPI     3AH         ;IF CHAR >= ":",
5010                JNC     NOTDEC      ;THEN NOT VALID DEC NUMBER (2)
5020                ANI     0FH         ;CONVERT ASCII DIGIT TO BINARY
5030                                    ;

5040 MAKEDECDIGIT   PUSH    B           ;SAVE LENGTH COUNT
5050                PUSH    H           ;SAVE VALUE
5060                LXI     H,DECVALUES ;POINT TO DECIMAL EQUIVALENTS
5070                DAD     B           ;POINT TO EQUIVALENT
5080                DAD     B
5090                MOV     C,M         ;EQUIVALENT INTO BC

5100                INX     H
5110                MOV     B,M
```

```
4120 MAKEHEX1       DCR    C              ;ADJUST LENGTH FOR "0","H" IDENTIFIERS
4130                JZ     HEX            ;IF LENGTH = 0, THEN DONE
4140                                      ;
4150                LDAX   D              ;GET CHARACTER FROM STRING
4160                CPI    30H            ;IF CHAR < "0",
4170                JC     NOTHEX         ;INVALID HEX NUMBER (2)
4180                CPI    3AH            ;IF CHAR < ":",
4190                JC     MAKEHEXDIGIT   ;THEN VALID HEX DIGIT (0-9)
4200                CPI    41H            ;IF CHAR < "A",
4210                JC     NOTHEX         ;THEN INVALID HEX NUMBER (2)
4220                CPI    47H            ;ELSE IF CHAR >= "G",
4230                JNC    NOTHEX         ;THEN INVALID HEX NUMBER (2)
4240                SUI    7              ;CONVERT ASCII A (41H) TO HEX A (3AH)
4250                                      ;
4260 MAKEHEXDIGIT   ANI    0FH            ;ELSE EXTRACT LO 4 BITS (DIGIT)
4270                PUSH   B              ;SAVE LENGTH COUNTER
4280                MVI    B,0
4290                MOV    C,A            ;PUT DIGIT IN BC
4300                DAD    H              ;SHIFT HL LEFT 4 BITS TO
4310                DAD    H              ;MAKE ROOM FOR NEW DIGIT
4320                DAD    H
4330                DAD    H
4340                DAD    B              ;ADD DIGIT TO LEAST SIG NIBBLE
4350                POP    B              ;RESTORE LENGTH
4360                INX    D              ;POINT TO NEXT CHARACTER
4370                JMP    MAKEHEX1       ;CONTINUE UNTIL LENGTH = 0
4380                                      ;
4390 NOTHEX         MVI    A,2            ;NOT VALID HEX NUMBER DUE TO
4400                JMP    EXITMAKEHEX    ;INVALID CHARACTER
4410                                      ;
4420 NOTHEXID       MVI    A,1            ;NOT VALID HEX NUMBER DUE TO
4430                JMP    EXITMAKEHEX
4440                                      ;
4450 HEX            MVI    B,9            ;HEX NUMBER - TYPE = DATA16
4460                XRA    A              ;VALID HEX NUMBER
4470                                      ;
4480 EXITMAKEHEX    RET
4490                                      ;


4500 ;*************************************************************************
4510 ;*
4520 ;* MAKEDEC CONVERTS AN ASCII REPRESENTATION OF A DECIMAL NUMBER
4530 ;* INTO ITS BINARY EQUIVALENT.  THE NUMBER MUST BE OF THE FORM
4540 ;* 0XXXXXH, WHERE XXXXX IS THE ASCII FORM OF THE NUMBER, AND MAY
4550 ;* BE ZERO TO FIVE DIGITS LONG.
4560 ;* THE NUMBER MUST BE PRECEEDED BY A "0", AND TERMINATE WITH THE
4570 ;* DECIMAL IDENTIFIER, "D".
4580 ;*
4590 ;* MAKEDEC EXPECTS: (SOURCEPTR)   = ASCII STRING
4600 ;*                  SOURCECNT     = STRING LENGTH (BUT NOT > 7)
4610 ;*                  STRING TO CONSIST OF 0,1,2,3,4,5,6,7,8,9
4620 ;*                  BUT NO OTHER CHARACTERS
4630 ;*
```

*(Continued on top of page 34)*

```
5120                POP    H              ;RESTORE VALUE FOR SUMMATION LOOP
5130 MAKEDEC2       ORA    A              ;IF DIGIT DECREMENTED TO ZERO,
5140                JZ     MAKEDEC3       ;THEN EQUIVALENT ADDED (DIGIT) TIMES
5150                DAD    B              ;ELSE ADD EQUIVALENT TO RESULT
5160                JC     TOOBIGDEC      ;INVALID DECIMAL NUMBER - TOO BIG (3)

5170                DCR    A
5180                JMP    MAKEDEC2
5190 MAKEDEC3       POP    B              ;RESTORE LENGTH
5200                INX    D              ;POINT TO NEXT CHARACTER
5210                JMP    MAKEDEC1       ;CONTINUE UNTIL LENGTH = 0
5220                                      ;

5230 NOTDEC         MVI    A,2            ;NOT VALID DEC NUMBER DUE TO
5240                JMP    EXITMAKEDE2    ;INVALID CHARACTER
5250                                      ;
5260 NOTDECID       MVI    A,1            ;NOT VALID DEC NUMBER DUE TO
5270                JMP    EXITMAKEDE2    ;INVALID IDENTIFIER

5280                                      ;
5290 TOOBIGDEC      MVI    A,3            ;DECIMAL NUMBER GREATER THAN 65535
5300                POP    B              ;RESTORE LENGTH
5310                JMP    EXITMAKEDE2
5320                                      ;
5330 EXITMAKEDEC    MVI    B,9            ;DEC NUMBER - TYPE = DATA16
5340                XRA    A              ;VALID DEC NUMBER
5350                                      ;
5360 EXITMAKEDE2    RET
5370                                      ;

5380 DECVALUES      DW     0              ;DUMMY VALUE
5390                DW     1              ;DEC 1     = HEX 1
5400                DW     0AH            ;DEC 10    = HEX A
5410                DW     64H            ;DEC 100   = HEX 64
5420                DW     3E8H           ;DEC 1000  = HEX 3E8
5430                DW     2710H          ;DEC 10000 = HEX 2710
5440                                      ;
```

**end**

# Tiny File System

```
 ASM
00C6-            1000 FFF .EQ $C6
00D7-            1010 WWW .EQ $D7
00D2-            1020 RRR .EQ $D2
000C-            1030 SN   .EQ 12
01C6-            1040 MAXBCK .EQ 454
000D-            1050 SWT  .EQ 13
0052-            1060 XTND .EQ $52
0050-            1070 ACCL .EQ $50
0054-            1080 AUXL .EQ $54
FB84-            1090 IDIV ,EQ $FB84
00FF-            1100 EOF  .EQ $FF
00FC-            1110 MXCH .EQ 252
000E-            1120 BASE .EQ 14
000B-            1130 ER   .EQ 11
03D9-            1140 RWTS .EQ $3D9
                 1150      .OR $90BF
                 1160      .TA $30BF
90BF- 4C 88 91   1170 TOBD JMP BDTS
90C2- 4C 7E 91   1180 TOBF JMP BFRF
90C5- 4C D7 90   1190 TOOP JMP OPEN
90C8- 4C CF 91   1200 TOCL JMP CLOS
90CB- 4C FD 91   1210 TORD JMP RDBF
90CE- 4C 6C 92   1220 TOGT JMP GTCH
90D1- 4C 06 93   1230 TOWR JMP WRBF
90D4- 4C 5D 93   1240 TOPT JMP PTCH
90D7- 20 55 91   1250 OPEN JSR SLOP
90DA- 8D C5 93   1260      STA OPSV
90DD- A9 00      1270      LDA #0
90DF- 85 0B      1280      STA ER
90E1- A8         1290      TAY   HAS ZERO
90E2- B1 54      1300      LDA (AUXL),Y   1:OPEN 0:CLOS
90E4- C9 01      1310      CMP #1
90E6- D0 05      1320      BNE NTOP
90E8- A9 02      1330      LDA #2  ERROR, FILE WAS OPEN
90EA- 85 0B      1340      STA ER
90EC- 60         1350      RTS
90ED- A9 01      1360 NTOP LDA #1
90EF- 91 54      1370      STA (AUXL),Y  IBTYPE(0)
90F1- C8         1380      INY
90F2- AD BF 91   1390      LDA SLOT
90F5- 91 54      1400      STA (AUXL),Y  IBSLOT(1)
90F7- C8         1410      INY
90F8- AD C6 91   1420      LDA DRVE
90FB- 91 54      1430      STA (AUXL),Y IBDRVN(2)
90FD- C8         1440      INY
914F- 91 0E      1900      STA (BASE),Y
9151- 20 6D 91   1910      JSR FILL  BYTEPRES AND CHSP
9154- 60         1920      RTS
9155- C9 D2      1930 SLOP CMP #RRR
9157- D0 03      1940      BNE NOTR  THIS SUB SELECTS
9159- A9 01      1950      LDA #1    THE OPERATOR CODE
915B- 60         1960      RTS       HERE IT IS TO READ
915C- C9 D7      1970 NOTR CMP #WWW
915E- D0 03      1980      BNE NOTW
9160- A9 02      1990      LDA #2   CODE FOR WRITE
9162- 60         2000      RTS
9163- C9 C6      2010 NOTW CMP #FFF
9165- D0 03      2020      BNE NOTF
9167- A9 04      2030      LDA #4   CODE FOR FORMAT
9169- 60         2040      RTS
916A- A9 00      2050 NOTF LDA #0
916C- 60         2060      RTS    DEFAULT IS NO OP
916D- A0 FC      2070 FILL LDY #MXCH  BYTEPRES=0
916F- A9 00      2080      LDA #0
9171- 91 0E      2090      STA (BASE),Y
9173- A9 FF      2100      LDA #$FF   EOF
9175- 88         2110      DEY
9176- 91 0E      2120 FLLP STA (BASE),Y  SET BFFR EOF
9178- 88         2130      DEY
9179- D0 FB      2140      BNE FLLP
917B- 91 0E      2150      STA (BASE),Y   AND ZERO
917D- 60         2160      RTS
917E- 85 0E      2170 BFRF STA BASE  IN:A=LOW BUFF ADDR
9180- 84 0F      2180      STY BASE+1   Y=HIGH BUFF ADD
9182- 85 54      2190      STA AUXL     OUT:BASE =BUFFR
9184- C8         2200      INY
9185- 84 55      2210      STY AUXL+1       AUX=IOB
9187- 60         2220      RTS
9188- 8C CA 91   2230 BDTS STY BCKH
918B- 8D C9 91   2240      STA BCKL  IN: LOW BUCK IN A
918E- 84 51      2250      STY ACCL+1
9190- 85 50      2260      STA ACCL
9192- A9 60      2270      LDA #$60  OBJECT--KEEP DRIVE
9194- 8D BF 91   2280      STA SLOT   SLOT,TRACK,SECTOR
9197- A9 01      2290      LDA #1     REFERENCES IN THS
9199- 8D C6 91   2300      STA DRVE   SUBROUTINE
919C- A9 0D      2310      LDA #13   13 SECTORS PER TRCK
919E- 85 54      2320      STA AUXL  SET FOR DIV
91A0- A9 00      2330      LDA #0
91A2- 85 55      2340      STA AUXL+1
91A4- 85 52      2350      STA XTND
```

```
90FE- A9 00     1450       LDA #0
9100- 91 54     1460       STA (AUXL),Y IBVOL(3)
9102- C8        1470       INY
9103- AD C7 91  1480       LDA TRCK
9106- 91 54     1490       STA (AUXL),Y IBTRK(4)

9108- C8        1500       INY
9109- AD C8 91  1510       LDA SECT
910C- 91 54     1520       STA (AUXL),Y IBSECT(5)
910E- C8        1530       INY
910F- A9 C6     1540       LDA #DCTB
9111- 91 54     1550       STA (AUXL),Y IBDCTP(6)
9113- C8        1560       INY
9114- A9 93     1570       LDA /DCTB
9116- 91 54     1580       STA (AUXL),Y IBDCTP(7)
9118- C8        1590       INY
9119- A5 0E     1600       LDA BASE
911B- 91 54     1610       STA (AUXL),Y IBBUFP(8)
911D- C8        1620       INY
911E- A5 0F     1630       LDA BASE+1
9120- 91 54     1640       STA (AUXL),Y IBBUFP(9)
9122- C8        1650       INY
9123- C8        1660       INY  TWO BYTES UNUSED
9124- A9 00     1670       LDA #0  USE ONE FOR
9126- 91 54     1680       STA (AUXL),Y   NEXTCHAR=0
9128- C8        1690       INY    (11)
9129- AD C5 93  1700       LDA OPSV
912C- 91 54     1710       STA (AUXL),Y   IBCMD(12)
912E- C8        1720       INY    IBSTAT (13) OUTPUT
912F- C8        1730       INY    IBSMOD (14) OUTPUT
9130- C8        1740       INY
9131- AD CD 91  1750       LDA RSLT
9134- 91 54     1760       STA (AUXL),Y  MST RCT SLT(15
9136- C8        1770       INY
9137- AD CE 91  1780       LDA RDRV
913A- 91 54     1790       STA (AUXL),Y MST RCT DRV(16
913C- C8        1800       INY
913D- AD C9 91  1810       LDA BCKL
9140- 91 54     1820       STA (AUXL),Y CURRBUCK-L(17)
9142- C8        1830       INY
9143- AD CA 91  1840       LDA BCKH
9146- 91 54     1850       STA (AUXL),Y CURRBUCK-H(18)
9148- A0 FE     1860       LDY #$FE  NEXTBUCK INDEX
914A- A9 00     1870       LDA #$00
914C- 91 0E     1880       STA (BASE),Y
914E- C8        1890       INY
```

*(Continued on top of page 36)*

```
91A6- 85 53     2360       STA XTND+1
91A8- 20 84 FB  2370       JSR IDIV
91AB- A5 50     2380       LDA ACCL   DIVIDEND LOW BYTE
91AD- 8D C7 91  2390       STA TRCK
91B0- A5 52     2400       LDA XTND   REMAINDER LOW BYT
91B2- 8D C8 91  2410       STA SECT
91B5- A5 0E     2420       LDA BASE
91B7- 85 54     2430       STA AUXL
91B9- A4 0F     2440       LDY BASE+1
91BB- C8        2450       INY
91BC- 84 55     2460       STY AUXL+1
91BE- 60        2470       RTS
91BF- 00        2480 SLOT  .HS 00
91C0- AD CE 91  2490       LDA RDRV       DRIVE

91C3- C8        2500       INY
91C4- 91 0E     2510       STA (BASE),Y
91C6- 00        2520 DRVE  .HS 00
91C7- 00        2530 TRCK  .HS 00
91C8- 00        2540 SECT  .HS 00
91C9- 00        2550 BCKL  .HS 00
91CA- 00        2560 BCKH  .HS 00
91CB- 00        2570 NBKL  .HS 00
91CC- 00        2580 NBKH  .HS 00
91CD- 60        2590 RSLT  .HS 60
91CE- 01        2600 RDRV  .HS 01
91CF- 20 7E 91  2610 CLOS  JSR BFRF
91D2- A0 00     2620       LDY #0
91D4- B1 54     2630       LDA (AUXL),Y   CHECK OPEN
91D6- C9 01     2640       CMP #1
91D8- F0 05     2650       BEQ WSOP
91DA- A9 02     2660       LDA #2   WAS CLOSED
91DC- 85 0B     2670       STA ER
91DE- 60        2680       RTS
91DF- A0 0C     2690 WSOP  LDY #12
91E1- B1 54     2700       LDA (AUXL),Y  IBCMND
91E3- C9 02     2710       CMP #2  WRITE FLAG
91E5- D0 0C     2720       BNE SHUT
91E7- A9 00     2730       LDA #$00  SET NEXTBUCK=0
91E9- A0 FE     2740       LDY #$FE
91EB- 91 0E     2750       STA (BASE),Y
91ED- C8        2760       INY
91EE- 91 0E     2770       STA (BASE),Y
91F0- 20 06 93  2780       JSR WRBF
91F3- A0 00     2790 SHUT  LDY #0
91F5- 98        2800       TYA
```

*(Continued on next page)*

# Tiny File System (Continued)

```
91F6- E6 0F     2810       INC BASE+1   BECAUSE RWTS
91F8- 91 0E     2820       STA (BASE),Y  MAY USE AUX
91FA- A0 0C     2830       LDY #12 MARK CLOSED
91FC- 60        2840       RTS
91FD- A2 01     2850 RDBF  LDX #1 TO TEST READ OPEN
91FF- 20 A0 92  2860       JSR CHOP  CHECK OPERATOR
9202- 20 B5 92  2870       JSR CHEF  CHECK EOF
9205- 20 3E 92  2880 RDNX  JSR RWPR
9208- A4 54     2890       LDY AUXL
920A- A5 55     2900       LDA AUXL+1
920C- 20 D9 03  2910       JSR RWTS
920F- 90 05     2920       BCC RDST
9211- A9 03     2930       LDA #3
9213- 85 0B     2940       STA ER
9215- 60        2950       RTS
9216- A0 FE     2960 RDST  LDY #$FE  FOR NEXTBUCK
9218- B1 0E     2970       LDA (BASE),Y
921A- AA        2980       TAX
921B- C8        2990       INY
921C- B1 0E     3000       LDA (BASE),Y
921E- E6 0F     3010       INC BASE+1
9220- A0 12     3020       LDY #18  SET CURRBUCK=NEXTBK
9222- 91 0E     3030       STA (BASE),Y
9224- 88        3040       DEY
9225- 8A        3050       TXA
9226- 91 0E     3060       STA (BASE),Y
9228- A9 00     3070       LDA #0   NEXTCHAR=0
922A- A0 0B     3080       LDY #11
922C- 91 0E     3090       STA (BASE),Y
922E- AD CD 91  3100       LDA RSLT
9231- A0 0F     3110       LDY #15  AND SET MOST RCT
9233- 91 0E     3120       STA (BASE),Y  SLOT AND
9235- AD CE 91  3130       LDA RDRV   DRIVE
9238- C8        3140       INY
9239- 91 0E     3150       STA (BASE),Y
923B- C6 0F     3160       DEC BASE+1  FOR GTCH
923D- 60        3170       RTS
923E- A0 11     3180 RWPR  LDY #17  GET LOW CURRBUCK
9240- B1 54     3190       LDA (AUXL),Y
9242- AA        3200       TAX
9243- C8        3210       INY
9244- B1 54     3220       LDA (AUXL),Y  HIGH CURRBUCK
9246- A8        3230       TAY
9247- 8A        3240       TXA
9248- 20 88 91  3250       JSR BDTS  SET UP S,D,T,S
924B- A0 01     3260       LDY #1
924D- AD BF 91  3270       LDA SLOT
92AE- A9 02     3740       LDA #2
92B0- 85 0B     3750 CHER  STA ER
92B2- 68        3760       PLA
92B3- 68        3770       PLA  RETURN TO MAIN,NOT IMMD
92B4- 60        3780 CHRT  RTS
92B5- A0 11     3790 CHEF  LDY #17 CHECK END OF FILE
92B7- B1 54     3800       LDA (AUXL),Y  LOW CURRBUCK
92B9- C9 00     3810       CMP #$00
92BB- D0 F7     3820       BNE CHRT  CANT BE EOF
92BD- C8        3830       INY
92BE- B1 54     3840       LDA (AUXL),Y
92C0- C9 00     3850       CMP #$00  HIGH CURRBUCK
92C2- D0 F0     3860       BNE CHRT  LAST CHANCE
92C4- A9 01     3870       LDA #1   THIS WAS LAST SECTR
92C6- 85 0B     3880       STA ER
92C8- 68        3890       PLA    TWO LEVEL RETURN
92C9- 68        3900       PLA
92CA- A9 FF     3910       LDA #EOF
92CC- 60        3920       RTS   RETRN EOF
92CD- 85 52     3930 CHUS  STA XTND COMPAR UNSGD BYTS
92CF- 84 53     3940       STY XTND+1
92D1- 45 53     3950       EOR XTND+1 INPUT A AND Y
92D3- 30 0E     3960       BMI LDBT   OUTPUT 1 A>Y
92D5- A5 52     3970 LDEQ  LDA XTND          0 A=Y
92D7- C5 53     3980       CMP XTND+1       -1 A<Y
92D9- F0 04     3990 TEST  BEQ REQ EQUAL
92DB- 30 03     4000       BMI ALSY
92DD- A9 01     4010 AGRY  LDA #1  A GREATER
92DF- 60        4020 REQ   RTS
92E0- A9 FF     4030 ALSY  LDA #$FF ALESS THAN Y
92E2- 60        4040       RTS
92E3- A5 52     4050 LDBT  LDA XTND LEAD BITS NOT EQ
92E5- 30 F6     4060       BMI AGRY A IS ON SO GREATER
92E7- 10 F7     4070       BPL ALSY A IS OFF SO LESS
92E9- 8D 05 93  4080 CHEM  STA CHLS  CHECK END OF MDIUM
92EC- A9 01     4090       LDA /MAXBCK   IN A:LOW BUCK
92EE- 20 CD 92  4100       JSR CHUS         Y:HIGH BUCK
92F1- 10 05     4110       BPL CHLB  COMPARE WITH MAXBK
92F3- A9 01     4120 CHEX  LDA #1   A(HMXB)<Y(HBUCK)BAD
92F5- 4C B0 92  4130       JMP CHER
92F8- D0 BA     4140 CHLB  BNE CHRT A=Y SO OK
92FA- AC 05 93  4150       LDY CHLS
92FD- A9 C6     4160       LDA #MAXBCK A=LMXB Y=LBUCK
92FF- 20 CD 92  4170       JSR CHUS
9302- 30 EF     4180       BMI CHEX A<Y BAD
9304- 60        4190       RTS      OTHERWISE OK
9305- 00        4200 CHLS  .HS 00  LOW BYTE STORAGE
```

```
9250- 91 54     3280       STA (AUXL),Y
9252- 8D CD 91  3290       STA RSLT
9255- C8        3300       INY
9256- AD C6 91  3310       LDA DRVE
9259- 8D CE 91  3320       STA RDRV
925C- 91 54     3330       STA (AUXL),Y
925E- C8        3340       INY
925F- C8        3350       INY
9260- AD C7 91  3360       LDA TRCK
9263- 91 54     3370       STA (AUXL),Y
9265- C8        3380       INY
9266- AD C8 91  3390       LDA SECT
9269- 91 54     3400       STA (AUXL),Y
926B- 60        3410       RTS
926C- 20 7E 91  3420 GTCH  JSR BFRF  STORTE BUFF REF
926F- A2 01     3430       LDX #1  TO CHECK FOR READ
9271- 20 A0 92  3440       JSR CHOP
9274- A0 0B     3450       LDY #11  NEXTCHAR
9276- B1 54     3460       LDA (AUXL),Y
9278- A0 FC     3470       LDY #MXCH CHARPRES
927A- D1 0E     3480       CMP (BASE),Y
927C- D0 0D     3490       BNE GTOK
927E- 20 B5 92  3500       JSR CHEF  CHECK EOF
9281- 20 FD 91  3510       JSR RDBF  REFILL BUFFER
9284- A5 0B     3520       LDA ER
9286- F0 03     3530       BEQ GTOK
9288- 4C CA 93  3540       JMP MONRTS HARD ER ON READ
928B- E6 0F     3550 GTOK  INC BASE+1  MAY HAVE READ
928D- A0 0B     3560       LDY #11  NEXT CHAR
928F- B1 0E     3570       LDA (BASE),Y
9291- AA        3580       TAX
9292- 18        3590       CLC
9293- 69 01     3600       ADC #1
9295- 91 0E     3610       STA (BASE),Y NEXTCHAR++
9297- C6 0F     3620       DEC BASE+1
9299- 8A        3630       TXA
929A- A8        3640       TAY
929B- B1 0E     3650       LDA (BASE),Y
929D- 4C CA 93  3660       JMP MONRTS THE BYTE AT LAST
92A0- 86 0D     3670 CHOP  STX SWT   INPUT X:TEST COND
92A2- A9 00     3680       LDA #0  READ CHECKS
92A4- 85 0B     3690       STA ER
92A6- A0 0C     3700       LDY #12  IBCMD=1,2 FOR R,W
92A8- B1 54     3710       LDA (AUXL),Y
92AA- C5 0D     3720       CMP SWT
92AC- F0 06     3730       BEQ CHRT
```

*(Continued on top of page 38)*

```
9306- A2 02     4210 WRBF  LDX #2  TO CHECK WRITE OPEN
9308- 20 A0 92  4220       JSR CHOP
930B- A0 11     4230       LDY #17 LOW CURRBUCK
930D- B1 54     4240       LDA (AUXL),Y
930F- 8D C9 91  4250       STA BCKL  FOR LATER USE
9312- AA        4260       TAX       FOR IMMD USE
9313- C8        4270       INY
9314- B1 54     4280       LDA (AUXL),Y  HICH CURRBUCK
9316- 8D CA 91  4290       STA BCKH
9319- A8        4300       TAY   HIGHBUCK IN Y
931A- 8A        4310       TXA   LOW BUCK IN A
931B- 20 E9 92  4320       JSR CHEM  OK TO USE CRRBUCK?
931E- A0 FE     4330       LDY #$FE  LOW NEXTBUCK
9320- B1 0E     4340       LDA (BASE),Y
9322- 8D CB 91  4350       STA NBKL
9325- AA        4360       TAX
9326- C8        4370       INY  HIGH NEXTBUCK
9327- B1 0E     4380       LDA (BASE),Y
9329- 8D CC 91  4390       STA NBKH
932C- A8        4400       TAY    HIGHBUCK IN Y
932D- 8A        4410       TXA    LOW BUCK IN A
932E- 20 E9 92  4420       JSR CHEM OK TO ASSERT NXBK?
9331- 20 3E 92  4430       JSR RWPR
9334- A5 55     4440       LDA AUXL+1
9336- A4 54     4450       LDY AUXL   LOW BYT IOB ADDR
9338- 20 D9 03  4460       JSR RWTS
933B- 90 05     4470       BCC WRST
933D- A9 03     4480       LDA #3  WRITE HARDWARE ERR
933F- 85 0B     4490       STA ER
9341- 60        4500       RTS
9342- A0 11     4510 WRST  LDY #17  CURRBUCK
9344- E6 0F     4520       INC BASE+1
9346- AD CB 91  4530       LDA NBKL RECOVER NEXTBUCK
9349- 91 0E     4540       STA (BASE),Y  CURRBUCK=NXBK
934B- AD CC 91  4550       LDA NBKH
934E- C8        4560       INY
934F- 91 0E     4570       STA (BASE),Y
9351- A0 0B     4580       LDY #11 NEXT CHAR=0
9353- A9 00     4590       LDA #0
9355- 91 0E     4600       STA (BASE),Y
9357- C6 0F     4610       DEC BASE+1
9359- 20 6D 91  4620       JSR FILL  EOF FILL CHRSP
935C- 60        4630       RTS
935D- 8E C4 93  4640 PTCH  STX PTCC
9360- 20 7E 91  4650       JSR BFRF  PUT ONE CHAR IN BF
9363- A2 02     4660       LDX #2
```

*(Continued on next page)*

# Tiny File System (Continued)

```
9365- 20 A0 92 4670        JSR CHOP  CHECK OPEN WRITE
9368- A0 0B    4680        LDY #11  NEXTCHAR
936A- B1 54    4690        LDA (AUXL),Y
936C- C9 FC    4700        CMP #MXCH
936E- D0 33    4710        BNE PTOK
9370- A0 11    4720        LDY #17  NEXTB=CURRB+1
9372- B1 54    4730        LDA (AUXL),Y LCURR
9374- 18       4740        CLC
9375- 69 01    4750        ADC #1        +1
9377- A0 FE    4760        LDY #$FE
9379- 91 0E    4770        STA (BASE),Y  LNEXT
937B- A9 00    4780        LDA #0
937D- A0 12    4790        LDY #18
937F- 71 54    4800        ADC (AUXL),Y HCURR
9381- A0 FF    4810        LDY #$FF
9383- 91 0E    4820        STA (BASE),Y  HNEXT
9385- 20 06 93 4830        JSR WRBF EMPTY BUFFER
9388- A0 FE    4840        LDY #$FE  CURRB=NEXTB
938A- B1 0E    4850        LDA (BASE),Y
938C- AA       4860        TAX
938D- C8       4870        INY
938E- B1 0E    4880        LDA (BASE),Y
9390- A0 12    4890        LDY #18
9392- E6 0F    4900        INC BASE+1
9394- 91 0E    4910        STA (BASE),Y
9396- 8A       4920        TXA
9397- 88       4930        DEY
9398- 91 0E    4940        STA (BASE),Y
939A- C6 0F    4950        DEC BASE+1
939C- A5 0B    4960        LDA ER
939E- F0 03    4970        BEQ PTOK
93A0- 4C CA 93 4980        JMP MONRTS
93A3- E6 0F    4990 PTOK   INC BASE+1  MAY HAVE WRITTEN
93A5- A0 0B    5000        LDY #11  NEXTCHAR
93A7- B1 0E    5010        LDA (BASE),Y     ++
93A9- AA       5020        TAX
93AA- 18       5030        CLC
93AB- 69 01    5040        ADC #1
93AD- 91 0E    5050        STA (BASE),Y
93AF- C6 0F    5060        DEC BASE+1
93B1- A0 FC    5070        LDY #MXCH
93B3- 91 0E    5080        STA (BASE),Y  BYTEPR=NEXTCH
93B5- C8       5090        INY
93B6- A9 00    5100        LDA #0  SETTING HI BYTE CH PR
93B8- 91 0E    5110        STA (BASE),Y
93BA- 8A       5120        TXA
93BB- A8       5130        TAY     INDEX FOR NEXT CHAR
```



# H8 Text Editor

```
                ; TAPECOPY PROGRAM - READS A HEATH TED-8 CASSETTE FILE
                ; AND CONVERTS IT TO A CP/M ED DISK FILE.

                ; THIS PROGRAM WAS WRITTEN BY JIM SCOTT, 1724 BLAKEMORE ROAD,
                ; RICHMOND, VA 23225, (804) 232-2925.  THIS PROGRAM MAY BE
                ; USED OR COPIED FOR NON-COMMERCIAL PURPOSES ONLY.  ALL COPIES
                ; MUST INCLUDE THIS STATEMENT.  THE AUTHOR ACCEPTS NO
                ; LIABILITY FOR ANY DAMAGE RESULTING FROM THE USE OR
                ; MALFUNCTION OF THIS PROGRAM, AND NO WARRANTY EXPRESS OR
                ; IMPLIED APPLIES TO THIS PROGRAM.

                ; INSTRUCTIONS:
                ;
                ; TO USE THIS PROGRAM, TYPE "TAPECOPY FILENAME", WHERE
                ; "FILENAME" IS THE COMPLETE NAME OF THE CP/M FILE TO BE
                ; CREATED, INCLUDING THE DRIVE NAME LETTER IF NECESSARY.
                ; FOR EXAMPLE, "TAPECOPY B:TESTPGM.ASM".
                ;
                ; TO THE MESSAGE "WATCH DATA ON SCREEN?", REPLY "Y" OR "N"
                ; TO SPECIFY WHETHER YOU WANT ALL OF THE DATA DISPLAYED ON
                ; THE SCREEN AS IT IS TRANSFERRED; SUGGESTION: REPLY "Y"
                ; FOR SHORT FILES (TO VERIFY THAT THE TRANSFER IS OPERATING
                ; CORRECTLY); REPLY "N" FOR LONG FILES TO AVOID A LONG RUN.
                ;
                ; WHEN ASKED TO "READY THE CASSETTE RECORDER", POSITION THE
                ; RECORDER JUST BEFORE THE BEGINNING OF THE FILE AND PRESS
                ; THE "PLAY" BUTTON; THEN TYPE CR.  (IF YOU REPLIED "Y" TO
                ; "WATCH DATA?", THE DISPLAY WILL BEGIN WHEN THE RECORDER
                ; REACHES THE END OF THE FIRST TAPE RECORD.)
                ;
                ; THE MESSAGE "END OF COPY" INDICATES SUCCESSFUL COMPLETION.

                ; THIS PROGRAM USES ROUTINES CONTAINED IN THE H8 PANEL
                ; MONITOR ROM.  THE FRONT PANEL WILL NOT DISPLAY ANYTHING IN
                ; PARTICULAR, AND MAY BE BLANK.  IN THE EVENT OF A TAPE
                ; CHECKSUM ERROR, THE SYSTEM WILL SIMPLY STOP WITH NO MESSAGE,
                ; AND THE RECORDER WILL STOP.  IF THIS HAPPENS, REBOOT AND
                ; TRY AGAIN.


4200 =          BASE    EQU     4200H   ;HEATH VERSION
4300 =          TPA     EQU     BASE+100H  ;TRANSIENT PROGRAM AREA.
4200 =          BOOT    EQU     BASE+0H ;RE-BOOT ENTRY POINT
4205 =          BDOS    EQU     BASE+5H ;DOS ENTRY POINT
4300            ORG     TPA

                ; CP/M BDOS FUNCTIONS

0001 =          READC   EQU     1       ;READ CONSOLE INTO (A).
0002 =          TYPEC   EQU     2       ;WRITE TO CONSOLE FROM (E).
000D =          INIT    EQU     13      ;INITIALIZE DISK IN DRIVE A.
0010 =          CLOSE   EQU     16      ;CLOSE FILE.
0011 =          FIND    EQU     17      ;FIND FILE IN DIRECTORY.
0013 =          DELE    EQU     19      ;DELETE FILE.
0015 =          WRITE   EQU     21      ;WRITE RECORD.
0016 =          MAKE    EQU     22      ;CREATE FILE DIRECTORY ENTRY.

                ; HEATH PAM-8 ADDRESSES

027A =          CTC     EQU     027AH   ;VERIFY TAPE CHECKSUM.
02B5 =          SRS     EQU     02B5H   ;SCAN RECORD START.
02D9 =          RNB     EQU     02D9H   ;READ NEXT TAPE BYTE.
00F9 =          OPTPC   EQU     0F9H    ;OUTPUT TAPE CONTROL PORT.

                ; FILE CONTROL BLOCK AND BUFFER ADDRESSES
```

```
93BC- AD C4 93 5140       LDA PTCC    SAVED CHAR TOPUT
93BF- 91 0E    5150       STA (BASE),Y
93C1- 4C CA 93 5160       JMP MONRTS
93C4- 00       5170 PTCC  .HS 00
93C5- 00       5180 OPSV  .HS 00
93C6- 00 01 EF
93C9- D8       5190 DCTB  .HS 0001EFD8
93CA- 85 45    5200 MONRTS STA $45
93CC- 86 46    5210       STX $46
93CE- 84 47    5220       STY $47
93D0- 08       5230       PHP
93D1- 68       5240       PLA
93D2- 85 48    5250       STA $48
93D4- 60       5260       RTS

:5495
:ASM


     SYMBOL TABLE    9176- FLLP      923E- RWPR
                     926C- GTCH      03D9- RWTS
     0050- ACCL      928B- GTOK      91C8- SECT
     92DD- AGRY      FB84- IDIV      91F3- SHUT
     92E0- ALSY      92E3- LDBT      9155- SLOP
     0054- AUXL      92D5- LDEQ      91BF- SLOT
     000E- BASE      01C6- MAXBCK    000C- SN
     91CA- BCKH      93CA- MONRTS    000D- SWT
     91C9- BCKL      00FC- MXCH      92D9- TEST
     9188- BDTS      91CC- NBKH      90BF- TOBD
     917E- BFRF      91CB- NBKL      90C2- TOBF
     92B5- CHEF      916A- NOTF      90C8- TOCL
     92E9- CHEM      915C- NOTR      90CE- TOGT
     92B0- CHER      9163- NOTW      90C5- TOOP
     92F3- CHEX      90ED- NTOP      90D4- TOPT
     92F8- CHLB      90D7- OPEN      90CB- TORD
     9305- CHLS      93C5- OPSV      90D1- TOWR
     92A0- CHOP      93C4- PTCC      91C7- TRCK
     92B4- CHRT      935D- PTCH      9306- WRBF
     92CD- CHUS      93A3- PTOK      9342- WRST
     91CF- CLOS      91FD- RDBF      91DF- WSOP
     93C6- DCTB      9205- RDNX      00D7- WWW
     91C6- DRVE      91CE- RDRV      0052- XTND
     00FF- EOF       9216- RDST
     000B- ER        92DF- REQ       :4995
     00C6- FFF       00D2- RRR       :6000    .LIST ON
     916D- FILL      91CD- RSLT      :ASM
```

**end**

```
425C =          FCB     EQU     BASE+5CH        ;FILE CONTROL BLOCK ADDRESS
425D =          FCBFN   EQU     FCB+1   ;FILE NAME
4265 =          FCBFT   EQU     FCB+9   ;DISK FILE TYPE (3 CHARACTERS)

426B =          FCBRC   EQU     FCB+15  ;FILE'S RECORD COUNT (0 TO 128)
427C =          FCBCR   EQU     FCB+32  ;CURRENT (NEXT) RECORD NUMBER (0-127)
4280 =          BUFF    EQU     BASE+80H        ;DISK BUFFER ADDRESS

                ; OTHER HANDY EQUATES.

0008 =          BS      EQU     08H     ;BACKSPACE.
000D =          CR      EQU     0DH     ;CARRIAGE RETURN.
000A =          LF      EQU     0AH     ;LINE FEED.
001A =          CTRLZ   EQU     1AH     ;CONTROL-Z (RECORD FILLER).

                ; MAIN LOOP:

4300 =          START   EQU     $
4300 CDE046             CALL    MSGOUT
4303 5441504520D        DB      'TAPE-TO-DISK UTILITY',CR,LF
4319 5741544348         DB      'WATCH DATA ON SCREEN? (Y OR N)',0
4338 =          SCNRPLY EQU     $
4338 CDFD46             CALL    REPLY   ;GET REPLY FROM CONSOLE.
433B FE59               CPI     'Y'     ;YES?
433D CA4F43             JZ      YSCREEN
4340 FE4E               CPI     'N'     ;NO?
4342 CA5443             JZ      NSCREEN
4345 CDE046             CALL    MSGOUT  ;BACKSPACE AND WAIT FOR CORRECT REPLY.
4348 08200800           DB      BS,' ',BS,0
434C C33843             JMP     SCNRPLY


434F =          YSCREEN EQU     $
434F 3E01               MVI     A,1     ;SET SCREEN REVIEW INDICATOR.
4351 C35543             JMP     SETSCRN

4354 =          NSCREEN EQU     $
4354 97                 SUB     A       ;INDICATE NO SCREEN REVIEW.


4355 =          SETSCRN EQU     $
4355 321747             STA     SCRNSW  ;STORE SCREEN INDICATOR.
4358 CDE046             CALL    MSGOUT
435B 0D0A524541         DB      CR,LF,'READY THE CASSETTE RECORDER AND TYPE '
4382 4352205748         DB      'CR WHEN READY.',0
4391 CDFD46             CALL    REPLY   ;WAIT FOR ANY REPLY.
4394 CDE046             CALL    MSGOUT  ;CR-LF TO COMPLETE PREVIOUS MSG.
4397 0D0A00             DB      CR,LF,0


439A 3E01               MVI     A,1     ;INITIALIZE FIRST-PASS INDICATOR.
439C 321847             STA     FIRSTSW

439F =          LOOP    EQU     $
439F CDC043             CALL    GETCHAR

43A2 CDFD44             CALL    PUTCHAR
43A5 3A1D47             LDA     ENDSW
43A8 B7                 ORA     A
43A9 CA9F43             JZ      LOOP
43AC CDE046             CALL    MSGOUT
43AF 454E44204F         DB      'END OF COPY',CR,LF,0
43BD C30042             JMP     BOOT
```

*(Continued on next page)*

# H8 Text Editor (Continued)

```
                ; GETCHAR:
                ; RETURNS NEXT CHARACTER FROM TAPE, IN (A).
                ; ENDSW = 0 UNLESS END OF FILE.

43C0 =          GETCHAR EQU     $
43C0 3A1847             LDA     FIRSTSW ;FIRST TIME INTO ROUTINE?
43C3 B7                 ORA     A
43C4 C4FB43             CNZ     OPENTPE ;IF SO, OPEN TAPE.


43C7 3A1947             LDA     SPACSW  ;IN THE MIDST OF A BLANK STRING?
43CA B7                 ORA     A
43CB C2BA44             JNZ     CNTSPAC ;IF SO, GO FINISH STRING.

43CE 3A1A47             LDA     CRLFSW  ;IN THE MIDST OF A CR-LF STRING?
43D1 B7                 ORA     A
43D2 C2D444             JNZ     CNTCRLF ;IF SO, GO FINISH STRING.

43D5 3A1B47             LDA     INBUFSW ;UNFINISHED RECORD IN BUFFER?
43D8 B7                 ORA     A
43D9 C2ED43             JNZ     CALLCHR ;IF SO, GO GET NEXT CHARACTER.

43DC 3A1C47             LDA     EOFSW   ;END OF INPUT?
43DF B7                 ORA     A
43E0 C2DB44             JNZ     EOF     ;IF SO, GO SET END-OF-FILE INDICATOR.

43E3 CD2244             CALL    RECIN   ;GET NEXT RECORD INTO BUFFER.

43E6 3A1B47             LDA     INBUFSW ;UNFINISHED RECORD IN BUFFER?
43E9 B7                 ORA     A
43EA CADB44             JZ      EOF     ;IF NOT, GO SET END-OF-FILE INDICATOR.

43ED =          CALLCHR EQU     $
43ED CD9144             CALL    CHARIN  ;GET NEXT CHARACTER FROM INPUT BUFFER.

43F0 FE81               CPI     81H     ;DOES BYTE REPRESENT BLANKS?
43F2 D2B044             JNC     EXPSPAC ;YES, GO EXPAND.

43F5 FE00               CPI     0       ;END OF INPUT LINE?
43F7 CACC44             JZ      EXPCRLF ;YES, GO EXPAND INTO CR-LF.

43FA C9                 RET             ;RETURN WITH CHAR IN (A).



                ; START TAPE INPUT AND GET HEADER RECORD.


4408 CDB502             CALL    SRS     ;GET HEADER REC: (D,E)=EOF/TYPE, SEQ,
                                                         (H,A)=DATA COUNT.
440B 7A                 MOV     A,D     ;END OF FILE?
440C E680               ANI     80H
440E 321C47             STA     EOFSW   ;IF SO, SET EOF SWITCH.

4411 7A                 MOV     A,D     ;HEATH TED-8 RECORD (TYPE = 3)?
4412 E67F               ANI     7FH
4414 FE03               CPI     3
4416 C2FB43             JNZ     OPENTPE ;NO, TRY AGAIN.

4419 7B                 MOV     A,E     ;HEADER RECORD (SEQ = 0)?
441A B7                 ORA     A
441B C2FB43             JNZ     OPENTPE ;NO, TRY AGAIN.

441E 322147             STA     RECSEQ  ;STORE SEQ #.
4421 C9                 RET             ;RETURN.


4472 2A2847             LHLD    DATAPTR ;ACCUMULATE EACH BYTE INTO BUFFER.
4475 77                 MOV     M,A
4476 23                 INX     H
4477 222847             SHLD    DATAPTR

447A C36344             JMP     MOVBYTE


                ; TURN OFF TAPE.

447D =          TAPEOFF EQU     $
447D CD7A02             CALL    CTC     ;VERIFY CHECKSUM.

4480 AF                 XRA     A       ;TURN OFF TAPE.
4481 D3F9               OUT     OPTPC

4483 2A2847             LHLD    DATAPTR ;SAVE PTR TO LAST BYTE IN BUFFER.
4486 2B                 DCX     H
4487 222A47             SHLD    DATAEND

448A 212C47             LXI     H,DATABUF       ;SAVE PTR TO BEGINNING OF
448D 222847             SHLD    DATAPTR         ;  FIRST DATA WITHIN BUFFER.
4490 C9                 RET             ;RETURN.



                ; MOVE NEXT TAPE CHARACTER INTO (A).
                ; IF THIS EXHAUSTS THE BUFFER, TURN OFF INBUFSW.


4491 =          CHARIN  EQU     $
4491 2A2847             LHLD    DATAPTR ;(H,L) = BUFFER POINTER.
4494 7E                 MOV     A,M     ;(A) = NEXT CHARACTER.
4495 F5                 PUSH    PSW     ;SAVE CHARACTER.
4496 23                 INX     H       ;ADD 1 TO BUFFER POINTER.
4497 222847             SHLD    DATAPTR
449A 2B                 DCX     H       ;BACK UP AGAIN FOR COMPARE.
449B E5                 PUSH    H
449C 2A2A47             LHLD    DATAEND ;WAS THIS THE LAST CHARACTER
449F 44                 MOV     B,H     ;  IN THE BUFFER?
44A0 4D                 MOV     C,L
44A1 E1                 POP     H
44A2 CD0947             CALL    BHCMP   ;COMPARE DATAEND TO OLD DATAPTR.
44A5 3E01               MVI     A,1     ;INBUFSW = 1 MEANS BUFF NOT EXHAUSTED.
44A7 C2AB44             JNZ     STOBSW
44AA 3D                 DCR     A       ;INBUFSW = 0 MEANS THIS IS LAST
44AB =          STOBSW  EQU     $       ;  CHARACTER IN BUFFER.
44AB 321B47             STA     INBUFSW
44AE F1                 POP     PSW     ;RETURN CHARACTER IN (A).

44AF C9                 RET

                ; EXPAND A BYTE OF VALUE 81H AND UP INTO (A)-80H SPACES.

44B0 =          EXPSPAC EQU     $
44B0 DE80               SBI     80H     ;SAVE NUMBER OF SPACES.
44B2 321F47             STA     NUMSPAC
44B5 3E01               MVI     A,1     ;SET INDICATOR OF BLANK STRING.
44B7 321947             STA     SPACSW

                ; CONTINUE EXPANDING STRING OF SPACES.

44BA =          CNTSPAC EQU     $
```

```
                ; GET NEW RECORD.

4422 =          RECIN   EQU     $
4422 3A1C47             LDA     EOFSW   ;END OF FILE?


43FB =          OPENTPE EQU     $
43FB 97                 SUB     A
43FC 321B47             STA     INBUFSW ;INITIALIZE INDICATOR OF REC IN BUFFER.
43FF 321947             STA     SPACSW  ;INITIALIZE BLANK STRING INDICATOR.
4402 321A47             STA     CRLFSW  ;INITIALIZE CR-LF SEQUENCE INDICATOR.
4405 321D47             STA     ENDSW   ;INITIALIZE END-OF-COPY INDICATOR.

4425 B7                 ORA     A
4426 C0                 RNZ             ;YES, RETURN.

4427 CDB502             CALL    SRS     ;GET DATA RECORD.
442A 6F                 MOV     L,A     ;STORE DATA COUNT.
442B 222647             SHLD    DATACNT

442E 7A                 MOV     A,D     ;END OF FILE.
442F E680               ANI     80H
4431 321C47             STA     EOFSW   ;IF SO, SET EOF SWITCH.

4434 7A                 MOV     A,D     ;TED-8 RECORD (TYPE = 3)?
4435 E67F               ANI     7FH
4437 FE03               CPI     3
4439 C2E144             JNZ     BADTAPE ;NO, ERROR.

443C 3A2147             LDA     RECSEQ  ;SEQ # INCREASED BY 1?
443F 3C                 INR     A
4440 BB                 CMP     E
4441 C2E144             JNZ     BADTAPE ;NO, ERROR.
4444 322147             STA     RECSEQ  ;YES, SAVE IT.

4447 2A2647             LHLD    DATACNT ;NULL RECORD?
444A 7C                 MOV     A,H
444B B5                 ORA     L
444C C25844             JNZ     INITBUF ;NO, GO AHEAD.
444F CD7A02             CALL    CTC     ;YES, VERIFY CHECKSUM,
4452 AF                 XRA     A       ;  TURN OFF TAPE,
4453 D3F9               OUT     OPTPC   ;  AND TRY NEXT RECORD.
4455 C32244             JMP     RECIN

4458 =          INITBUF EQU     $
4458 3E01               MVI     A,1     ;INDICATE RECORD IN BUFFER.
445A 321B47             STA     INBUFSW

445D 212C47             LXI     H,DATABUF       ;INITIALIZE BUFFER POINTER.

4460 222847             SHLD    DATAPTR

                ; ACCUMULATE DATA INTO BUFFER.

4463 =          MOVBYTE EQU     $
4463 2A2647             LHLD    DATACNT ;COUNT = 0?
4466 7C                 MOV     A,H
4467 B5                 ORA     L
4468 CA7D44             JZ      TAPEOFF ;YES, NO MORE DATA.

446B 2B                 DCX     H       ;DECREMENT DATA COUNT.
446C 222647             SHLD    DATACNT

446F CDD902             CALL    RNB     ;READ NEXT BYTE.
```

*(Continued on top of page 42)*

```
44BA 3A1F47             LDA     NUMSPAC ;IS THIS THE LAST SPACE
44BD D601               SUI     1       ;  IN THE STRING?
44BF 321F47             STA     NUMSPAC ;SAVE NEW COUNT.
44C2 C2C944             JNZ     OUTSPAC ;NO, LEAVE INDICATOR SET.
44C5 97                 SUB     A       ;YES, TURN OFF INDICATOR.
44C6 321947             STA     SPACSW

44C9 =          OUTSPAC EQU     $
44C9 3E20               MVI     A,' '   ;RETURN WITH (A) = SPACE.
44CB C9                 RET

                ; EXPAND A BYTE OF VALUE 0H INTO A CR-LF SEQUENCE.

44CC =          EXPCRLF EQU     $

44CC 3E01               MVI     A,1     ;SET INDICATOR OF CR-LF SEQUENCE.
44CE 321A47             STA     CRLFSW
44D1 3E0D               MVI     A,CR    ;RETURN WITH A CR, FIRST.
44D3 C9                 RET

                ; CONTINUE WITH SECOND HALF OF CR-LF SEQUENCE.

44D4 =          CNTCRLF EQU     $
44D4 97                 SUB     A
44D5 321A47             STA     CRLFSW
44D8 3E0A               MVI     A,LF    ;RETURN WITH A LF, SECOND.
44DA C9                 RET

                ; SET ENDSW TO INDICATE END-OF-COPY, BECAUSE PREVIOUS CALL
                ; TO GETCHAR TURNED ON EOFSW.

44DB =          EOF     EQU     $
44DB 3E01               MVI     A,1
44DD 321D47             STA     ENDSW
44E0 C9                 RET

                ; DISPLAY TAPE ERROR MESSAGE AND QUIT.

44E1 =          BADTAPE EQU     $
44E1 CDE046             CALL    MSGOUT
44E4 5441504520         DB      'TAPE SEQUENCE ERROR',CR,LF,0
44FA C30042             JMP     BOOT

                ; PUTCHAR:
                ; PUT CHARACTER IN (A) TO DISK.

44FD =          PUTCHAR EQU     $


44FD 321647             STA     CHAR    ;SAVE CHARACTER.
4500 3A1847             LDA     FIRSTSW ;FIRST TIME INTO ROUTINE?
4503 B7                 ORA     A
4504 C41545             CNZ     OPENDSK ;IF SO, OPEN DISK.

4507 3A1D47             LDA     ENDSW   ;END OF INPUT?
450A B7                 ORA     A
450B C20B46             JNZ     CLOSDSK ;IF SO, GO CLOSE DISK.

450E 3A1647             LDA     CHAR    ;GET CHARACTER BACK INTO (A).
4511 CDD445             CALL    CHAROUT ;PUT NEXT CHARACTER TO DISK BUFFER.

4514 C9                 RET             ;RETURN.

                ; OPEN DISK FILE.
```

*(Continued on next page)*

# H8 Text Editor (Continued)

```
4515 =          OPENDSK EQU     $
4515 97                 SUB     A
4516 321847             STA     FIRSTSW ;TURN OFF FIRST-PASS INDICATOR.

4519 115C42             LXI     D,FCB   ;SEE IF FILE EXISTS.
451C 0E11               MVI     C,FIND
451E CD0542             CALL    BDOS
4521 FEFF               CPI     255
4523 C44445             CNZ     DUPFILE ;YES, ERASE OR QUIT?

4526 115C42             LXI     D,FCB   ;OPEN FILE FOR WRITE.
4529 0E16               MVI     C,MAKE
452B CD0542             CALL    BDOS
452E FEFF               CPI     255     ;OPEN OK?
4530 CAA545             JZ      NOSPACE ;NO, OUT OF DIRECTORY SPACE.


4533 217C42             LXI     H,FCBCR ;INITIALIZE NEXT RECORD NUMBER.
4536 3600               MVI     M,0

4538 218042             LXI     H,BUFF  ;SAVE POINTER TO START OF
453B 222347             SHLD    WRECPTR ;  OUTPUT BUFFER.

453E 3E80               MVI     A,128   ;INITIALIZE COUNT OF BYTES
4540 322547             STA     WRECCNT ;  REMAINING IN OUTPUT BUFFER.

4543 C9                 RET             ;RETURN.

4544 =          DUPFILE EQU     $
4544 CDE046             CALL    MSGOUT
4547 46494C4520         DB      'FILE ALREADY EXISTS.  ENTER X TO ABORT.
456F 435220544F         DB      'CR TO ERASE IT.',0
457F =          DUPRPLY EQU     $
457F CDFD46             CALL    REPLY   ;GET REPLY FROM CONSOLE.
4582 FE58               CPI     'X'     ;ABORT?
4584 CA0042             JZ      BOOT    ;YES, QUIT.
4587 FE0D               CPI     CR      ;ERASE?
4589 CA9645             JZ      DUPERAS ;YES, GO DO IT.
458C CDE046             CALL    MSGOUT  ;NO, BACKSPACE & WAIT FOR REAL REPLY.
458F 0820080D           DB      BS,' ',BS,0
4593 C37F45             JMP     DUPRPLY

4596 =          DUPERAS EQU     $
4596 CDE046             CALL    MSGOUT  ;FINISH PREV MSG WITH CR-LF.


4599 0D0A00             DB      CR,LF,0

459C 115C42             LXI     D,FCB   ;ERASE FILE WITH SAME NAME.
459F 0E13               MVI     C,DELE
45A1 CD0542             CALL    BDOS
45A4 C9                 RET             ;RETURN.

45A5 =          NOSPACE EQU     $
45A5 CDE046             CALL    MSGOUT
45A8 4552524F52         DB      'ERROR - OUT OF DIRECTORY SPACE (OPEN).'
45CE 0D0A00             DB      CR,LF,0
45D1 C30042             JMP     BOOT

                ; PUT NEXT CHARACTER INTO DISK BUFFER.


45D4 =          CHAROUT EQU     $
45D4 CDF645             CALL    DISPCHR ;DISPLAY CHARACTER IF WANTED.

45D7 2A2347             LHLD    WRECPTR ;MOVE CHARACTER INTO BUFFER
```

```
462D 0E15               MVI     C,WRITE
462F CD0542             CALL    BDOS
4632 FE00               CPI     0       ;WRITE OK?
4634 C23846             JNZ     WERR    ;NO, DISK ERROR.
4637 C9                 RET             ;YES, RETURN.

4638 =          WERR    EQU     $
4638 FE01               CPI     1       ;RETURN CODE = 1?
463A C25D46             JNZ     WENOT1  ;NO.
463D CDE046             CALL    MSGOUT  ;YES, DISPLAY MSG.
4640 4552524F52         DB      'ERROR IN EXTENDING FILE',CR,LF,0
465A C30042             JMP     BOOT
465D =          WENOT1  EQU     $
465D FE02               CPI     2       ;RETURN CODE = 2?
465F C28446             JNZ     WENOT2  ;NO.
4662 CDE046             CALL    MSGOUT  ;YES, DISPLAY MSG.
4665 4552524F52         DB      'ERROR - OUT OF DISK SPACE',CR,LF,0
4681 C30042             JMP     BOOT
4684 =          WENOT2  EQU     $
4684 FEFF               CPI     255     ;RETURN CODE = 255?

4686 C28946             JNZ     WENOTFF ;NO.
4689 CDE046             CALL    MSGOUT  ;YES, DISPLAY MSG.
468C 4552524F52         DB      'ERROR - OUT OF DIRECTORY SPACE (WRITE).'
46B3 0D0A00             DB      CR,LF,0
46B6 C30042             JMP     BOOT
46B9 =          WENOTFF EQU     $
46B9 CDE046             CALL    MSGOUT
46BC 554E4B4E4F         DB      'UNKNOWN PERMANENT WRITE ERROR.',CR,LF,0
46DD C30042             JMP     BOOT

                ; DISPLAY MESSAGE ON CONSOLE.
                ; MESSAGE TEXT FOLLOWS CALL STATEMENT, AND ENDS WITH
                ; A BYTE OF 0.
                ; RETURN IS TO INSTRUCTION FOLLOWING 0 BYTE.
                ; REGISTERS H,L,A ARE DESTROYED.

46E0 =          MSGOUT  EQU     $

46E0 E1                 POP     H
46E1 =          MSGOUT1 EQU     $
46E1 7E                 MOV     A,M
46E2 FE00               CPI     0
46E4 CAEE46             JZ      MSGOUT2
46E7 CDF046             CALL    DISPLAY
46EA 23                 INX     H
46EB C3E146             JMP     MSGOUT1
46EE =          MSGOUT2 EQU     $
46EE 23                 INX     H
46EF E9                 PCHL


                ; DISPLAY ON CONSOLE THE CHARACTER IN (E).

46F0 =          DISPLAY EQU     $
46F0 E5                 PUSH    H
46F1 D5                 PUSH    D
46F2 C5                 PUSH    B
46F3 5F                 MOV     E,A
46F4 0E02               MVI     C,TYPEC
46F6 CD0542             CALL    BDOS
46F9 C1                 POP     B
46FA D1                 POP     D
46FB E1                 POP     H
46FC C9                 RET
```

```
45DA 77              MOV     M,A     ;  AND ADD 1 TO OUTPUT
45DB 23              INX     H       ;  BUFFER POINTER.
45DC 222347          SHLD    WRECPTR

45DF 3A2547          LDA     WRECCNT ;SUBTRACT 1 FROM BYTES REMAINING.
45E2 3D              DCR     A
45E3 322547          STA     WRECCNT
45E6 C0              RNZ             ;RETURN IF NOT END OF OUTPUT RECORD.

45E7 CD2A46          CALL    WRITER  ;WRITE FINISHED RECORD.

45EA 218042          LXI     H,BUFF  ;REINITIALIZE FOR NEXT RECORD.
45ED 222347          SHLD    WRECPTR
45F0 3E80            MVI     A,128
45F2 322547          STA     WRECCNT

45F5 C9              RET             ;RETURN.


45F6 =         DISPCHR EQU     $
45F6 F5              PUSH    PSW     ;SAVE CHARACTER.

45F7 3A1747          LDA     SCRNSW  ;SCREEN REVIEW WANTED?
45FA B7              ORA     A
45FB CA0946          JZ      DISPRTN ;NO, RETURN.
45FE 210746          LXI     H,MSGCHR        ;SET UP ONE-CHAR MSG.
4601 F1              POP     PSW
4602 F5              PUSH    PSW
4603 77              MOV     M,A
4604 CDE046          CALL    MSGOUT  ;SEND CHARACTER TO SCREEN.
4607 2000      MSGCHR  DB      ' ',0
4609 =         DISPRTN EQU     $
4609 F1              POP     PSW     ;RESTORE CHARACTER IN (A).
460A C9              RET

               ; CLOSE DISK.

460B =         CLOSDSK EQU     $
460B 3A2547          LDA     WRECCNT ;UNFINISHED OUTPUT RECORD?
460E FE80            CPI     128
4610 CA2146          JZ      CLOSOUT ;NO, GO CLOSE.


4613 2A2347          LHLD    WRECPTR ;YES, GET ADDRESS OF NEXT BYTE.
4616 47              MOV     B,A ;USE # BYTES REMAINING AS LOOP COUNT.

4617 =         CLOSLP  EQU     $
4617 361A            MVI     M,CTRLZ ;MOVE FILLER CHARACTER INTO RECORD.
4619 23              INX     H       ;BUMP TO NEXT RECORD BYTE.
461A 05              DCR     B       ;DECREMENT # BYTES REMAINING.
461B C21746          JNZ     CLOSLP  ;LOOP UNTIL RECORD IS FULL.

461E CD2A46          CALL    WRITER  ;WRITE OUTPUT RECORD.

4621 =         CLOSOUT EQU     $
4621 115C42          LXI     D,FCB   ;CLOSE DISK FILE.
4624 0E10            MVI     C,CLOSE
4626 CD0542          CALL    BDOS

4629 C9              RET             ;RETURN.


               ; WRITE AN OUTPUT RECORD.

462A =         WRITER  EQU     $
462A 115C42          LXI     D,FCB
```

*(Continued on top of page 44)*

```
               ; GET SINGLE-CHARACTER REPLY FROM CONSOLE INTO (A).

46FD =         REPLY   EQU     $
46FD E5              PUSH    H
46FE D5              PUSH    D
46FF C5              PUSH    B
4700 0E01            MVI     C,READC
4702 CD0542          CALL    BDOS
4705 C1              POP     B
4706 D1              POP     D
4707 E1              POP     H
4708 C9              RET

               ; COMPARE (B,C) TO (H,L).

4709 =         BHCMP   EQU     $
4709 78              MOV     A,B
470A BC              CMP     H
470B C0              RNZ

470C 79              MOV     A,C
470D 95              SUB     L
470E C8              RZ
470F 1F              RAR
4710 B7              ORA     A
4711 17              RAL
4712 F8              RM
4713 AF              XRA     A
4714 3C              INR     A
4715 C9              RET


4716           CHAR    DS      1       ;AREA FOR SAVING CHARACTER.
4717           SCRNSW  DS      1       ;1 MEANS SCREEN REVIEW WANTED.


4718           FIRSTSW DS      1       ;1 MEANS FIRST PASS THROUGH MAIN LOOP.
4719           SPACSW  DS      1       ;1 MEANS BLANK STRING BEING GENERATED.
471A           CRLFSW  DS      1       ;1 MEANS CR-LF SEQ BEING GENERATED.
471B           INBUFSW DS      1       ;1 MEANS MORE INPUT CHARS IN BUFFER.
471C           EOFSW   DS      1       ;80H MEANS LAST INPUT RECORD READ.
471D           ENDSW   DS      1       ;1 MEANS LAST PASS THROUGH MAIN LOOP.
471E           SAVEA   DS      1       ;WORK AREA.
471F           NUMSPAC DS      2       ;NUMBER OF SPACES TO BE GENERATED.
4721           RECSEQ  DS      2       ;SEQUENCE NUMBER OF INPUT RECORD.
4723           WRECPTR DS      2       ;POINTER TO NEXT BYTE IN OUTPUT BUFFER.
4725           WRECCNT DS      1       ;NUMBER OF BYTES LEFT IN OUTPUT BUFFER.
4726           DATACNT DS      2       ;# DATA BYTES TO BE MOVED INTO BUFFER.
4728           DATAPTR DS      2       ;POINTER TO NEXT CHARACTER IN BUFFER.
472A           DATAEND DS      2       ;ADDRESS OF LAST DATA BYTE IN BUFFER.
472C           DATABUF DS      128     ;INPUT BUFFER.
47AC
```

**end**

# COMAL 80

**Figure 1. Only the underlined words are keyed in by the user, whereas the rest shows how COMAL-80 by error-messages guides the user through to a correct program.**

```
*10 DIM ARRAY(17
ERROR 14: ")" expected
10 DIM ARRAY(17)
*20 OPEN
ERROR 30: "FILE" expected
20 OPEN FILE
ERROR 23: Operand expected
20 OPEN FILE 1
ERROR 21: "," expected
20 OPEN FILE 1,
ERROR 23: Operand expected
20 OPEN FILE 1,"TEST"
ERROR 21: "," expected
20 OPEN FILE 1,"TEST",
ERROR 43: READ/WRITE/RANDOM expected
20 OPEN FILE 1,"TEST",READ
*
```

**end**

**Figure 2. More than words can tell . . . this program describes the strength of COMAL-80. Sorry the names are all in Danish.**

```
0010 // POLYNY 27/11 - 1980
0020 INPUT "EPSILON: ": EPS
0030 DIM PROG$ OF 10
0040 DIM IND(0:50), POLY(-1:50)
0050 GR#:=0; ROD:=0; GROP#:=0
0060 MAT IND:=0
0070 MAT POLY:=0
0080 REPEAT
0090  INPUT "PROGRAM: ": PROG$
0100  CASE PROG$ OF
0110  WHEN "IND"
0120   EXEC IND
0130  WHEN "UD"
0140   EXEC UD
0150  WHEN "RETAB"
0160   EXEC RETAB
0170  WHEN "ROD"
0180   EXEC ROD
0190  WHEN "VæRDI"
0200   EXEC VÆRDI
0680  NEXT I#
0690  POLY(-1):=0
0700 ENDPROC RETAB
0710 PROC ROD CLOSED
0720  GLOBAL POLY, GR#, ROD, EPS
0730  INPUT "A,B: ": A, B
0740  FA:=FNA(A); FB:=FNA(B)
0750  IF FA=0 THEN
0760   M:=A
0770  ELIF FB=0 THEN
0780   M:=B
0790  ELIF SGN(FA)=SGN(FB) THEN
0800   PRINT "A: ";FA;"      ";"B: ";FB;"ENS FORTEGN!"
0810  ELSE
0820   IF FA)0 THEN A:+B; B:=A-B; A:-B
0830  ENDIF
0840  REPEAT
0850   M:=(A+B)/2; FM:=FNA(M)
0860   IF FM(0 THEN
0870    A:=M
0880   ELIF FM)0 THEN
0890    B:=M
0900   ELSE
0910    A:=M; B:=M
0920   ENDIF
0930  UNTIL ABS(B-A)(EPS
0940  PRINT M;" (";FM;")"
0950  ROD:=M
0960 ENDPROC ROD
0970 PROC VÆRDI CLOSED
0980  GLOBAL GR#, POLY
0990  DIM A$ OF 20
1000  REPEAT
1010   INPUT "START: ": A$
1020   SLUT:=A$=""
1030   IF SLUT THEN GOTO HOPUD
1040   FRA:=VAL(A$)
1050   INPUT "TIL: ": A$
1060   IF A$()"" THEN
1070    TIL:=VAL(A$)
1080    INPUT "STEP: ": A$
1090    IF A$="" THEN
1100     STP:=1
1110    ELSE
1120     STP:=VAL(A$)
1130    ENDIF
1140   ELSE
```

```
0210  WHEN "DIV"
0220   EXEC HORNER
0230  WHEN "DIF"
0240   EXEC DIF
0250  WHEN "KVA"
0260   EXEC KVA
0270  OTHERWISE
0280   //DO NOTHING
0290  ENDCASE
0300 UNTIL PROG$=""
0310 PROC IND CLOSED
0320  GLOBAL IND, GR#, POLY, GROP#
0330  INPUT "GRAD: ": GR#
0340  GROP#:=GR#
0350  FOR I#:=GR# DOWNTO 0 DO
0360   PRINT USING "##: ": I#,
0370   INPUT "": IND(I#)
0380   POLY(I#):=IND(I#)
0390  NEXT I#
0400  POLY(-1):=0
0410 ENDPROC IND
0420 PROC UD CLOSED
0430  GLOBAL POLY, GR#
0440  FOR I#:=GR# DOWNTO 0 DO
0450   PRINT USING "-##: ": I#,
0460   PRINT POLY(I#)
0470  NEXT I#
0480  PRINT "----------------------------------------"
0490  PRINT "REST: ";POLY(-1)
0500  PRINT
0510 ENDPROC UD
0520 PROC HORNER CLOSED
0530  GLOBAL POLY, GR#, ROD
0540  X:=POLY(GR#)
0550  FOR I#:=GR#-1 DOWNTO -1 DO
0560   Y:=POLY(I#); POLY(I#):=X; X:=X*ROD+Y
0570  NEXT I#
0580  POLY(GR#):=0; GR#:-1
0590  PRINT "REST: ",
0600  PRINT POLY(-1)
0610  PRINT
0620 ENDPROC HORNER
0630 PROC RETAB CLOSED
0640  GLOBAL IND, POLY, GR#, GROP#
0650  GR#:=GROP#
0660  FOR I#:=GR# DOWNTO 0 DO
0670   POLY(I#):=IND(I#)
```

```
1150    TIL:=FRA; STP:=1
1160   ENDIF
1170   FOR X:=FRA TO TIL STEP STP DO
1180    PRINT X;": ";FNA(X)
1190   NEXT X
1200 LABEL HOPUD
1210  UNTIL SLUT
1220 ENDPROC VÆRDI
1230 DEF FNA(X)
1240  GLOBAL POLY, GR#
1250  Y:=POLY(GR#)
1260  FOR I#:=GR#-1 DOWNTO 0 DO
1270   Y:=Y*X+POLY(I#)
1280  NEXT I#
1290  FNA:=Y
1300 ENDDEF FNA
1310 PROC DIF CLOSED
1320  GLOBAL POLY, GR#
1330  K1:=POLY(GR#)
1340  FOR I#:=GR# DOWNTO 1 DO
1350   K2:=K1; K1:=POLY(I#-1); POLY(I#-1):=K2*I#
1360  NEXT I#
1370  GR#:-1; POLY(-1):=0
1380 ENDPROC DIF
1390 PROC KVA CLOSED
1400  GLOBAL POLY, GR#
1410  IF GR#<>2 THEN
1420   PRINT "GRAD = ";GR#
1430  ELSE
1440   A:=POLY(2); B:=POLY(1); C:=POLY(0)
1450   D:=B*B-4*A*C
1460   IF D<0 THEN
1470    PRINT "INGEN ROD.  D = ";D
1480   ELSE
1490    IF B<0 THEN
1500     X1:=(-B+SQR(D))/2/A
1510    ELSE
1520     X1:=(-B-SQR(D))/2/A
1530    ENDIF
1540    X2:=C/A/X1
1550    PRINT "X1, X2 = ";X1;"      ";X2
1560   ENDIF
1570  ENDIF
1580 ENDPROC KVA
```

**end**

# Book Reviews

**Computer Consciousness:**
**Surviving the Automated 80's**
**By H. Dominic Covvey/**
**Neil Harding McAlister**
Published by **Addison-Wesley Publ. Co.**
**$5.95, 212 pages**
Reviewed by **Joseph B. Rothstein**

The choice of the title is a curious and somewhat inappropriate one. It raises the expectation that this volume will clarify the impact of the computer on lives during the coming decade – an expectation that is heightened early on when the authors emphatically state "Survival on the job, survival in the classroom, and survival in society depends on it." The computer is an agent of change," they add. "Its uncontrolled application may metamorphose our institutions – not in beneficial ways that correct existing problems, but in ways that we do not intend."

These are powerful ideas, deserving of analysis, thoughtful speculation, and orderly examination. At first blush, *Computer Consciousness: Surviving the Automated 80's* might seem to be an attempt at coming to grips with these social, political and personal winds of change.

But such is not the case, for what we find instead is a guide to evaluating and purchasing a computer system, albeit a well-organized and systematic guide. This particular case of false advertising may be forgiven, for it would take far more than this one slim volume to fully evaluate or forecast the changes which our society will undergo during the coming decade as a result of the spread of automation and machine intelligence – and also to provide a tutorial approach to computer hardware, software and systems integration.

The authors have pursued the second course, and the tutorial approach is the strength of this text. There is no fat in this volume, and the novice reader may soon find himself or herself overwhelmed by the sheer quantity of information. Apparently mindful of this possibility, the authors have thoughtfully provided, at the end of each chapter, a list of new words encountered in the chapter, as well as an index and brief glossary at the end of the book.

While first-time buyers of an appliance computer intended primarily for games, children's education or household use might find more in this book than they can or want to handle, professionals or institutions contemplating a big-ticket purchase would do well to arm themselves with the knowledge this book provides and points to. Even a computerist with some years of experience may find this text useful as a brush-up reference, including as it does capsule definitions and descriptions of a wide variety of hardware devices, software tools, and systems options.

The authors have done a creditable job of organizing such an encyclopedic body of information. A brief introduction describes the book as an attempt to compile a large amount of general introductory material from widely dispersed sources, and offers a quick, descriptive overview of hardware and software, and their place in a computer system.

Subsequent chapters examine functional aspects of these large divisions. First, hardware – ranging from input and output devices to memory devices, CPU's and telecommunications options, and closing with considerations for the hardware shopper. Next, software is examined from the ground up – machine language, assemblers, operating systems, high-level languages and applications packages, and how to choose software. Finally, the computer system as a whole is briefly discussed, including real-life problems and considerations, personal requirements and potential difficulties, and considerations in selecting a vendor.

The emphasis is on breadth, not depth, and the business person or department head would be foolish to think that any one book can provide a thorough basis for evaluating increasingly complex alternatives of hardware, software, and human systems. But as a place to start, this slim volume covers most of the concepts, options and jargon that the system user or automation-minded manager would be likely to encounter. It may not answer – or even ask – the larger questions implicit in its title, but if survival is the goal, then a catalogue of the armaments and combatants is a good place to start.

**The MC6809 Cookbook**
**By Carl D. Warren**
Published by **TAB**
**$6.95, 176 pages**
Reviewed by **Richard L. Lozes**

Recently we have seen the introduction of a few microprocessors which have been hailed as interim processors between the 8-bit and 16-bit designs. Perhaps it is better to view these processors as more balanced and better designed versions of their ancestors. Certainly, their instruction sets reflect the awareness that software has to be written for these chips, and that the early microprocessor instruction sets made that task more difficult than necessary.

The MC6809 is a good example of these mature 8-bit microprocessors. Continuing in the style of previous "cookbooks," Carl Warren discourses on the 6809 function.

I was disappointed by Mr. Warren's treatment of the topic. While the chapter on the instruction set certainly has much to recommend it on the basis of regularity of format of the information on each instruction, those aspects of the book which are less mechanical to produce did not fare as well. Even the spelling and grammar reveal poor editing despite the fact that the book was typed originally on magnetic media.

On to specifics . . .

The timing diagrams are few and cramped. I know, from other Motorola data sheets, that the information necessary to design a system around a CPU chip is nearly all on the sheets. Certainly this part of the book must be supplemented.

Chapter three on addressing modes is too brief, and poorly written. "I'm reasonably sure that on this first reading you haven't grasped everything that was presented," says Mr. Warren. Apparently he was too interested in seeing an early publication date to rewrite the chapter in order to make it understandable. After all, what is the point of authorship?

One may as well not bother with the six pages devoted to the MEK6809EA assembler. Again, other sources must be consulted. Apparently, the author's pet version of bastard BASIC (VTL-09) took precedence, even though it is so cramped as to be nearly useless.

In closing, I would suggest that one browse through this book rather carefully to first ascertain whether it contains the information desired before parting with $6.95 to buy it. I would not spend my money on it.

**Microcomputer Interfacing Handbook**
**A/D & D/A**
**By Joseph J. Carr**
Published by **TAB**
**$8.95, 350 pages**
Reviewed by **Richard L. Lozes**

"We hope that most copies sold will become shop tools in laboratories and other facilities – a well-worn shop

tool." Indeed they will! Mr. Carr has penned an excellent book. It is thorough, well-organized, and carefully edited.

Although interfacing, the titular topic, is not mentioned explicitly until page 270, the first 269 pages are indeed essential. After reading them, one knows not only how A/D and D/A converters work, but also how to select devices for an application, and how to avoid pitfalls in designing a conversion system.

It is not as simple as picking up a chip or two at the local electronics store and wiring it onto the microcomputer bus. To design a workable system of modest expense, one needs to consider such things as how many signals will be measured (generated) simultaneously, how often, how accurately, what sort of buffering, level shifting or filtering might be required, etc. With these parameters established, one can begin to apportion the tasks among various pieces of hardware and software, and then specify components. For example, a cheap and slow-running conversion system could utilize spare CPU cycles to do much of the conversion work (assuming a cheap programmer were available), while a system operating in the MHz range would almost certainly require a pipeline of high speed hardware.

This book is well rounded. It left me with an appreciation of the finer details of conversion as well as of the systems view. I was particularly pleased by the latter, since it is the general structure which is most difficult to conceive and which determines most of the crucial parameters, such as cost and reliability, in any design endeavor. (That recognition, incidentally, is what engendered "structured design" in computer software.) It is also rather self-contained, including ancillary material such as a chapter on analog filters.

The author has wisely chosen to include only such theory as necessary for the reader to parametrize his converter by analogy to the explicitly solved examples in the text. It helps if the reader knows Ohm's law, the formulas for series and parallel resistances, and similar simple electrical relations. It also might help to be able to recognize the simple integral occurring in RC circuit behavior theory. Other required background is minimal.

With this book, one may not be able to design cutting edge applications, but the vast majority of A/D and D/A conversions will be within one's reach. ■■


**You Read**

# Dr. Dobb's Journal

**And You Don't Subscribe?!**

**Save $13**
**off newsstand prices for 2 yrs.**
**Save $5 for 1 yr.**

Can you afford to miss an issue with information vital to your interests? As a subscriber you can look forward to articles on Small-C, FORTH, CP/M, S-100, Compiler optimization, Concurrent Programming and more, delivered right to your door. And you'll never miss the issue that covers your project.

**DR. DOBB'S JOURNAL**
**P.O. Box E**
**Menlo Park, CA 94025**

**Yes!** Sign me up for

____ 2 yrs. $47 ____ 1 yr. $25

____ I enclose a check/money order
____ Charge my Visa, MasterCard, American Express
____ Please bill me later

Name _____

Address _____

_____

_____ Zip _____

Credit Card _____ Exp. date _____

Account No. _____

Signature _____

# Installing a Printer Toggle in North Star DOS 5.2


## by Bert Smart

*Bert Smart, 639 Norvell St., El Cerrito, CA 94530.*


It was not long after purchasing CP/M 1.44 a few years ago. I discovered one of the great features supplied with CP/M was the ability to toggle the printer on so that I could print out simultaneously with the console output. This was done by typing control P (↑P). Another very handy feature was being able to stop screen paging by typing control S (↑S).

So at that time I implemented in the North Star DOS 5.0 the control P for the printer toggle and space bar to stop paging.

Now with North Star DOS 5.2 available at 100 hex, the need to redo the DOS drivers was mandatory. Since I did not have the parity option on my North Star dynamic memory boards, I decided to use that space in the DOS drivers to hold the necessary changes for the implementation of control P for the printer toggle and space bar for stopping paging.

To do this, do the following:

1. Insert the copy of your new North Star DOS 5.2 disk and boot up.
2. Load a copy of DOS at 4000H by typing +LF DOS 4000 (CR).
3. Load your monitor so the necessary changes can be made in the DOS image at 4000H. Type +GO M0E00 (CR). The monitor will respond with a > prompt.
4. Type >DOS 4859 (CR). The monitor will respond with the hex address and the existing byte. Change the next four bytes to be as follows: C3 01 0A C9 (CR).
5. Type >DS 48E7 (CR). Change the next four bytes to be: C3 15 0A C9 (CR).
6. Type >DS 48FC (CR). Change the next four bytes to be: C3 29 0A C9 (CR).
7. Type >DS 4800 (CR). Change the next thirty-four bytes to be: C9 DB 02 E6 7F FE 10 CA 0B 0A C9 3A 34 0A 2F 32 34 0A C3 4E 0A 78 D3 02 3A 34 0A FE FF CA 22 0A 78 C9 3E 01 CD DA 0A 78 C9 FE 03 37 FE 20 CA F9 0A C9 00 00 (CR).
8. Now exit the monitor by typing: >OS (CR).
9. Now save the changed DOS to your disk by typing: +SF DOS 4000 (CR).

The above changes have done the following: changed the code at 0A59 to a jump to 0A10, at 0AE7 to a jump to 0A15 and at 0AFC to a jump to 0A29. The code previously resident at the foregoing addresses was relocated to the address 0A00 along with the new code for the control P printer toggle and the code for the space bar to stop the paging. If you have very carefully followed the above changes and your system is configured in the standard North Star DOS, it will work just fine. As always, a word of caution: be sure that you are working on a *copy* of the original and not on the only master you have. ■■

```
?
+LF DOS 4000
+GO M0E00
MONITOR 5.2
>DH 4859,4
4859  C3 01 0A C9
>DH 48E7,4
48E7  C3 15 0A C9
>DH 48FC,4
48FC  C3 29 0A C9

>DH 4800,34
4800   C9 DB 02 E6   7F FE 10 CA   0B 0A C9 3A   34 0A 2F 32
4810   34 0A C3 4E   0A 78 D3 02   3A 34 0A FE   FF CA 22 0A
4820   78 C9 3E 01   CD DA 0A 78   C9 FE 03 37   FE 20 CA F9
4830   0A C9 00 00
>OS
+SF DOS 4000
+
?
+
```

### H8 Text Editor

*(Continued from page 18)*

Also, I have noticed two minor bugs:

1. The input data buffer, DATABUF, is too short. It probably should be about 1000 bytes long, instead of 128. However, being at the end of the program, it is not likely to make any difference.

2. After the BADTAPE label, the following instructions should be inserted, to turn off the tape in the event of a tape sequence error:

```
XRA  A
OUT  OPTPC ;TURN OFF TAPE.
```

After you have converted your files, don't forget they may need some modification with the CP/M text editor. In particular, there are some minor differences between the format used by the Heath assembler and that used by the CP/M assembler. Also, the converted file may take up slightly more space than necessary, because it will take advantage of neither Heath's space compression nor CP/M's convention of storing unconverted Tab characters (control I) on disk.

■■

**(LISTING ON PAGE 40)**


Why wait
to start
increasing your sales?
Call now
to reserve
advertising space
in the next
*Dr. Dobb's Journal*
(415) 323-3111

# Dr. Dobb's Clinic

by D. E. Cortesi, Resident Intern

This is the second appearance of *Dr. Dobb's Clinic*, which is a place for the display of techniques, discoveries, and pitfalls. We want to cover any sort of method or trick you've found in your exploration of any sort of personal computer software. The Clinic is the place to expose errors in published documentation, to warn others away from pitfalls, and above all to show off those *eureka!* moments that make systems work so rewarding.

The first Clinic used up all the material we had on hand. We scrambled to find some more for this one (and it's pretty thin, too). The Clinic needs material, or it will have to be dropped. Where will that material come from? From you, the readers, or not at all. The requirements are very simple, much easier than those for an article.

Write to us. Describe your discovery or technique in a couple of pages, in any format you like so long as we can read it. Don't worry about being "literary." Don't bother with a tutorial introduction for the novice reader; this isn't a column for novices. Do include a daytime phone number so that if we have questions we can call you.

Special note to OEMs: if you have any interesting application notes or user tips that you feel deserve wider circulation, send them to the Clinic. If they're of general interest, we can use them.

## A Small Discovery

We were working on a game program, using Microsoft Disk BASIC 5.0 on a CP/M system. In this game, as in many others, the program needed to spin along updating a simulation, while simultaneously watching the keyboard. As soon as the player pressed a key the program was to respond, then go back to its work.

The INKEY$ function was designed to solve just such problems. It shows up at the heart of most games for the TRS-80 and PET. But we weren't sure – did this big, business-like BASIC for CP/M have INKEY$? After all, the interface to the keyboard of a CP/M terminal isn't anything like that of the smaller machines. A quick search of the Microsoft manual confirmed our suspicions: INKEY$ isn't listed, and therefore doesn't exist.

Hours of labor went into proving that this particular game could not be made playable without something like INKEY$. At last, in a fit of flakey frustration, we coded it anyway and typed "RUN". And it ran! Disk BASIC 5.0 for CP/M does indeed have a perfectly good, working, INKEY$ function. It just didn't make it into the manual, that's all.

## The Lost Chord

It is often useful to know when a binary integer contains only a single 1-bit; that is, when it represents an integer power of 2. Suppose you were working on a binary division or multiplication routine. If you knew in advance that the multiplier or divisor had only a single 1-bit, you could perform the operation entirely with shifts and gain a lot of speed. Of course, the test for a power of two would have to be very short, or you couldn't afford to test every divisor or multiplier.

Some years ago, we saw the code for just such a test. It did not loop; it was a short piece of straight-line assembly code (the machine was an IBM 370, but that shouldn't make any difference). It used an exclusive-or, or perhaps more than one. We recall thinking at the time "Gee, that's clever; have to remember that one." Recently we needed it and of course can't remember it any more.

Can a reader re-invent that clever test? The code should accept a 16-bit integer and return a suitable signal (maybe a flag setting) when the integer contains exactly 15 0-bits and a single 1-bit. Express the solution in any common assembly language. We'll find some suitable prize for the best one(s).

## Decimated

It's long bothered this writer that, while it is very easy to multiply a binary number by 10, it is very difficult to divide one by 10. The multiplication algorithm, as most every programmer knows, is based on the relationship between multiplication and addition. That is,

N * 10

can be factored out to

N+N+N+N+N+N+N+N+N+N

and that in turn factors to

2 * ( ( 4 * N ) + N )

In binary, once you have all the factors down to integer powers of two, you have the problem licked. The last expression becomes (using the C notation for a left shift)

( ( N << 2 ) + N ) << 1

and Bob's your uncle. If the universe had one jot of fairness, division by 10 would be as simple. Something using right shifts and subtracts would do nicely. The universe persists in paying no attention to the needs of programmers; division by 10 seems to require the use of a complete, tedious binary division routine.

The problem is, programs convert decimal digits to binary only once in a while, and almost never in a central processing loop where time counts. They convert from binary to decimal for output much more frequently, and division by 10 is the central feature of binary-to-decimal conversion.

It is possible to write a divide-by-10 routine that is faster than the usual 16-bit divide function. Such a routine can take advantage of its knowledge that 10 is the only divisor. Rather than a full 16-into-16 or 8-into-16 division it can divide the 16-bit dividend by a 5-bit divisor. This allows it to get away with only 12 iterations rather than 16, and to use byte rather than word arithmetic. Listing 1 shows such a routine. It requires just over 1000 clock periods on an 8080. The 16-into-32 bit routine in [1] uses over 1500 Z80 clock periods; the 16-into-16 bit routine in [2] takes 2300 clocks.

Listing 2 shows a binary to decimal conversion routine that calls the routine in Listing 1.

Still, there ought to be something better. Numerical analysis is called for here. It is possible to represent a binary number in a generalized way as a polynomial. For instance, this:

$f(X) :: aX^7 + bX^6 + \ldots + fX^2 + gX + h$

stands for any 8-bit number (given that X=2). The coefficients a, b, . . ., h are either 1 or 0; evaluate f(X) to get the value of the number. It takes a certain shift of vision to see it, but in fact a byte in your computer doesn't contain bits, it contains one set of eight coefficients for that expression.

When the coefficients are known, all the zero terms can be dropped out. Ten is 1010 binary, so the polynomial for ten is just

$t(X) :: X^3 + X$

and binary division by ten can be stated in general terms as f(X)/t(X). That division can be carried out in symbolic form by anyone masochistic enough to do so. The result for the symbolic quotient is:

$q(X) :: aX^4 + bX^3 + (c-a)X^2 + (d-b)X + (e-c+a)$

and the symbolic remainder is:

r(X) :: (f-d+b)$X^2$ + (g-e+c-a)X + h

Frankly, it doesn't give much help. The expressions are true enough. Check them yourself. Pick a binary number and substitute its bits 7 . . . 0 for the letters a . . . h respectively. Substitute 2 for X and evaluate; you'll find that q(X) is the quotient of division by 10 and r(X) the remainder. But it doesn't suggest any clever, non-looping algorithm – or is that just a failure of imagination?

Perhaps a reader can find a better analysis of the problem than we can, or perhaps one knows a better cut-and-try algorithm than the one in Listing 1. If so, share it with us, please! ■■

## Listing 1 – Fast Divide-by-10 routine.

```
;
;           fast – divide – by – ten routine
;
; A version of the standard 16 by 8 bit divide which
; performs a division by ten, most useful in converting
; binary to decimal (or BCD). Takes advantage of the
; fact that 10 is a four-bit value, effectively divides
; a 5-bit signed value 10 into a 16-bit signed number.
; Handles case of 8000h as 32768 – not strictly true,
; but that ambiguous value has to be handled somehow.
;
;  Input:  HL = number to be divided – must be in
;               range of 0 . . . 32768 (0000 to 80000).
;
;  Output: HL = quotient (input/10)
;          A  = remainder (input mode 10)
;
; BC and DE preserved; flags unpredictable.
;
FASTDIV10:
        PUSH    D           ; save 2 work registers
        PUSH    B
        MVI     E,0         ; clear start of quotient
        MVI     C,12        ; iteration count
        MVI     B,50H       ; 5-bit divisor (10 shl 3)
;
FASTDIV1:
        MOV     A,H         ; see if high 5 bits of
        SUB     B           ; . .dividend exceed 10.
        JC      FASTDIV2
        MOV     H,A         ; they did, update dividend
        INX     D           ; . .and insert 1-bit in quotient.
FASTDIV2:
        DCR     C           ; are we done?
        JZ      FASTDIV3
        XCHG                ; no, shift quotient left 1
        DAD     H
        XCHG
        DAD     H           ; . .and shift the dividend.
        JMP     FASTDIV1
FASTDIV3:
        MOV     A,H         ; remainder is in high 5 bits
        RRC                 ; of the old dividend –
        RRC                 ; . .right-justify it in
        RRC                 ; . .the a-reg.
        XCHG                ; restore quotient to HL,
        POP     B           ; . .all set for next call.
        POP     D
        RET
```


**References:** [1] Barden, William Jr., *The Z-80 Microcomputer Handbook*, Howard W. Sams Inc., 1979.

[2] *8080/8085 Assembly Language Programming*, Intel Corporation manual number 9800949, 1979.


## Listing 2: Binary-to-decimal conversion routine.

```
;
;          binary to decimal-character routine:
; Stores the decimal representation of number in HL
; in the six bytes at [DE+]. The output field should
; have been blanked first, for only as many bytes are
; written as are necessary.
;
; Fastdiv10 produces the digits in reverse order, most
; significant last. Call it until HL has been reduced
; to zero, stacking digits and counting them. Then pop
; and store them, thus getting them high- to low-order.
;
;  Input  =    HL: the number to be converted
;              DE -> the first byte of the field
;
;  Output =    DE advanced to byte following last
;              digit. Field has blank/minus, digits.
;
;
; A, flags unpredictable. HL, BC preserved.
;
CVTDECIMAL:
        PUSH    H           ; save working registers
        PUSH    B
        MOV     A,H         ; check sign
        ORA     A
        MVI     A,BLANK     ; . .assume it's positive
        JP      CVTDEC2     ;  (it is)
        MOV A,L ! CMA ! MOV L,A  ; it isn't. .
        MOV A,H ! CMA ! MOV H,A
        INX     H           ; . .form two's-complement
        MVI     A,MINUS     ; set up negative sign
CVTDEC2:
        STAX    D           ; store the sign character
        INX     D
        MVI     C,0         ; C will count digits pushed
CVTDEC3:
        CALL    FASTDIV10 ; A <-- low-order digit
        PUSH    PSW         ; . .save it on stack
        INR     C           ; . .and count it.
        MOV     A,L
        ORA     H           ; quotient reduced to zero?
        JNZ     CVTDEC3     ; . .no, repeat.
CVTDEC4:  ; there are now [C] digits on the stack.
        POP     PSW
        ORI     '0'         ; get digit, make printable
        STAX    D           ; store it down
        INX     D
        DCR     C           ; count it off
        JNZ     CVTDEC4     ; repeat 1 to 5 times.
        POP     B
        POP     H
        RET
```

# Of Interest

by Dave Cortesi

Be glad you don't know the hordes of vendors that are clamoring for your attention. Nearly a bushel of press releases was winnowed to arrive at this month's selection. A number of good items went to the round file because they brought time-dependent information too late. *DDJ* has one of the shortest publishing cycles in the field, but even we can't help the publicist for a May event who mails his releases in March. Three months is the minimum lead-time for any announcement; four is better.

## One Fantastic Weekend

One event that got its material out in good time is **PACIFICON**, the Seventh Annual Wargaming Convention. It will be held in San Mateo, California on July 3-5. "Fantasy and science fiction games, tournaments and demonstrations will occur around the clock," we are told. Not strictly a computer event, but how could any Adventure player resist?

## The Spring Lists

The major publishers are out with their spring lists, and several titles are worth noting. We won't list "contact points" for publishers; look for these books at your usual bookstore.

Addison-Wesley has released the second edition of Donald Knuth's *Seminumerical Algorithms*, Volume 2 of that author's *Art of Computer Programming*. Any reader of this magazine should be familiar with that excellent series, distinguished among textbooks by the author's wit, erudition, and graceful prose. This new edition is the first book prepared with Knuth's TEX and METAFONT programs, described in this magazine a few issues back.

McGraw-Hill has *Digital Harmony* by John Whitney. Whitney, it seems, is working toward the synthesis of computer graphics and computer sound into a single new medium of artistic expression. The book is said to describe his computer, peripherals, and film techniques," and to contain many color plates; the price is $21.95.

The same publisher has what sounds like a gem: *The Devil's DP Dictionary* by Stan Kelly-Bootle, "a spoof dictionary modeled on Ambrose Bierce's classic, *The Devil's Dictionary*." It defines computer science as "a study akin to numerology and astrology, lacking the precision of the former and the success of the latter," and a random file as "one in which records may be lost in any order." 160 pages of this costs $7.95; buy a copy before anyone else and build your reputation as a wit.

Osborne McGraw-Hill has released the *Apple II User's Guide* by Poole, McNiff, and Cook ($15), and claim it is "the definitive guide to the Apple II and Apple II Plus computers." That word "definitive" is hard to swallow, but the list of topics sounds comprehensive. Osborne has also brought out *6809 Assembly Language Programming* by Leventhal ($17), a timely addition to their series of such books. Now, when can we have one for the 8086?

## A Mainframe Item

In the world of gigabytes and megaflops, there are several publishers who sell large libraries of FORTRAN subroutines. **IMSL**, Inc. is one such. They wrote to tell us that they've released Edition 8 of their library, with over 500 math and statistical subroutines, well-tested and predictably accurate. They're proud of the fact that they have 1100 installations. We wonder if they can conceive of the size of market that awaits a personal-computer edition of such a library.

## A FORTRAN Daisy-Driver

**Escape, Ltd.** has prepared a library of plotting subroutines in FORTRAN. This library is designed for personal machines, having been prepared under Microsoft FORTRAN and then tested on a larger machine to verify ANSI compatibility. The subroutines are designed to make it easy to build line and bar graphs from a FORTRAN program, merge the graphs with text, and output the whole on a Diablo/Xerox daisy printer. The package, including source code, costs $600.

## A FORTH for OSI

The spread of FORTH is beginning to remind us of the inexorable growth of some jungle vine. An outfit called **FORTH Tools** has prepared a fig-FORTH implementation for the OSI C1, C2, and C4 systems with 5" disks. The system runs stand-alone, but uses the OS-65D disk format. They claim strict compatibility with the FORTH Interest Group model of the language, including disk words. A 6502 macro assembler is part of the $50 package.

## Bisynch for Oasis

**Phase One** has prepared a Bisynch communications line driver for their Oasis operating system. The package is said to provide exact emulation of the IBM 2780/3780 terminals, and costs $450. Phase One didn't note it, but we will: this opens the way to inserting an Oasis-controlled system into an IBM-based Remote Job Entry network. IBM's RJE networking protocols allow job sharing and load balancing to be done across many CPUs. Sticking in a micro opens a lot of interesting possibilities for synergism between the small and large machines. Go for it, Phase One: build a SNA driver.

## Two for the North Star

What is it about North Star BASIC that makes people want to compile it? **DATEK** is the latest company to announce a compiler for that dialect. Theirs compiles to relocatable modules of 8080 code. Assembler statements may be inserted in the BASIC source as REM statements. An assembler is included, from which we deduce that the compiler produces assembler source rather than generating the object code directly. The package required double or quad density disks, supports DOS 5.2DQ, and costs $400.

The same company has what they believe to be a superlative word processing system for the North Star. It includes full-screen editing capability for many terminals, and support for Diablo and NEC daisies. No mention here of proportional spacing, but most useful formatting options are available in the $495 price.

## Bargains for CP/M

The **Software Toolworks** has several CP/M programs for sale at very attractive prices. They offer C/80, a compiler for a subset of the C programming language, an implementation of LISP called LISP/80, and a RATFOR preprocessor whose output is acceptable to Microsoft FORTRAN. These and other programs are "priced in the $20–$40 range." Right on.

Those who make extensive use of MAC and RMAC might like a product from **Spun Software**. RLIST is a listing formatter that converts the .PRN and .SYM files written by those assemblers into a more handsome .LST file. The out-

put includes a true cross-reference; printing can be suppressed for portions of the file such as unproductive sequences of macro statements. The price is $35.

**Elliam Associates** have a text retrieval system they call Information Master, and sell it at the reasonable price of $37.50. They have a pre-indexed file of magazine articles to go under it for $10, and have now added a file indexing all the programs available on the CP/M Users Group diskettes for $15 ($20 separately).

## TRS-80 Secure from Prying Eyes,

The Locker is a software extension to TRSDOS 2.0 that makes the LIST and LLIST commands ineffective. The objective is to make your proprietary BASIC programs runable but un-viewable. **Management** sell it for $25, and claim that it goes in simply and is trouble-free.

## . . . Gets Analog Input,

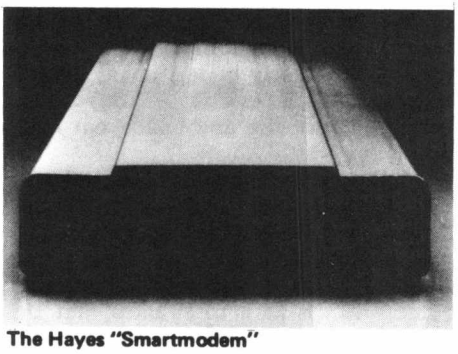
**Connecticut microComputer**, who make a line of analog input devices for small machines, have added the TRS-80 to the list of machines they serve. They have a unit that connects their analog devices to the TRS-80's expansion or printer port for $60. Into it they want you to plug their 16-channel analog input module, or their driver for the BSR X-10 home controller.

## . . . Goes on the Air

A ham possessed of a TRS-80 can put the machine on the air as an automated mailbox with the M8300 RTTY software package from **Macrotronics**. The System stores and forwards messages and can be used with a ham repeater. Messages sent or received can be accessed from disk with any of the common text editors. I/O can go via the same company's ham interface hardware or through the machine's RS232 board and a modem.

## . . . With Either of Two Modems

Speaking of modems, we've had announcements of two new ones. **Hayes** have announced what they call the Smartmodem, a direct connect modem that is controlled by commands expressed as strings of ASCII characters so that it may be driven easily from a program in a high-level language. It responds to the commands with result codes that may be either decimal numbers or English words. The Smartmodem has both auto-answer and auto-dial features, and can dial with either pulses or tones, at program command. The $280 price includes a power pack and a telephone cable.



**The Hayes "Smartmodem"**

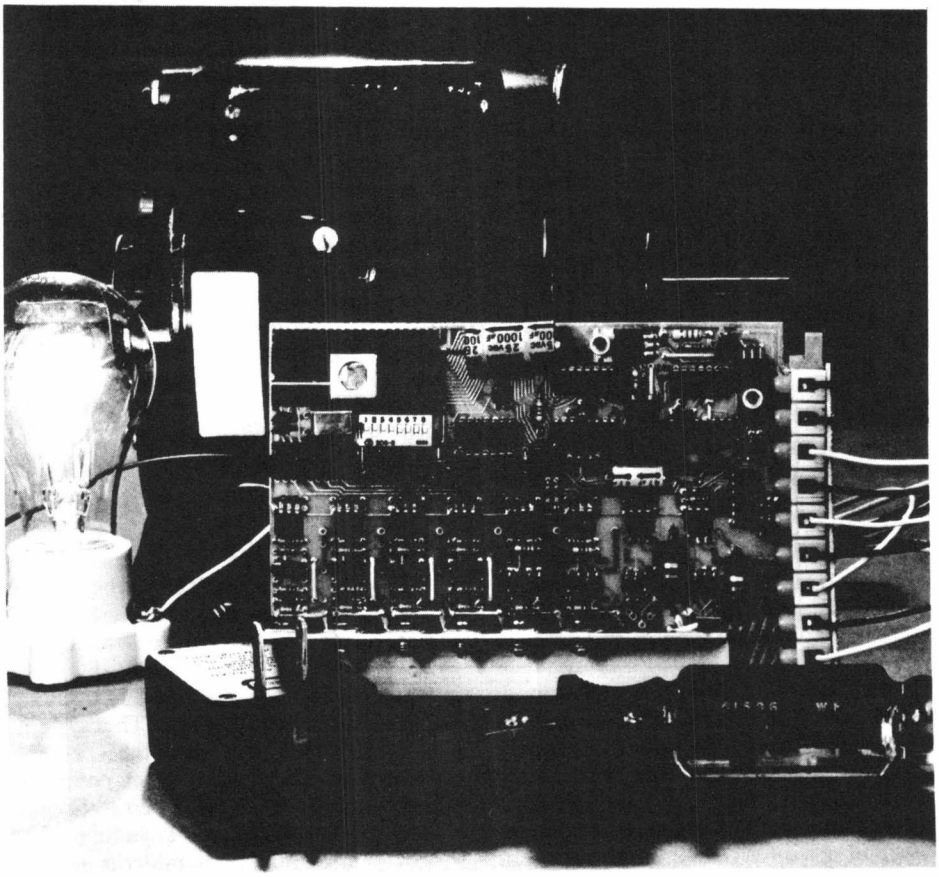
**The Microperipheral Corp.** wants you to know about its Microconnection devices, direct connect modems for the Atari 400 and 800 and for all TRS-80 models. All units have provision for connecting a cassette recorder to the modem for recording the on-line conversation. The Atari version does not require the use of Atari's RS232 peripheral unit, and does contain a serial printer interface allowing hardcopy of the communication. The TRS-80 units cost $200; that for the Atari is priced at $250. An auto-dial, auto-answer option is $80.

## Boards for the SS-50 Bus

Two companies have popped out with new hardware for systems built around the 68xx CPUs and the SS-50 bus. **Gimix**, who've been at it for a while, are offering a choice of three different disk controller boards. Their fanciest one uses DMA for data transfers and supports any mix of drive diameters and densities. It's priced at $550; less competent boards are available at $350 and $230 for various hardware configurations. Gimix supplies its version of FLEX for the 6809 with each.

**Percom** have prepared a two-channel serial I/O card for SS-50 machines. They claim it will work with all versions of the SS-30 (I/O portion of the SS-50) bus. The card can have its own bit rate generator, which would apparently otherwise require a card of its own. No price was given in Percom's release. ■■

**WINTEK's single-board computer.**

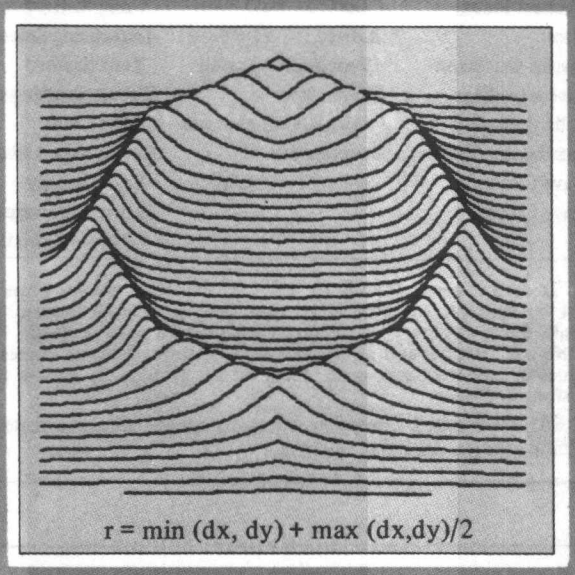


$2.50
($3.00 in Canada)

# DR. DOBB'S JOURNAL

For Users of Small Computer Systems

Number 57 July 1981 Volume 6, Issue 7




**Pidgin — A Systems Programming Language**
*William A. Gale*

**Time on the Horizon**
*D. J. Yates*

**BASIC Rubik's — A Cube Simulator**
*J. W. Pehoushek*

**....and much more!**

## For Rubikubists an Ally in BASIC

## Moving Toward Hardware Independence

## The Doctor's Problems Minimized!

DR. DOBB'S JOURNAL
For Users of Small Computer Systems

July 1981 Volume 6, Issue 7, Number 57

# Contents

## Articles

**Pidgin – A Systems Programming Language** . . . 10
*by William A. Gale*
**Time on the Horizon** . . . . . 15
*by D. J. Yates*
**BASIC Rubik's – A Cube Simulator** . . . . . 19
*by J. W. Pehoushek*

## Departments

Letters . . . . . 4
Of Interest . . . . . 38
Clinic . . . . . 40
Book Reviews . . . . . 43

**Publisher**
Michael Madaj
**Editor**
Marlin Ouverson
**Assistant Editor**
Julie Anton
**Contributing Editors**
Dave Caulkins
Dave Cortesi
**Art Director**
Clifford West
**Artist**
Barbara Ruzgerian
**Typesetter**
Michelle Moonstone
**Proofreader**
Nancy Heubach
**Marketing Director**
Craig S. Harper
**Advertising Sales**
Kent Safford
**Circulation Manager**
Peter Clark
**Circulation Assistant**
Leah Dansby
**Business Manager**
Maureen Christine

*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025.

© 1981 by People's Computer Company unless otherwise noted on specific articles. All rights reserved.

# Editorial

First things first. Our cover has changed in order to reflect a new approach to the magazine publishing business. We are not selling out to the highest bidder; *DDJ* has always been at the cutting edge and is staying that way. Doc won't settle for less!

The real change has been in the computer hobbyist market. No longer a group of bleary-eyed hackers, we have acquired importance by filling people's needs. Those needs have grown along with the numbers of small computers and their owners, who need to know where they can get answers and inspiration. We hope more folks will pick up this journal, learn from it and contribute to it. Ours is an important work and if we can further it by cleaning house a bit, then 'tis done.

*DDJ*'s new cover answers one more need. We got plain tired of answering mail from orthodontists and dentists who were inquiring about application software for members of their profession! Doc, as you know, is very much a general practitioner when it comes to computers.

Next point: while every other magazine has been talking about ways to produce a customized publication for subscribers, Dr. Dobb's has been just that since the beginning. All articles, reviews, essays and programs are researched and written by readers, not by professional journalists. The contents are determined by what our readers express to us. The rapid turn-around time from our receipt of an article to its publication is so short (sometimes a matter of weeks) that it can be dizzying. But it enables us to be very responsive to our subscribers.

We would like to hear from some more potential book reviewers. If you would like to help Doc and get your literary feet wet at the same time, send a stamped, self-addressed envelope with your request for writer's guidelines. Don't be shy – and don't underestimate the importance of good book reviewers. In a field with so many new titles coming out daily, someone needs to sort through and evaluate each for accuracy, relevance and significance. Let us know your background, interests, hardware and software experience. We'll try to find something to intrigue you!



Marlin Ouverson
Editor

# Letters

## Still Alive and Well . . .

Dear Marlin,

I cannot allow Mr. Bill Bolton's letter to *DDJ* in the May 1981 issue to go unanswered. He was writing in response to the review of our software package, ZDM, which was published in your February 1981 issue. Mr. Bolton seemed to imply that a Z-80 debugger (ZDM) is useless unless it displays Zilog mnemonics and states that "new Z-80 software should be using Zilog mnemonics, not mnemonic sets from companies long out of business".

First, let me state that TDL software is far from "out of business". It is very much alive under the name of CDL – Computer Design Laboratories. Second, there are a large group of Z-80 users out there who grew up on the Intel 8080 and still prefer an extended 8080 set to the logical, but yet often confusing, Zilog mnemonics.

RD SOFTWARE recognizes that the marketplace can support both Zilog *and* extended 8080 mnemonics. Accordingly, since December 1980, we have been offering *both versions* on a *single disk* i.e., ZDM (extended 8080, and ZDMZ (100% Zilog).

Obviously, Mr. Bolton was not aware of this due to the time lag from software submittal to review publication.

The point is that there is still a market for Z-80 programs employing an extended 8080 mnemonic set as well as standard Zilog.

Very truly yours,
R. F. Doolittle
RD SOFTWARE
1290 Monument Street
Pacific Palisades, CA 90272

## Lingua Franca

Dear Marlin:

Dobb's ex machina vincit omnia! With a hearty hi-ho Silver! Etc., etc. Received today one complete package freefer-nothin' of PET service kit. I enclose a copy of Commodore's letter and my t.u. letter. Thank *you, DDJ,* and if you can get IMSAI/Fischer-Freitas to get off the stack as well . . . what can I say? Many, many thanks, from your . . .

Faithful servant and friend and fan, etc.,
John Thayer Jensen
P. O. Box 358 Yap
Caroline Islands 96943

## Small-C: Bug-Fix Bug

Dear *DDJ:*

Bravo for your recent series of articles on Small-C. While implementing the formal parameter correction suggested by P. L. Woods (*DDJ* #52), an error in the bug fix was discovered. The error had the effect of causing incorrect relative stack offsets to be calculated if the declarations for formal parameters were done with more than one type declaration statement. To correct this, the following changes should be made to the "NEWFUNC" and "GETARG" routines that were presented.

Changes to "NEWFUNC":

Add a local declaration "INT ARGTOP;".
Add the statement "ARGTOP= ARGSTK;" between the "SP=0;" and "WHILE (ARGSTK)" statements.
Add a second formal parameter ("ARGTOP"), to all calls to "GETARG". Ie. change "GETARG (CCHAR);" to "GETARG (CCHAR,ARGTOP) ;" and change "GETARG (CINT) ;" to "GETARG (CINT,ARGTOP) ;".

Changes to "GETARG":

Change the function header from "GETARG(T)" to "GETARG (T,ARGTOP)".
Add the formal parameter declaration "INT ARGTOP ;".
Delete the "ARGTOP= ARGSTK ;" statement.

Additional note, a descendent of Small-C called Small C Plus is available from Alpha Omega Computer Systems, Inc., for $55 on 8" CP/M compatible diskette. This version adds the following features to Small-C:

- –argv/argc command line parameter parsing (allows compiles to be done from CP/M submit files).
- –the addition of "for" and "dowhile" loops.
- –the "switch-case-default" statement.
- –extensions to the compiler to allow separate compilation of function modules in conjunction with Microsoft's M80 assembler.
- –numerous bugs have been fixed.

CP/M is a trademark of Digital Research, Inc.

Yours truly,
Kirk Bailey
Alpha Omega Computer Systems, Inc.
P. O. Box U
Corvallis, OR 97330

Dear Sirs,

We would like to bring to your attention a bug in the code submitted by P. L. Woods in your February issue.

The correction is rather simple, yet the error is serious. Mr. Woods assigns ARGTOP = ARGSTK in the function getarg (); since ARGSTK is being decremented in the very same function, ARGTOP will change also! It only happens that the example Mr. Woods tried works.

The simplest way to fix the problem would be to assign a global variable to ARGSTK before the first call to getarg (), then use that global variable rather than ARGTOP to correct the stack offset.

Another solution (shown below) is to pass an extra parameter to getarg(), so the calling sequence would pass ARGTOP into a local variable in getarg ().

In the listing below, there are some differences in the label names and a few function names. These differences should not detract from the corrections.

If anyone out there would be interested in a working version of the small c compiler (with extensions such as for (;;) loops, do . . . until loops, switch (or 'case') statements, nested levels of inclusion, octal and hex constants, more explicit error messages, and other improvements, they should get in touch with Intersoft Unlimited, 30 Jills Court, Barrie, Ontario, L4M 4L7, Canada.

The compiler currently runs on the TRS-80 Model I and Model III, producing assembler code as output that is directly compatible with Microsoft's popular M80 assembler. An Edtasm-compatible version should be available soon.

Work is progressing well on a version for CP/M, as well as a 6809 version to run under Flex and OS-9. The price is $60 U.S., and includes source code for the support package on diskette and a comprehensive Users Manual, as well as the compiler itself.

Mr. Cain did a remarkable job on the original Small-C compiler, and it is encouraging to see people like Mr. Woods continuing to work on the language.

Small-C is an excellent language for a variety of systems-type applications, and with a little luck and a lot of work, its use should become widespread throughout the hobbyist-computer world.

Sincerely,
Mike Gore,
Bernie Roehl
#301-137 University Ave. W.
Waterloo, ON N2L 3E6
Canada

## "Waiter, there's a . . ."

Dear Sirs,

When I responded to your subscription offer I intended to get the trial issue and then cancel. I haven't enough time to read the three magazines I already received, so I felt I could ill afford another. But, like a fly in the soup, I am caught. Your high editorial standards are to blame; it is a joy to see that one can discuss such a technical subject without lapsing into semi-literacy or a forest of jargon.

I also enjoy *DDJ* for its content. I found the Pythagorean approximation (Dr. Dobb's Clinic, May '81) very clever. The purists might be interested that, at the expense of an additional four shifts and a subtraction, its accuracy can be nearly doubled. Thus,

dapprox=(1 − 1/16) max (dh,dv) + min dh,dv)/2

has a minimum error of 6.3%. And

dapprox=0.96 max (dh,dv) + 0.4 min (dh,dv)

is accurate to within 4%, which is probably good enough to use in coordinate rotations for video graphics.

Sincerely,
Robert S. Stephenson
244 Littleton St., Apt. 3
W. Lafayette, IN 47906

## Robust Programs?

Dear Sir,

Please find enclosed a banker's draft for $88.00 for a 2 year's subscription to *Dr. Dobb's Journal.*

As an aside I would like to see the following change in content in *DDJ.* Since the advent of good portable operating systems like the UCSD p-system, in the wake of CP/M, microcomputer users, like myself, need more high level language tools to help us develop good software. I feel that Dr. Dobbs is losing ground in this respect. The quality of the articles carried is excellent; but I think the days

## Small - C Bug Fix
## by Mike Gore and Bernie Roehl

```
/*
 *      Declare argument TYPEs
 *
 * Called from newfunc, this routine modifies the symbol
 * table entry created in newfunc() to agree with the
 * actual order of declared arguments.
 *              Mike Gore  - March 1981
 */

getarg( t, top)     /* t : CCHAR or CINT */
int t, top;         /* top : stack offset */
{
    char n[ NAMESIZE ],  *argptr;
    int j, locaddr;

    /* top tells how many arguments newfunc found
     * while scaning arguments. From this we can
     * calculate the actual stack offset and stuff
     * it back into the symbol table.
     */
    while( 1 )
    {
         if( match( "*" ) )
             j = POINTER;
        else
             j = VARIABLE;
        if(symname( n ) ) {
             /* if it's a pointer skip stuff between [] */

            if( match( "[" ) ) {
               while( inbyte( ) != ']' )
                   if( endst( ) )
                       break;
               j = POINTER;         /* mark as POINTER */
            } /* end if "[" */

                 /* if non zero we found the entry ! */

            if (argptr = findloc (n)) {

               /* stuff away type info that we just found */

                argptr[IDENT] = j;
                argptr[TYPE]  = t;

             /* reverse stacking offset */
             /* the runtime lib. does not sign extend ! */

                locaddr = top - (argptr[OFFSET + 1] << 8)
                        - argptr[OFFSET];

             /* replace with reversed offset */

                argptr[OFFSET] = locaddr;
                argptr[OFFSET + 1] = locaddr >> 8;
            } /* end if findloc */
            else
               error(" in getarg: expecting argument name");
        } /* end if symname */
        else
             illname();
        argstk = argstk - 2;
        if( endst( ) )
             return;
        if( match( "," ) == 0 )
             error("in getarg: failed to find comma");
```

*(Listing continued on page 6)*

## Small - C Bug Fix *(Continued)*
## By Mike Gore and Bernie Roehl

```
    } /* end while 1 */
} /* end of getarg() */
/*
 *     Begin a FUNCTION
 *
 * Called from parse, this routine tries to make a FUNCTION
 * out of what follows.
 */
newfunc( )
{
    char n[ NAMESIZE ], *ptr;
    int argtop;
    if( symname( n ) == 0 )
    {
        error("in newfunc: illegal function name or declaration");
        kill( );                /* invalidate line */
        return;
    }

    /* See if the function name is in the symbol table */

     if( ptr = findglb( n ) ) {

    /* If the ident field isn't a function it's a variable;
       thus the function name is multidefined. */

     if( ptr[IDENT] != FUNCTION ) {
            multidef( n );
        }
        else {

    /* If we have marked the offset field as a function,
       we have a multidefined function name */

     if( ptr[OFFSET] == FUNCTION )
                  multidef( n );
              else

    /* Now we mark the offset field we just tested as a
       function, since the name is not multidefined. The
       compiler may see a reference to a function before
       its definition; this is the reason for the
       apparent redundancy in the code used above. */

                  ptr[OFFSET] = FUNCTION;
        }
    }
    else     /* if not in table, define as FUNCTION now */
       addglb( n, FUNCTION, CINT, FUNCTION);

       /* we had better see open paren for args */

    if (match( "(" ) == 0)
        error("in newfunc: missing opening parenthesis");
    global( n );            /* output FUNCTION NAME */
    nl( );

       /* clear local stack ptr */

    locptr = STARTLOC;

       /* reset argument count */
    argstk = 0;

       /* count arguments between () */

    while( match( ")" ) == 0 ) {
```

*(Listing continued on page 8)*

of long Assembly language programs are limited. They lack the robustness and maintainability needed, whereas a good high level language (e.g. Pascal) can yield these features.

Nevertheless keep up the good work. I am particularly enamoured with the new format. Still great value for the money.

Yours faithfully,
Mark Dawson
58, Newbridge Hill,
Bath,
Avon BAI 3PU, England

*Dear Mark,*

*Thanks for your subscription, and even more thanks for your encouraging words. We try to keep DDJ oriented toward systems programming. That is, we aim it at the tool-makers rather than the tool-users. Assembly language is (unfortunately) still the tool-maker's main tool. Although Pascal and C can often replace it, our contributors haven't always learned that. We can only print the articles we receive.*

*We'd be delighted to receive contributions from you or anyone else, demonstrating solutions to systems-type problems in Pascal or C.*

*–The Editors*

## Errata

*A DDJ #55 software review mistakenly spoke of "Scribe IX." The correct, copyrighted name is "Scribewriter IX," available from Friedman Ads/Promotions.*

## About the Motorola 6809

This note is an initial reaction to Lance Leventhal's new book, *6809 Assembly Language Programming* (OSBORNE/McGraw-Hill, Berkeley, 1981). He sent me a complimentary copy, without explaining why. My guess is that he felt I didn't know enough about this microprocessor – a correct inference from my inexact, casual references to it in *DDJ* #49. I shall not here try to review the book, other than to say that it strikes me as excellent (and ought to have been published sooner!). My intent is to correct earlier comments on the 6809 multi-byte opcodes.

The 6809 does indeed have 2-byte (and even 3-byte) opcodes, and 2 "leading-byte" codes ($10 and $11). However – unlike the Z80 – the leading-byte mode is not the major enhancement and

**Small - C Bug Fix** *(Continued)*
**by Mike Gore and Bernie Roehl**

```
    /* any legal NAME bumps argstk (keeps track of arg count) */

    if( symname( n ) ) {
        if ( findloc(n))
            multidef(n) ;
        else {

            /* add local symbol to be processed */
            /* later by getarg()                */

            addloc(n, 0, 0, argstk);
            argstk = argstk + 2;
        } /* end else if findloc */
    } /* end if symname */
    else {
         error("in newfunc: illegal argument name");
         junk( );
    } /* end else if symname */
    blanks( );

    /* if not closing paren, should be comma */

    if( streq( &line[lptr], ")" ) == 0 ) {
        if( match( "," ) == 0 )
            error("in newfunc: expected comma in parameter-list");
    } /* end if streq ... */
    if( endst( ) )
        break;
} /* end while ")" */

/* Set argtop to the current offset. We will
   pass this to getarg() so the the stacking offsets
   can be reversed */

argtop = argstk;
while( argstk) {

/* Now let user declare what TYPEs those arguments
       were */

    if( amatch( "char", 4 ) ) {
        getarg( CCHAR, argtop);
        ns();
        continue;
    } /* end if "char" */
    if( amatch( "int", 3 ) ) {
        getarg( CINT, argtop);
        ns();
        continue;
    } /* end if "int" */
error("in newfunc: wrong type or number of arguments");
break;
} /* end while argstk */
stkptr = 0;             /* preset stack POINTER */

/* Do a statement but if it is a return, skip
   and clean up the stack */

if( statement( ) != STRETURN )

    {
        modstk( 0 );
        ret( );
    }
    stkptr = 0;                 /* reset stack ptr again */
    locptr = STARTLOC;          /* deallocate all locals */
}
```

**end**

does not completely redefine the following opcode. The $10 changes 15 of the 16 normal program-counter-relative branch codes from "short" (one offset byte, range −128 to +127) to "long" (two offset bytes, range −32768 to +32767.) Only the LBRA (unconditional long branch) instruction was considered important enough to be assigned a one-byte code, since it is likely to replace the JMP in most programs.

The $11 redefines the on-chip 16-bit register to be used by the following opcode (e.g., the one-byte code $FE specifies stack-register U, while $11FE alters this to stack-register S.) Both leading-bytes are also used to create 2 new software-interrupt instructions from the one-byte $3F interrupt code. All this is useful, but the expectation is that the leading-byte mode will not be used very frequently, whereas nearly all of the Z80 enhancements rely on leading-bytes.

In fact, the Z80 influence is negligible, though there is a new 16-bit Y index-register (with some coding less efficient than the otherwise similar X, because of the shortage of one-byte codes and perhaps the desire to keep some free for later upgrading). The influence I now see (I hope not again mistakenly!) is more that of the Signetics 2650, that extends its opcodes by using the 3 high-order bits of the 1-byte address to specify the addressing mode. The 6809 "opcode-extension" is a full 8-bit "post-byte" that defines a great variety of addressing modes. The extension is required by 56 opcodes (8 of which also have a leading-byte) and allows 14 major addressing variants. Nine of these require no additional addressing information. Two require one additional byte that specifies an 8-bit offset, two require two additional bytes (16-bit offset), and one has two additional bytes that are the actual address of memory locations where the "true" address has been stored ("indirect" addressing.) These effectively 2- or 3-byte opcodes are the most spectacular, complex enhancements created for the 6809. There are other remarkable ones, defined by one-byte opcodes, but those who want details must read the Leventhal book. The 6809 is too complex to be described even in a long article!

*A few thoughts on the impact of the 6809.* Only the TRS-80 Model III has so far adopted it (again proving the acumen of Radio Shack executives) though other mass-market systems may. Even

Synertek – long dedicated to the 6502 – has designed a new 6809-based SYM SBC. Although the brand-new Osborne-1 system is Z80-based, the only possible rationale for that is the existence of much complex, tested Z80 software; my guess is that the Osborne-2 will bypass the 6809 level and upgrade either to the 68000 or the coming 32-bit Intel processor (again depending on the availability of software). No older processor can compete in performance with the 6809. Its problems are lack of software, reluctance of owners of working systems to upgrade it (partly because of the software cost), and the possibility that more powerful processors may attain a superior price/performance ratio.

It must be the difficulty of equalling the performance of the 6809 that has already aborted the projected 6516 upgrade of the 6502 (and no doubt other upgrades or new designs). This leaves the field wide-open to Japanese designers, who are unwisely scorned by some notable Americans. American companies may be unwilling to gamble, compete with Motorola (at this level), or learn how to handle designers of genius. The Japanese may.

My personal reaction – significant only to the extent that it may prove typical of 6502 *aficionados* – is that there's no need for a rush move to the 6809. It should be 6800 system users who adopt it first, do the spadework, and prove its programming speed, efficiency, and elegance. It will surely excel, but by what margin? Even 6800 users have not stampeded to its upgrade, proving that the machine-code-compatibility (with the older design) played a major role in the quick adoption of the Z80 – however grotesque it may seem to a logical mind. The Motorola 6809 assembler can "translate" 6800 assembly-language into the 6809 equivalents, and similar translators could be written for other micro assembly-languages. However, this is not likely to yield optimal code, using the full power of the 6809. Only the human mind can do that. Although I do not admire the 6800 set, regret that so much of it was retained, and (as a first impression) feel that not all the 6809 innovations are ideal, its overall superiority is so great that I intend to get the SYM version.

H. T. Gordon
College of Natural Resources
U. C. Berkeley, Berkeley, CA 94702

## More on N-Logs

Dear Editor,

The comments of Mr. Mikes in *DDJ* #52 regarding my article on N-Logs were greatly appreciated. His criticisms fall into two groups, the first relating to history, the second having to do with the meaning of words and symbols. I will try to respond.

His remarks as to the origin of 'e', the base of what are now called 'Natural' or 'Naperian' logarithms are quite correct. I was deceived by the all-too-common

*(Continued on page 22)*

Our high-quality editorial
and
your high-quality products
are made for each other!
Call (415) 323-3111 to reserve your space
in the next *Dr. Dobb's Journal*.

# Pidgin—A Systems Programming Language

*DDJ is pleased to present the first of two articles by Dr. William Gale (the second article will appear next month). In these articles Dr. Gale describes three unusual pieces of software: Tincmp, a clever macro processor; Pidgin, a systems programming language; and Meta4, a compiler-compiler.*

*Dr. Gale calls Pidgin a "low-level language." This is only justified by its small number of data types (bytes, words, and vectors of bytes or words). Pidgin's assortment of control statements is as good as that of many high-level languages. Despite this, Pidgin is simple enough that it can be fully implemented as a set of Tincmp macros.*

*Tincmp, the macro processor, is the key to the series. Dr. Gale illustrates its power by using it as a translator from Pidgin source code to 6502 machine language. We're sure that isn't the end of its uses. Tincmp is given here as a program written in Pidgin. The bootstrap procedure is to paraphrase that listing in BASIC or Pascal, feed the Pidgin macros and the Tincmp source program to that program, and receive a machine-language version of Tincmp as output.*

*Next month DDJ will publish Dr. Gale's description of Meta4, a compiler-compiler ( a program that takes a formal language definition as input and delivers a compiler for that language as output). Meta4 is a development of Meta II, described in DDJ #44. The source for Meta4 will be given in Pidgin. Those readers who've brought up Tincmp by then will be able to implement Meta4 at once.*

*We hope DDJ's readers will be as excited by Tincmp, Pidgin, and Meta4 as we are. DDJ welcomes contributions based on Dr. Gale's articles. Some possibilities: a listing of Tincmp in BASIC or Pascal, or Tincmp macros to translate Pidgin to another machine language or to another programming language, or reports of your application of any of the three programs to other problems.*

Pidgin is a low-level, machine-independent programming language. Because it is easily compiled, it is suitable for systems programming applications where portability is the foremost requirement. The original design purpose was cross-time portability, so work done on my current machine could be cumulative for my next machine. But it should also be suitable for describing to a Z80 what I've done on my 6502. People with a homebrew system that nobody else writes software for may be particularly interested in a pool of machine-independent software.

Pidgin can be compiled by a macro processor that recognizes nine paramenters each of length one. Such a macro processor is easy to write, the more difficult part being to write the macros for the 80 statements of Pidgin. A macro processor written in Pidgin is included with this article. It takes 500 lines with 8000 characters including comments, and compiles into 3.5 kilobytes (with another 5 kilobytes for macro storage).

Next month another Pidgin program will be described, the Meta4 compiler generator. A compiler generator is a program that makes it easier to write a compiler. Meta4 will thus allow defining more powerful languages than Pidgin, with a machine-independent compiler. It is my hope in presenting Pidgin and these programs that some of you Z80 and 6800 users will write and publish Pidgin programs that I can use on my 6502. Microcomputers are software limited, and the bigger the pool of users and writers, the bigger our pool of software will be.

*© 1981 by William A. Gale. All rights reserved.*

**by William A. Gale**

## Brief Description of Pidgin

The 80 statements of Pidgin can be seen from the templates in either the first or second macro sets included. The discussion of Tincmp, later in this article, will clarify how these macros are to be read. The templates start with six declaration statements. The declarations set aside storage locations for each of the two data types and the four combinations of vector types. For the vectors, the declarations give the length of storage to reserve. Indexing is from zero, so the maximum index is one less than the reserved spaces. BYTE variables allow indexing BYTE vectors from 0 to 255, or INT vectors from 0 to 127. INT variables allow indexing at least from 0 to 32000.

The macros continue with fifteen data transfer statements. The transfer statements give the means for storing and retrieving in each of the four vector types. They also give the means for translating from BYTE to INT and vice versa. The BYTE translation of an INT is the low order bits of the INT. The transfer statements also include statements for setting BYTE and INT variables to a constant number, and for setting a BYTE variable to a character constant. (The statement XX= '2' will set XX to the constant 50 if ASCII is used internally.) Two other transfers got added late, and are at the end of the macros. PACK and UNPACK are abbreviations that are rather ugly, but I finally decided I didn't want to be without them when they are so easy to do for a micro. UNPACK moves the lowest byte in the named INT to the third argument, and the second lowest byte to the second argument. PACK copies the third argument into the lowest byte of the named INT and the second argument into the next lowest byte. If the INT is larger than 2 bytes, the remainder is zero filled.

Following the bulk of the transfer statements are the eleven arithmetic statements. The arithmetic statements include the usual four binary operations for integer variables, and addition and subtraction for BYTE variables. They also include increment and decrement operations for both types. Addition and subtraction are modulo some constant, so that (0−1) + 1=0.

There follow eleven statements to generate and manipulate logical values for byte variables. The logical statements include eight comparisons and three Boolean algebra statements. The eight comparisons each set a BYTE variable after comparing two variables of the same type for equality (==), inequality (!=), strictly less (<!), and less or equal (<=). The BYTE variable is set to zero if the relation is false. If it is true, the byte will be set with at least one particular bit non-zero. (This allows the boolean operations to be done on bytes defined by comparisons.) The three Boolean operations form the Boolean and (&), or (?), and not (!) for BYTE variables. For INT variables 0−1 <0, while for BYTE variables 0 < 0 − 1. That is, under comparison, INT acts as if it holds positive and negative numbers for some range, while BYTE acts as if it holds only positive numbers.

The following 25 statements are control statements of various types. I will discuss them in groups that are not in the same order as the macros.

The assembler control statements will be highly machine dependent, and should occur first in the program. Then they can be changed easily for a different machine. They allow definition of where to assemble the program (LOMEM), and where to keep the variables (REGISTER and HIMEM). If you need something more here, don't hesitate to add it. TOP can also be used to set up assembler conditions.

The program control statements are the most varied. The main program is marked off by BEGINMAIN and ENDMAIN. Subroutines are marked by SUB $$ and ENDSUB. On encountering a GOSUB $$ statement, control passes to the statement following the corresponding SUB $$ statement. Here the two characters can be any two alphanumeric symbols. On finding a RETURN or ENDSUB, control passes to the statement following the calling GOSUB $$ statement. When a GOTO $$ is encountered, control passes to the corresponding LOC $$ statement. Only decimal digits are legal for the parameters in LOC and GOTO pairs. This restriction to 100 labels is not a problem because of the remaining control statements.

The IF $$ statement specifies a BYTE variable with the two parameter characters. If the named BYTE variable is nonzero, control continues with the next following statement. Otherwise control passes to the statement beyond the next unmatched ELSE or ENDIF. Unmatched means that, for instance, between an IF and ELSE, you can put other complete sets of IF– (ELSE) –ENDIF statements.

The ON $$ statement also specifies a BYTE variable. If the named variable is nonzero, execution continues with the following statement. Otherwise, control passes to the statement beyond the next unmatched ENDWHILE statement. When an ENDWHILE is encountered, control passes to the first previous unmatched WHILE. So WHILE-ON-ENDWHILE is the form for all loops in Pidgin.

The CHOOSE ON sequence is a choice among variables, not among constants. The CHOOSE ON $$ statement specifies a BYTE variable, as does the CASE $$ statement. When a CHOOSE ON statement is encountered, control passes to the next CASE statement for which the two variables are equal, or else to the DEFAULT if there is one, or the statement following the ENDCHOOSE if there is no DEFAULT. After matching a CASE variable, when a CASE, DEFAULT, or ENDCHOOSE statement is encountered, control passes to the statement following the next ENDCHOOSE. (In the above specification, words specifying the nesting of CHOOSE's are omitted. I think that makes the explanation clearer. But the CHOOSE, like IF and WHILE can be nested.)

STOP $ terminates execution of the program, and makes the digit specified available to any supervisory program. A zero is interpreted as a normal end, any other digit is an error condition.

Two specific control statements are designed to let Pidgin programs speak to another one. BEGINMAIN specifies a BYTE variable AC and a BYTE indexed INT vector, IAV. AC (Argument Count) can be used to give the number of entries to be found in IAV. The key point is that these are special locations, known to all Pidgin programs. They are particularly useful for passing file references, or program addresses between programs. CALL I$$ is a kluge that will be recognized by BASIC users. It will make a subroutine call to the absolute location stored in the INT variable named.

The final group of statements has the input-output statements. Pidgin is designed so that input and output is usually character by character. WRITE $$ will output the named byte to the terminal. READ $$ will get one character from the terminal and set the named BYTE variable equal to it. The common exception to single-character input and output is MS, which will output the nine specified characters to the terminal. The single quote character (') should not occur among the parameters of MS. The remaining input-output statements deal with files.

The first statement to use is OPEN $$ FOR $$ AT I$$. The first two parameters specify an INT indexed BYTE vector to be used for a buffer. The third and fourth parameters specify a BYTE variable that contains either 'R', for read, or 'W' for write. The fifth and sixth parameters name an INT variable that gives a "block number". The specification of a buffer, and allocation of space means that any number of files can be open, so long as one buffer per file is allocated. The compiler only uses two buffers. The operations recognized by Pidgin are limited to read and write, but there are useful extensions (append, read or write). The block number is the key to a rudimentary file system as described in *DDJ* #56. This simple file system will allow programs to be written in Pidgin that give named files and other nice features. In this way a rather machine-independent operating system is possible. The block number is a number from one through some maximum that specifies a unique external place to put the data contents of one buffer. This avoids dependence on a particular machine by pushing the translation from block number to drive, track, sector, or what have you, down to assembly-language support programs. The translation is responsible for setting a reasonable sequence for speed of access. But all that is required is a unique map between disk sectors (or whatever) and block number.

The file input-output statements all return an error code in the specially designated BYTE variable ER. Error code zero indicates all normal; one indicates end of file or end of medium; two indicates illegal operation; and three indicates all other problems.

OPEN returns error code two if the buffer was already open, zero otherwise. This indicates that the "buffer" will contain at least some information beyond the file contents. (The Apple requires a 17-byte table to control disk input-output, which is included in the space allocated for the buffer, for instance.) For this reason, the declared lengths of the buffers will be machine dependent.

CLOSE $$ specifies a buffer. It returns ER=2 if the buffer was already closed. It marks the buffer closed. If the buffer was open to write, then CLOSE will flush the buffer, writing the last block of data, and making any closed file marks.

READ $$ FROM $$ specifies a BYTE variable with the first two parameters and a buffer with the last two. This statement reads one character from the specified buffer until it has read all of the characters in the buffer. Then it refills the buffer by getting the next sequential block number of data from the external medium. When end of file is reached, it returns both ER=1 and the character 0–1.

WRITE $$ INTO $$ is the converse. It will write one specified character into the specified buffer until the buffer is full. It then writes the buffer onto the external medium and prepares to receive more data from the program. It keeps track of the block number to write on next, writing on se-

quentially incremented blocks. It returns ER=1 if the block number is greater than the maximum allowed, or zero.

READBUF $$ and WRITEBUF $$ give an access to the low-level programs that read and write one buffer of data from the external medium. They will be useful for programs that implement directory files or random-access files, but they are not used in the compiler. Programming the subroutines to support these file input-output statements took a substantial fraction of the time to make the compiler, perhaps a third.

The last "statement" in the macros is the null macro. It allows any line started with the end-line symbol to be ignored; that is, it allows line-long comments.

## Implementing Tincmp, a compiler for Pidgin

In hopes of getting back a return of portable programs, I offer a compiler to translate Pidgin to 6502. The compiler is in two parts, a macro processor (Tincmp) written in Pidgin, and two sets of macros. I term these tools a compiler because they can generate machine code from Pidgin language. Others might feel that with no more error checking than it has, it couldn't be so dignified.

The value of the Pidgin listing of the Tincmp macro processor when you don't yet have a Pidgin compiler is this: you can easily translate Pidgin to your local Basic. If you translate Tincmp to Basic, then you can use the macros provided to form a compiler from Pidgin to 6502 in Basic. Feed that compiler the Pidgin code for Tincmp and you will have a compiler from Pidgin to 6502 in 6502. It runs much faster. This is a reasonable way to get started because the macros are the hard part, and can be used with two versions of the macro processor. I have written several versions of Tincmp, and find it takes a few days to get the macro processor working. The macros on the other hand took several weekends. When I add macros, I find I can add about 20 in a weekend.

So to implement this Pidgin compiler you need to translate Tincmp to some language you now have. To help in that let me first explain what Tincmp does, then how it does it.

Tincmp has two inputs – a set of *macros,* and an input text. Each macro consists of a *template* and a *replacement.* Templates start with a special *begin-template* character and end with the special *end-line* character. Between these two characters there can be up to nine *parameter flags* and any number of other characters besides these three or newline. The end-line character can be followed by any characters at all, then a newline. All the characters after the end-line are ignored, so they can and should be used for comments.

To summarize, a template consists of
1. begin-template character
2. zero or more ordinary characters or zero through nine parameter flags in any order
3. end-line character
4. zero or more comment characters
5. newline

If the end-line character is omitted, it will be assumed at the first newline. The macros attached show many examples using ':' as the begin-template character, ';' as the end-line character and '$' as the parameter flag.

The replacement consists of zero or more lines. Each replacement line may have
1. zero or more ordinary characters or "operation codes"
2. an optional end-line character followed by zero or more comment characters
3. newline

The power of Tincmp lies in its operation codes. It can be thought of an as interpreter for a very simple machine language. The simplicity is that there are no branching or repetition codes. The operation codes basically move data between ten registers, a stack, and the output.

The first line of the macros gives copies of the changeable special characters. If the first character is 'X', no newlines will be put into the output. This is required to generate code at times, and text at other times. The second character is a copy of the begin-template character. The third character is a copy of the end-line character. The fourth character is a copy of the parameter flag. The fifth is a copy of the operation code character. The sixth is an "ignore" character (set to tab, but invisible in the listings). The ignore character is ignored wherever it occurs in the macros. Its purpose is to help make the macros more readable. The first line must consist of exactly six characters and a newline. Otherwise an error is noted, and processing stops. With all these special characters, a way around them when necessary is available. The character '@' is a (fixed) escape character. Any one character (including newline and @) following the @ will be accepted as part of the macros. You will see it used heavily in the second macro set, because the definitions of the labels created might be special characters, just by chance.

An important link between the template and the operation codes takes place while Tincmp is matching the templates

**Table 1**

**Fetch Operations**

| | |
|---|---|
| **P Parameter** | Fetch the value of the parameter specified by the index code. |
| **V conVert** | Fetch the value of the parameter specified by the index code. Subtract the character code for zero. If the result is not between zero and nine, set it to zero. |
| **L Literal** | Fetch the index code as a character. |
| **N Number** | Fetch the index code converted as a digit. |
| **! pop** | Fetch the top of the stack, decrementing the stack pointer. |
| **S Stack** | Fetch the top of the stack. |
| **U Unique** | Fetch a unique number. The numbers start at 100, and increase sequentially. (Default). |
| **H Hex** | Fetch the index code and the dispose code, interpreted as hexadecimal digits. Set the dispose code to 'C'. |
| **T Trace** | Set the trace mode on for the remainder of the macro. Skip the dispose section. |

against input text. A line of input text *matches* a template if it is the same length as the template and if each ordinary character in the template is matched exactly. Any single character matches a parameter flag AND the character matched by the n'th parameter flag is placed in the n'th register, n=1, . . .,9. Thus the operation codes can manipulate the parameters found.

When a template has been matched, ordinary characters in the replacement are output without change. The end-line symbol causes the output of a newline. Any comments and the newline are ignored. Each operation code consists of four characters, the operation code flag, the fetch code, the index code, and the dispose code. Table 1 gives the fetch codes and Table 2 gives the dispose codes. The index code may modify the fetch code or the dispose code (or both). Let me comment on them briefly.

The value placed in the register is the internal machine representation of the character matched. The P fetch gets this value. If some other number has been stored in a register since the matching, of course P will now get that number. The registers hold integer numbers. If the parameter should be a digit, the V fetch gets a number from zero through nine. Notice that any non-digit is coerced to zero. The register is selected for these two fetches by the index code, and register zero can be selected. Since it cannot be disturbed by parameters, it can be used for a counter. The L and N fetches get data from the index code. L gets the internal machine representation of any character while N coerces it to a digit. The number on top of the stack can be fetched either with (!) or without (S) popping the stack. The fetch code H interprets both the index code and the dispose code and the hexadecimal representation of a byte and outputs it. The fetch code U allows generating ten symbols unique to a given macro. Tincmp keeps a symbol counter which starts at 100. During a macro, the U fetch adds the index code to the current symbol counter to return a symbol. Only at the end of the macro is the symbol counter increased, and then by the maximum index used during the macro, plus one. This means that references to symbol '0' during one macro will return the same number. It is intended to be used for generating labels.

Tincmp requires three parameters to be provided in IAV, the global vector mentioned above in describing BEGINMAIN. These give the starting blocks for the files to hold (1) the macros, (2) the input text, (3) the output text.

That's what Tincmp does. This is how it does it: the subroutine IN reads the flag line and initializes constants and variables. The subroutine RM reads the macro file, loading an array with information. The information is stored as shown in Table 3. After reading the macros, the input is read. Each line is compared to each template in turn until a match is found, or until all templates have been checked. If no match is found, the line is written out unchanged. If a match is found, then the subroutine DM interprets the replacement text. When an operation code is found, the next three characters are picked up. A branch on the first character does the fetch and a branch on the third does the dispose.

Tincmp can be used in two functionally different ways to compile Pidgin. One way is to use it as a preprocessor to generate code for an assembler. The other way is to run it three times and let it do its own assembly. The first way may have the advantage that an assembler will check the code for missing variables, twice-defined labels, etc. It also leads to simpler macros. However, I found when I tried it with the SCII assembler I use, that it was slow to transfer the output file from Tincmp into the assembler. Also, even as small a program as Tincmp expanded into 23K bytes of standard assembler code.

**Table 2**

**Dispose Codes**

| | |
|---|---|
| **P Parameter** | Set the value of the parameter specified by the index code to the fetched. |
| **S Stack** | Increment the stack pointer and store the fetched on the stack. |
| **C Character** | Write the low byte of the fetched. |
| **H High** | Write the high byte of the fetched. |
| **N Number** | Write the fetched in decimal characters without leading zeros. (Default.) |
| **+ add** | Add the fetched to the top of the stack. |
| **– subtract** | Subtract the fetched from the top of the stack. |
| *** multiply** | Multiply the top of the stack by ten, then add the fetched. |

**Table 3**

**Storage of Macros**

```
storage of
    :T1;COMMENTS
    R1;ALL
    :T2$;ARE
    R2 PIC; IGNORED
```

| index | ILP | index | LS |
|---|---|---|---|
| 0 | 0 | 0 | T |
| 1 | 6 | 1 | 1 |
| 2 | 17 | 2 | ; |
| . . . | | 3 | R |
| | | 4 | 1 |
| | | 5 | ; |
| | | 6 | T |
| | | 7 | 2 |
| | | 8 | $ |
| | | 9 | ; |
| | | 10 | R |
| | | 11 | 2 |
| | | 12 | |
| | | 13 | P |
| | | 14 | 1 |
| | | 15 | C |
| | | 16 | ; |
| | | . . . | |

The assembler swallowed this (slowly), but a slightly larger file would have to be cut into chunks, which would be quite cumbersome. This was too bad, because once inside, it assembled in about ten seconds.

The other way requires macros that are difficult to understand, but it gets the job done faster for me. The first set of macros produces an intermediate file in which all 6502 instructions have been coded, but with all operands left on single lines following the instruction codes. The first set of macros is thus rather like machine code. The second set of macros produces a file which is a third set of macros. This set of macros is simpler than the first, because its most frequent task is just to count the program bytes. The third set of macros contains the definitions for all labels and variables. When the third set of macros is used on the intermediate file, the completed 6502 code is output. I have included macros for this method since I have found it more useful.

Using the macros and Tincmp shown here, Tincmp compiles itself in 2 minutes 10 seconds on my Apple II. This is 50, 35, and 45 seconds for the three passes. The disk is continuously on during this time, so I believe the processing time roughly balances the input/output time. When compiled, the code occupies 3.5K bytes. It uses 4.7K bytes to store the first and larger macro set, and another thousand or so bytes for buffers, variables, etc.

In making these macros I originally tried a method I've since abandoned, and about which you might profit by a warning. One of the problems with having two sets of macros is to keep the same targets in each, with compatible definitions. I thought it might be a good idea to start from a single file and produce these two sets of macros from it by using two other sets of macros. It worked, but had two problems worse than the one it solved. First, it was confusing keeping several different sets of special characters in mind. Second, I kept having to change two sets of macros anyhow, because another macro would usually bring in a new concept and therefore a new command. The macros shown here evolved from such a process, however, and have some traces of that mechanical production remaining.

In my view macros should use every byte-saving or microsecond-saving trick known to programmers. When faced with a choice of time or space, I've usually chosen the faster, more space-consuming macro. If you can code any of the Pidgin statements with a saving in either space or time and not worsen the other, I would be glad to hear about it.

## Genealogy

Tincmp's immediate parent is SIMCMP, described by W. Waite in *Implementing Software for Non-Numeric Applications*. SIMCMP is even simpler than Tincmp, having in essence exactly three kind of operation codes (UnN, PnC, VnN). With these much simpler codes, Waite defines the First Language Under Bootstrap (FLUB), a simpler language than Pidgin, having about 30 statements. I liked the approach of SIMCMP very much, but FLUB was very hard to write, and it seemed that I would need to write a variety of programs in the first language before I could move on to the second. (Like an editor, and some kind of executive.) Pidgin has proved convenient enough that I haven't yet felt like I had to get a better language going, but maybe soon.

## Availability in Machine-Readable Form

The program and macros published here and last month give you a long start in implementing Pidgin on your system. If you want to save yourself some typing, and can transfer files from an Apple near you, you may be interested in the following offer. An experimental operating system written in Pidgin is available from the author for software experimenters. It includes Pidgin and 6502 versions of Tincmp, an editor, a tiny executive, some utilities, and Pidgin test programs. It can now be run on Apple, and can be configured for 1 through 4 floppy disk drives, using 13- or 16-sector disks. It requires a 16K memory board or an Apple II (not plus). Two disks and documentation are available for $20. A third disk with Meta4, the compiler generator to be described next month, is available for an additional $10. If you are reading this after 1981, better write to check availability. You can write me at 439 S. Orange Ave., S. Orange, NJ 07079. ■■

LISTING ON PAGE 24





**Back Issues**

**Available for $3.50**

*IBM PC, it's still hot news*

**CP/M-86 vs. MSDOS: A Technical Comparison**
Our author sets the two operating systems for the IBM Personal Computer head-to-head, to see which comes out the winner!

**Word Processors for the IBM Personal Computer**
Reviewers sample some of the latest releases in this ever-popular application of computing power.

**Video Graphics for the IBM PC**
One of our popular writers delves into the details (some not clearly documented elsewhere) of high-resolution graphics. This advanced material is carefully introduced so less experienced readers won't be lost.

This and other back issues are available for $3.50 from:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**

# Time on the Horizon

by D. J. Yates

*Dr. D. J. Yates, Botany Department, University of Queensland, St. Lucia 4067, Queensland, Australia.*

Having used microcomputers heavily for over three years now, I am firmly of the opinion that they are the solutions to many problems yet to be posed. With respect to both hardware and software, their versatility is limited only by the mind of the user. The ability to interface a microcomputer to the outside world through more conventional instrumentation makes them particularly useful experimental tools. In my own teaching and research, interfaces to measuring instruments are particularly important in the study of biological and related physical processes.

Most of the instrumentation used produces potentiometric output in various millivolt ranges up to about one volt, although some digital signals are also produced. In addition, in some circumstances there is the need to control experimental conditions with analog or digital output from the microcomputer.

One of the commonest requirements of a system when measurement and control are required is for a means of timekeeping. Sampling and or control is usually on the basis of either time of day or of time elapsed since a particular event. This requires either a peripheral or inbuilt clock to which the software has access in one way or another. In some applications this may in fact be in the measuring instrument itself. Generally, however, such is not the case and the microcomputer must provide its own time base. This article describes a simple "software" clock for use in a microcomputer with the most elementary internal clock.

Peripherals are available for many of the popular microcomputers to provide them with a real-time clock and perhaps even a calendar as well. Most microcomputers include their own "clock" which is available to the programmer. This clock varies in sophistication from a simple flag which is set by hardware at a predetermined interval after being reset, to a software function which returns time of day (hours, minutes, seconds and perhaps fractions of a second) when accessed any time after initialization at power-on. This latter type of clock is clearly the most generally useful, particularly if it provides 1/10 second accuracy or better (e.g. the PET TI and TI$ facilities). It would seem that such a feature should be a basic requirement in an operating system and higher level languages. Apparently this type of 'software-accessible-time-of-day-clock' is not yet available in many microcomputers as a standard feature, although a number possess more primitive 'clocks'. The most primitive component of a clock is hardware and obviously the diversity of hardware implementations precludes simple standardization of the software clock. On the other hand, with many of the larger microcomputers being used in business applications, it would seem that such a software clock would be fairly high on a priority list of the users.

In the North Star Horizon, the clock consists of a flag which goes "high" at a specified interval after being reset. This interval is under hardware "control." That is, a header on the motherboard may be jumpered to provide one of a selection of clock intervals between 3.328 milliseconds and 3.328 x $2^{13}$ milliseconds (about 27 seconds). The state of the clock flag may be read as the second bit of the motherboard status byte. Implementation of a time function in machine code on the Horizon would not be particularly difficult and it is a little surprising that North Star have not done so, particularly considering the general excellence of the hardware and software they provide.

The task of writing a BASIC routine or function to keep track of either elapsed time or time of day is a simple matter and Listing 1 is that of a program including a subroutine which returns the time of day (since midnight) in hours, minutes and (whole) seconds and also as total elapsed seconds since initialization.

Statements 370-410 of the main program calculate the time interval to be used in calculations. It is based on the product of the primary clock interval (3.328 milliseconds) and a power of two which depends upon the value J. The value of J is equal to the pin number to which the clock is jumpered on the Clock Rate header on the Horizon motherboard. Obviously, this value is fixed once the header is wired. The input of time in line 450 allows time keeping from the time of running the program. Immediately the time is entered, the clock flag is reset by the OUT 6, 80 instruction (statement 460).

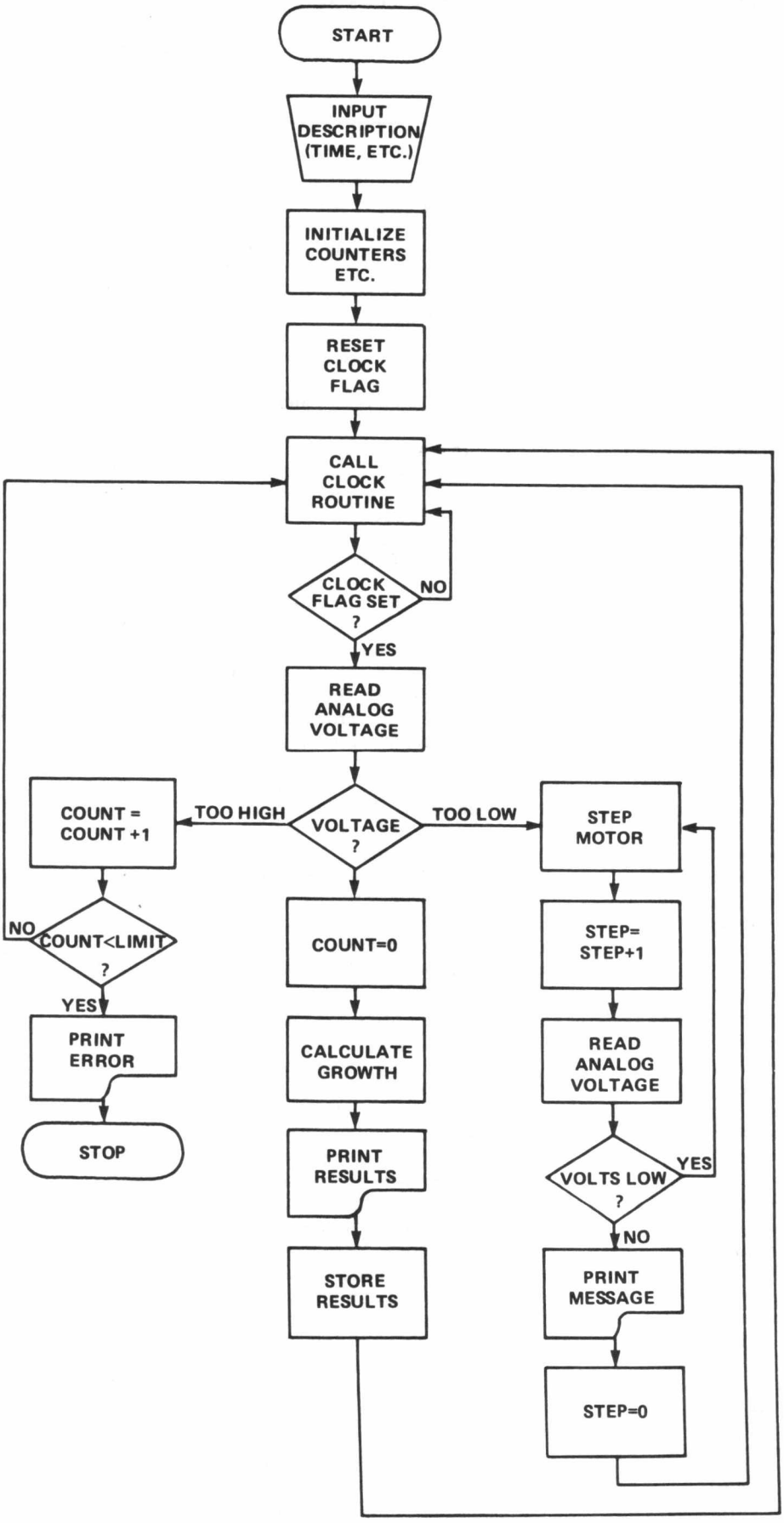
The time subroutine (statements 570-770) returns with Z9 = 0 if the clock flag is not set. If the clock flag is set then T9 is incremented by the clock interval and on the basis of T9, the time of day is calculated with seconds being rounded to the nearest whole second. On return to the main program, the time is printed. Such a program could test the value of Z9 (or H, M or S) and at the appropriate item execute relevant code. In the example, statement 510 causes a transfer back to the call to the time subroutine. Numerous variations are possible. For example, the major portion of the program may be executed only after a specified period (e.g., 10 seconds or 3 minutes) had elapsed.

In the field of biology it is often difficult to automate measurements of the various parameters which may be of interest. In the case of the measurement of the growth of grass leaves it is sometimes possible to use a linear variable displacement transformer (LVDT) to transform the vertical movement of a leaf tip as it grows into a DC voltage which may be monitored by a computer. Clearly there must be a time base if growth rate is to be calculated from changes in leaf length.

Figure 1 represents the basic flow chart of a program which uses a timing routine similar to that in Listing 1. The experimental set-up consisted of an LVDT connected by way of a beam and knife-edge pivot to a young maize leaf tip. When the leaf grew (upwards) the core of the LVDT moved down through the body of the LVFT and the resulting DC signal was monitored via an A-to-D converter by the computer. Reading of the analog signal occurred only after the clock flag had been set. Under normal conditions, growth rate was calculated and appropriate output performed before returning to the call to the clock routine.

The software and hardware included provision for handling both positive and negative over-range signals. In the case of a signal being too high (i.e. the leaf had not grown sufficiently for the signal to come on to scale) the program looped back to the call to the clock routine until the voltage had decreased to within allowable limits. If this did not occur within a reasonable time (e.g., 10 minutes), then the program would abort with appropriate output. If on the other hand the leaf grew to such an extent that the

FIGURE 1.


START
INPUT DESCRIPTION (TIME, ETC.)
INITIALIZE COUNTERS ETC.
RESET CLOCK FLAG
CALL CLOCK ROUTINE
CLOCK FLAG SET ?
NO
YES
READ ANALOG VOLTAGE
COUNT = COUNT +1
TOO HIGH
VOLTAGE ?
TOO LOW
STEP MOTOR
NO
COUNT<LIMIT ?
YES
PRINT ERROR
STOP
COUNT=0
CALCULATE GROWTH
PRINT RESULTS
STORE RESULTS
STEP= STEP+1
READ ANALOG VOLTAGE
VOLTS LOW ?
YES
NO
PRINT MESSAGE
STEP=0

other limit of the LVDT was reached, the software actuated a stepping motor attached to a micrometer head which repositioned the pivot point of the beam and in so doing, raised the core of the LVDT to the other end of its range. After re-adjustment to a preset limit, control returned to the main program.

In this application, the clock period used was about 13.63 seconds (actually 13.631488 seconds!) over which time measurable growth of a young maize leaf occurred. This period was, in fact, the period after which measurement of leaf growth was made, although measurements could be performed after some other period had elapsed merely by looping until the period had passed.

In using the approach outlined here to provide a BASIC program with a timing facility, one potential problem soon becomes apparent. If the code to be executed after a specified time always (or worse, sometimes) takes longer to execute than the clock interval, then the timekeeping ability of the software is entirely lost. This is because the software cannot know *when* the clock flag was set, but just *if* it is set; and unless it looks at the clock flag immediately after it is set (i.e., before it would be set again) it loses track of time. (This indicates too, that the loop which examines the clock flag should be fairly "tight" and should be at low-line numbers so that there is the minimum of time wasted by GOTO's.) It is essential in writing the software and, in the case of the Horizon, in hardwiring the clock header, that the longest interval between any successive "looks" at the clock flag is less than the clock interval. This of course means that the minimum time resolution is the clock period and that any shorter periods are not resolvable.

A further problem is that, at the least, a program has to explicitly call a subroutine in order to keep the clock on time. In addition, every program must re-initialize the clock and contain a subroutine such as that in Listing 1. On the other hand, there are many uses for such a simple software clock and in some situations (e.g., timing of users' response to questions) it may only require intervals to be timed and thus require no initialization other than an initial reset of the clock flag.

It may seem a relatively simple task for a user to write a machine-code routine to perform the clock functions and to keep a log of current time in (say) several spare memory locations. In the North Star DOS (and BASIC) this could be done perhaps by having the routine called from the character-in, character-out and control-C routines. Such a scheme works because the largest delays in the execution of a program occur when it is waiting for keyboard input or waiting for an output or waiting for an output device to accept output data. As the control-C routine is called after executing every BASIC statement, placing a call to a time routine within it will ensure frequent calls. For a program without disk access, this is probably satisfactory as long as individual statements do not take excessively long periods to execute (for example, calls to other user machine code routines). However, when disk activity is involved, considerable periods (e.g., from tenths of seconds to several seconds) may be involved while buffers are filled or emptied. To overcome such a problem involves placing calls to the time routine within the more primitive disk access routines of the DOS. I have not attempted to do this but the OFTEN routine which is now provided for in the North Star DOS would appear the logical place to start. Perhaps North Star Computers themselves will provide a good clock for their future software releases for the Horizon. The hardware component could conceivably even be included on the disk controller board and thus be available to all (future) North Star users. In the meantime, someone else might like to put these ideas into practice and publish them in these pages. ■■




**"The system statistics show . . . 6% accounts receivable, 23% inventory control and 71% Star Trek!"**

**LISTING ON PAGE 18**

## TIME ON THE HORIZON

```
10 REM
20 REM
30 REM
40 REM This Program and Subroutine allow real timing using the
50 REM HORIZON Real Time CLOCK and the Motherboard Status Byte.
60 REM
70 REM The program first sets the motherboard CLOCK FLAG then cycles
80 REM until it is set again. The CLOCK interval is set according to
90 REM the instructions on Page 75 of the HORIZON construction manual.
100REM
110REM
120REM            BY Dr. David Yates
130REM               Botany Department,
140REM               University of Queensland,
150REM               St Lucia, 4067
160REM               QUEENSLAND, Australia.
170REM
180REM Initially, period is 1.7039 seconds (i.e.J=5)
190REM and changes require both jumper and software  modification.
200REM  H IS HOURS
210REM  M IS MINUTES
220REM  S IS SECONDS
230REM  T9 IS TOTAL SECONDS SINCE 12 O'CLOCK
240REM
250REM  V is CLOCK interval set according to HORIZON Manual (p 75)
260REM  J is pin jumpered to pin 13 on the 14 pin DIP header at location 10A
270REM  on the HORIZON Motherboard.
280REM  R is the power of 2 used in calculating the clock interval
290REM
300REM
310 REM
320 DEF FNB(B,P)=INT(B/2^P)/2<>INT(INT(B/2^P)/2) \REM Returns Pth bit of B
330 REM
340REM
350REM
360REM
370 J=5 \REM Set pin no. ***NOTE For most applications this will be 1,2,3,4, or 5 ***
380 V=3.328 \REM Basic CLOCK interval in milliseconds
390 IF J>11 THEN 450 \REM J=12 (Cannot be 13)
400 R=14-J \REM Power of 2
410 V=V*2^R \REM Calculate V
420REM
430REM*********************************************
440REM
450 INPUT" WHAT IS TIME (H,M,S) ?  ",H,M,S
460 OUT 6,80
470 GOSUB 570 \REM CALL TIME ROUTINE
480  IF Z9=1 THEN PRINT " TIME IS  ", H," HRS",M," MINS",S," SEC"
490REM  Place program here
500REM  Place program here
510 GOTO470
520 END
530REM
540REM*********************************************
550REM
560REM
570REM Routine checks CLOCK flag and increments time if high
580REM
590REM
600 Z9=0 \REM Z9 returned as 0 if flag is low
610 IF FNB(INP(6),2)=0 THEN RETURN \REM Flag low
620 OUT 6,80
630 Z9=1 \REM Flag was high
640 T9=T9+V/1000
650 M=INT(T9/60)
660 H=INT(T9/3600)
670 M=INT((T9-H*3600)/60)
680 S=T9-H*3600-M*60
690 S=INT(S+.5)
700 RETURN
710 END
```

# BASIC Rubik's
# A Cube Simulator

by J. W. Pehoushek

*J. W. Pehoushek, 6038 Crittenden Dr., Cincinnati, OH 45244*

Magic Cubes, also known as Rubik's Cubes, have become the hottest plastic toy since the "Hula – Hoop" of the 60's. The cube, Buvos Kocka in the Hungarian of its inventor, is more than a toy, however. Mathematicians, computer scientists and artificial intelligence students the world over have taken to "The Cube" as an example of a basic mathematical problem, an exercise in group theory, and even as a parallel to particle physics. For those readers not yet smitten by "cubitis," I suggest reading Douglas Hofstadter's "Metamagical Themas" column in *Scientific American*, March 1981. Although claiming to be a mere "cubist," Hofstadter presents a thorough description of both the mechanical inner-workings of the cube and the logic fundamentals involved in any serious discussion of the subject. David Singmaster, an acclaimed "cubemeister" and professor of mathematical sciences and computing at the Polytechnic of the South Bank in London, has studied the cube thoroughly and is the author of *Notes on Rubik's Magic Cube*, available soon from Enslow Publishers of Hillside, N.J., at $5.95.

Having been introduced to the cube by my son, a Stanford computer science student, I quickly discovered the fascination with the device which has led to formation of Cubie Clubs at several universities. Further thought on the matter led me to conclude that before attacking the various theories involved in solving the cube problem, a tool for exploring moves would be convenient. A few hours with the home computer resulted in creation of the program BASIC RUBIK'S, a cube simulator. The program should run on any computer handling simple Basic statements, requiring only that you adapt the printer output routine to your particular system. Should you use a printer as the console output device, even this requirement could be eliminated, although you may wish to lay in a good supply of printer paper. You will, no doubt, suffer a severe case of at least third degree cubitis, and soon be swamped in printout.

I have chosen to follow the suggestions of Hofstadter and Singmaster in using the notation Left, Right, Front, Back, Up and Down to identify the six faces. The center facelets in each face are labeled L, R, F, B, U and D. On the actual cube these center facelets remain in a fixed position relative to each other. Similarly, in BASIC RUBIK'S the center facelets are fixed and the representation of the patterns are keyed to these positions. Since it is not possible to clearly show the position of each facelet in a single two-dimensional printed image, I've chosen to use the two views shown in Fig. 1. You may either let the computer arrange the starting position as shown in Fig. 1, or you may enter your own beginning arrangement.

I make no claims to elegance in programming, and as an engineer am more interested in the utility of the result. The basic principle of the simulator, however, may be of interest to you. At line 1590 a square matrix of size 5x5 is created for each face. The center 3x3 sub-matrix represents the actual nine facelets of any given face. The remaining border elements (row 1, row 5, column 1 and column 5) represent the edges of the face plane located on adjacent faces, ignoring the matrix corners.

Stored in each element of a face array during initialization are the ASCII numeric value of the letter for that facelet. Rotations of a face are simply accomplished by performing two sequential matrix operations, a transpose and a column switch. A transpose of a matrix is performed by moving all elements in row 1 to column 1, all elements in row 2 to column 2, etc. The column switch operation is performed by swapping column 1 with column 5, column 2 with column 4, and leaving column 3 untouched. Performing a transpose followed by column switch rotates the matrix 90 degrees CW, and a column switch followed by transpose rotates CCW. These routines are at lines 3940 and 4000, and

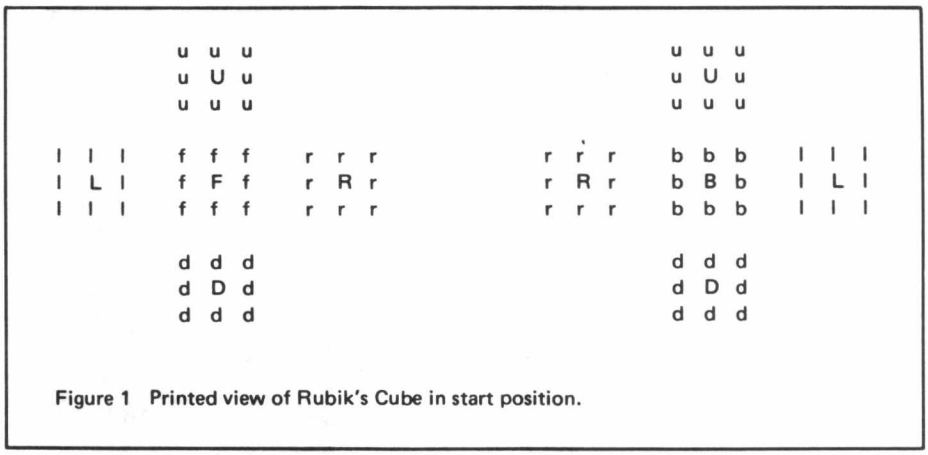



Figure 1 Printed view of Rubik's Cube in start position.

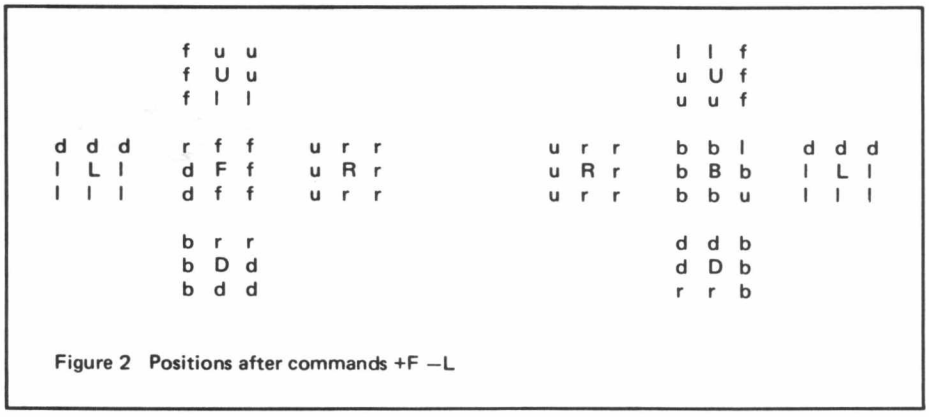



Figure 2 Positions after commands +F –L

have been combined into one FOR - - - NEXT loop for each.

The program allows users to enter commands which rotate an entire face by 90 degrees. Clockwise rotations are prefixed with a "+" sign. Commands are prompted by the phrase:

COMMAND = ?

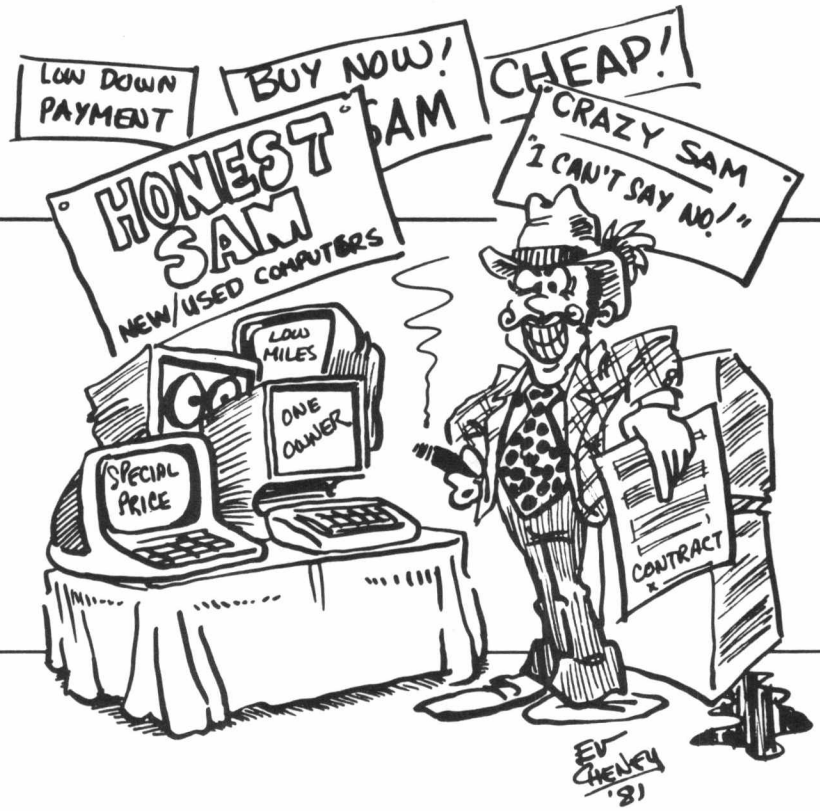
Commands must be entered one at a time, although since each command requires a second or two to execute, you may type in several commands each followed by a carriage return, if your computer has a large enough type-ahead buffer. A command "+F", followed by "–L", would result in the positions shown in Fig. 2.

The commands "V" and "P" causes display of cube positions on your video or printer device. When the "P" command is selected, a list of all previous commands executed since the last printout is also printed. Additionally, this list is printed along with a view of cube positions after every tenth command. This allows you to keep track of how you arrived at each position. You will find this feature handy as you wend your way through Rubik's intricacies. It is simply not possible to keep track of all the possible moves.

The commands "M" and "R" are used to memorize and recall arrangements from memory. At any time, when command M is entered, a snapshot of the current arrangement is saved, overwriting any previously saved arrangement. When this is done, all moves since the last printout and a view of the arrangements are printed. When command R is entered the saved arrangement is recovered and becomes the new current position. Moves since last printout are lost. This memory recall feature lets you explore moves "safely."

■■

**LISTING ON PAGE 34**




**"Well now . . . so you're thinking of buying a computer?"**

Moves since last printout:

```
            u u u                                   u u u
            u U u                                   u U u
            u u u                                   u u u

l l l       f f f     r r r             r r r       b b b     l l l
l L l       f F f     r R r             r R r       b B b     l L l
l l l       f f f     r r r             r r r       b b b     l l l

            d d d                                   d d d
            d D d                                   d D d
            d d d                                   d d d
```

Moves since last printout:
3R −L +U −D +F +F −B −B +L −D

```
            l b b                                   b b l
            u U u                                   u U u
            l b b                                   b b l

u r u       b r r     d b d             d b d       l l f     u r u
f L f       u F d     l R l             l R l       u B d     f L f
u r u       b r r     d b d             d b d       l l f     u r u

            r f f                                   f f r
            d D d                                   d D d
            r f f                                   f f r
```

Moves since last printout:
+B +F

```
            d l f                                   u f r
            u U u                                   u U u
            r f u                                   f l d

b r f       d u b     l b r             l b r       u u l     b r f
b L d       b F r     b R d             b R d       r B l     b L d
l r f       d d r     b l r             b l r       u d f     l r f

            l l d                                   b f u
            f D f                                   f D f
            u f b                                   d l l
```

The arrangement above has been saved.

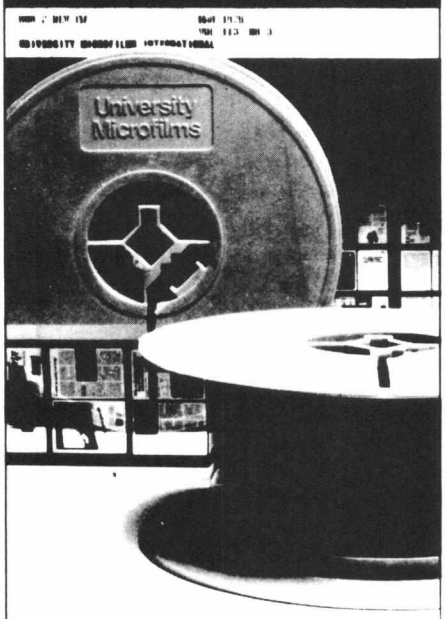
Moves since last printout:
+D −B +U −L

```
            l u u                                   b u l
            b U b                                   b U b
            l u b                                   u u l

b d f       u b d     l l r             l l r       b r u     b d f
u L d       f F r     b R l             b R l       u B f     u L d
d f d       r r f     d d f             d d f       u r r     d f d

            b f l                                   r d f
            l D l                                   l D l
            f d r                                   l f b
```

The following arrangement has been restored. Moves since last printout:

```
            d l f                                   u f r
            u U u                                   u U u
            r f u                                   f l d

b r f       d u b     l b r             l b r       u u l     b r f
b L d       b F r     b R d             b R d       r B l     b L d
l r f       d d r     b l r             b l r       u d f     l r f

            l l d                                   b f u
            f D f                                   f D f
            u f b                                   d l l
```


This publication is available in microform.




University Microfilms International

Please send additional information for _____

Name _____

Institution _____

Street _____

City _____

State _____ Zip _____

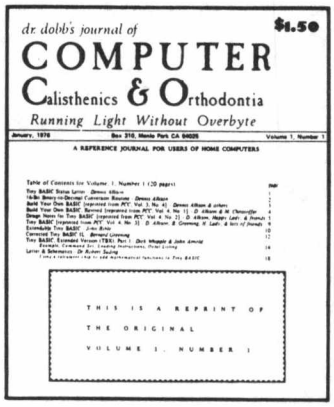
300 North Zeeb Road
Dept. P.R.
Ann Arbor, Mi. 48106
U.S.A.

30-32 Mortimer Street
Dept. P.R.
London W1N 7RA
England

Since 1976, the most innovative magazine for the technically-sophisticated microcomputer user.

# Dr. Dobb's Journal

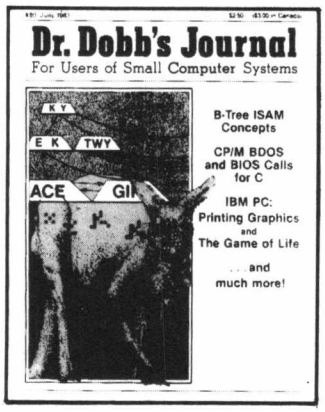
## For Users of Small Computer Systems


dr dobb's journal of
$1.50
COMPUTER
Calisthenics & Orthodontia
Running Light Without Overbyte
January, 1976 Box 310, Menlo Park CA 94025 Volume 1, Number 1
A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS
THIS IS A REPRINT OF
THE ORIGINAL
VOLUME 1, NUMBER 1

1976



1983


Dr. Dobb's Journal
For Users of Small Computer Systems
B-Tree ISAM Concepts
CP/M BDOS and BIOS Calls for C
IBM PC: Printing Graphics and The Game of Life
. . . and much more!

Have you been looking for a magazine that will provide you with serious articles on programming? *Dr. Dobb's Journal* can give you the technical depth you need to understand the latest software advances. We cover every microprocessor with articles on C, Forth, Unix, CP/M and much more. Our reputation for editorial quality is well known. Subscribe now, and discover what industry insiders have enjoyed since 1976.

Coming articles will cover:

- new language development
- 8- and 16-bit technology
- operating systems
- compilers, editors
- assemblers/cross-assemblers
- clever algorithms

**$25**

one year subscription
12 issues
saves $5 off the newsstand price

**YES!** Enter my subscription to *Dr. Dobb's Journal* and bill me later.

name

address

city state zip

**Mail to: DDJ, P.O. Box E., Menlo Park, CA 94025**

Why wait
to start
increasing your sales?
Call now
to reserve
advertising space
in the next
*Dr. Dobb's Journal*
(415) 323-3111

## Letters *(Continued from page 9)*

human custom of naming something after a man who did not invent it. There are, as far as I know, three kinds of logarithms: Naperian, Briggsian, and Gaussian. Of the three only the second bears the inventor's name.

I can't agree with his remarks about the use of e in computation. The fact that $\ln(-1) = i*\pi$ is certainly remarkable, but has nothing to do with the argument. Any compendium of mathematical series (e.g. Ref. 1) will show a number of series used for computing logarithms, and invariably the result is the Naperian variety. (That reference calls it the 'log'.) The algorithm used for computing logarithms in the HP calculators (see Ref. 2) and, I suspect, in most computer programs, is based on 'e'; in fact, I don't know that there is any other way to accomplish this computation.

To attempt to find the 'true' meaning of many technical words and symbols is a hopeless effort. It often doesn't exist! The late Dr. Percy Bridgman of Harvard is quoted as having said, "For of course the true meaning of a word is to be found by observing what a man does with it, not what he says about it." If we try to apply this principle to find the "true" meaning of abbreviations like LOG, LN, EXP and so on, we find that there is none! Everyone has his own preference! The BASIC program on my computer uses LOG to mean the natural logarithm, my HP calculator calls this LN, and uses LOG to mean "log-to-the-base-10", the TI calculator uses "lnx" and "log" respectively to mean the same things! All this confusion makes it necessary for any conscientious author who uses such abbreviations to define them in the book or article he writes. This is the only way to make sure that the reader knows the meaning that is intended.

As regards the Decibel, I am convinced that very few worthwhile concepts have ever been so completely misunderstood, misused and generally maltreated. For more than fifteen years I have been trying to find some way to help this situation. I have written a book (unpublished) and no less than ten articles (some published, see Ref. 3 and 4), N-Logs being my latest in this long history of "Windmill-Tilting." The reader would run out of patience, and the editor out of space if I attempted to list here all of the horrible sins that are committed in the use of decibels, logarithms and exponents. Perhaps a few final comments would be in order.

One usage that is particularly annoying is the use by computer engineers of the words "characteristic" and "mantissa." For several hundred years these words have been used to denote the part of a logarithm to the left, and to the right of the decimal. With all the words in the world to choose from why must we redefine them as parts of an exponential number? As if this weren't bad enough, the experts don't agree. Peter Stark says (in Ref. 5), "Each number in a computer must be divided into two parts: one part (called the mantissa) which gives the digits in the number, and a second part (called the characteristic or exponent) which provides information about the number's relative size." Rodney Zaks, in Ref. 6, referring to the number 1.0E+2 says, "1.0 is called the mantissa or characteristic, and the number following the E is called the exponent." I suggest we call the first part the multiplier, and the exponent just that, then we'll understand each other a little better, besides agreeing with Noah Webster and several thousand mathematics textbooks!

Finally I'd like to put the blame for all this confusion about logarithms right where I think it belongs – in the classroom. Almost everyone who suffered through the standard course in Trigonometry, which, for no earthly reason, always included Logarithms, finished with the confirmed conclusion that the whole mess was too complicated to bother with, and forgot about it as soon after the final exam as possible!

Kenneth A. Simons
2035 Willowbrook Drive
Huntingdon Valley, PA 19006

■ ■

**References:**

[1] B. O. Pierce, "A Short Table of Integrals", Third Ed., pp. 90-91.

[2] William E. Egbert, "Personal Calculator Algorithm IV: Logarithmic Functions", In Hewlett-Packard Journal, April 1978

[3] K. A. Simons, "The dB Anything", Proc. IEEE(Lett.), Vol. 61, pp. 495-496

[4] Comments following Chester H. Page, "Logarithmic Quantities and Units", Proc. IEEE (Lett.), Oct. 1973, pp. 1516-1517

[5] Peter A. Stark, "Computer Programming Handbook", TAB Books, No. 752, p. 319

[6] Rodnay Zaks, "Introduction to PASCAL", Sybex, Second Edition, p. 35

# Pidgin (Text on pages 10-14)

## First Macro Set

```
U:;$^
:BYTE $$;
        ^P0S^N1+^!0P
:INT I$$;
        ^P9S^N2-^!9P
:BYTE $$($$$);
        ^P9S^V3S^V4*^V5*^!0-^!9P
        ^HA9;
        ^P9C;
        ^H85^P0C;
        ^P0S^N1+^!0P
        ^HA9;
        ^P9H;
        ^H85^P0C;
        ^P0S^N1+^!0P
:INT I$$($$$);
        ^P9S
        ^V3S^V4*^V5*^S0+^!0-
        ^!9P
        ^HA9;
        ^P9C;
        ^H85^P0C;
        ^P0S^N1+^!0P
        ^HA9;
        ^P9H;
        ^H85^P0C;
        ^P0S^N1+^!0P
:BYTE $$($$$$$);
        ^P9S
        ^V3S^V4*^V5*^V6*^V7*^!0-
        ^!9P
:INT I$$($$$$$);
        ^P9S
        ^V3S^V4*^V5*^V6*^V7*^S0+^!0-
        ^!9P
:$$='$';
        ^HA9;
        ^P3C;
        ^H85;
        ^P1C^P2C;
:$$=+$$$;
        ^HA9;
        ^V3S^V4*^V5*^!0C;
        ^H85;
        ^P1C^P2C;
:I$$=+$$$$$;
        ^HA9;

        ^V3S^V4*^V5*^V6*^V7*^S0C;
        ^H8D;
        LI^P1C^P2C;
        ^HA9;
        ^!0H;
        ^H8D;
        HI^P1C^P2C;
:$$=$$;
        ^HA5;
        LI^P5C^P6C;
        ^HB9;
        L^P3C^P4C;
        ^H0A;
        ^H85;
        ^P1C^P2C;

:I$$=I$$(I$$);
        ^HAD;
        LI^P5C^P6C;
        ^H0A;
        ^HAA;
        ^HAD;
        HI^P5C^P6C;
        ^H2A;
        ^HA8;
        ^H8A;
        ^H18;
        ^H69;
        LI^P3C^P4C;
        ^H85;
        LBASE;
        ^H98;
        ^H69;
        HI^P3C^P4C;
        ^H85;
        HBASE;
        ^HA0;
        ^H01;
        ^HB1;
        LBASE;
        ^H99;
        LI^P1C^P2C;
        ^H88;
        ^HF0;
        ^N0S^N8-^!0C;

:$$($$)=$$;
        ^HA4;
        ^P3C^P4C;
        ^HA5;
        ^P5C^P6C;
        ^H91;
        ^P1C^P2C;

:I$$($$)=I$$;
        ^HA5;
        ^P3C^P4C;
        ^H0A;
        ^HA8;
        ^HAD;
        LI^P5C^P6C;
        ^H91;
        I^P1C^P2C;
        ^HC8;
        ^HAD;
        HI^P5C^P6C;
        ^H91;
        I^P1C^P2C;

        ^HE5;
        ^P5C^P6C;
        ^H85;
        ^P1C^P2C;
:$$++;
        ^HE6;
        ^P1C^P2C;
:$$--;
        ^HC6;
        ^P1C^P2C;
:I$$++;
        ^HEE;
        LI^P1C^P2C;
        ^HD0;
        ^H03;
        ^HEE;
        HI^P1C^P2C;

:I$$--;
        ^HA9;
        ^H00;
        ^HCD;
        LI^P1C^P2C;
        ^HD0;
        ^H03;
        ^HCE;
        HI^P1C^P2C;
        ^HCE;
        LI^P1C^P2C;

:I$$=-I$$;
        ^HA9;
        ^H00;
        ^H38;
        ^HED;
        LI^P3C^P4C;
        ^H8D;
        LI^P1C^P2C;
        ^HA9;
        ^H00;
        ^HED;
        HI^P3C^P4C;
        ^H8D;
        HI^P1C^P2C;

:I$$=I$$+I$$;
        ^HAD;
        LI^P3C^P4C;
        ^H18;
        ^H6D;
        LI^P5C^P6C;
        ^H8D;
        LI^P1C^P2C;
        ^HAD;
        HI^P3C^P4C;
        ^H6D;
        HI^P5C^P6C;
        ^H8D;
        HI^P1C^P2C;

        ^HA5;
        ^P3C^P4C;
        ^HC5;
        ^P5C^P6C;
        ^HF0;
        ^H02;
        ^HA2;
        ^HFF;
        ^H86;
        ^P1C^P2C;

:$$=$$==$$;
        ^HA2;
        ^HFF;
        ^HA5;
        ^P3C^P4C;
        ^HC5;
        ^P5C^P6C;
        ^HF0;
        ^H02;
        ^HA2;
        ^H00;
        ^H86;
        ^P1C^P2C;

:$$=$$<I$$;
        ^HA2;
        ^HFF;
        ^HA5;
        ^P3C^P4C;
        ^HC5;
        ^P5C^P6C;
        ^H90;
        ^H02;
        ^HA2;
        ^H00;
        ^H86;
        ^P1C^P2C;

:$$=$$<=$$;
        ^HA2;
        ^H00;
        ^HA5;
        ^P5C^P6C;
        ^HC5;
        ^P3C^P4C;
        ^H90;
        ^H02;
        ^HA2;
        ^HFF;
        ^H86;
        ^P1C^P2C;
:$$=I$$!=I$$;
        ^HA2;
        ^H00;
        ^HA0;
        ^H01;
        ^HB9;
        LI^P3C^P4C;
```

```
        ^P3C^P4C;
        ^H85;
        ^P1C^P2C;
:I$$=$$;
        ^HA5;
        ^P3C^P4C;
        ^H8D;
        LI^P1C^P2C;
        ^HA9;
        ^H00;
        ^H8D;
        HI^P1C^P2C;
:$$=I$$;
        ^HAD;
        LI^P3C^P4C;
        ^H85;
        ^P1C^P2C;
:I$$=I$$;
        ^HAD;
        LI^P3C^P4C;
        ^H8D;
        LI^P1C^P2C;
        ^HAD;
        HI^P3C^P4C;
        ^H8D;
        HI^P1C^P2C;
:$$=$$($$);
        ^HA4;
        ^P5C^P6C;
        ^HB1;
        ^P3C^P4C;
        ^H85;
        ^P1C^P2C;
:I$$=I$$($$);
        ^HA5;
        ^P5C^P6C;
        ^H0A;
        ^HA8;
        ^HB1;
        I^P3C^P4C;
        ^H8D;
        LI^P1C^P2C;
        ^HC8;
        ^HB1;
        I^P3C^P4C;
        ^H8D;
        HI^P1C^P2C;

:$$=$$(I$$);
        ^HA9;
        H^P3C^P4C;
        ^H18;
        ^H6D;
        HI^P5C^P6C;
        ^H8D;
        L^U0N;
        ^HAC;
```

*(Continued on page 24, column 2)*

```
:$$(I$$)=$$;
        ^HA9;
        H^P1C^P2C;
        ^H18;
        ^H6D;
        HI^P3C^P4C;

        ^H8D;
        L^U0N;
        ^HAC;
        LI^P3C^P4C;
        ^HA5;
        ^P5C^P6C;
        ^H99;
        L^P1C^P2C;
        ^H0A;
:I$$(I$$)=I$$;
        ^HAD;
        LI^P3C^P4C;
        ^H0A;
        ^HAA;
        ^HAD;
        HI^P3C^P4C;
        ^H2A;
        ^HA8;
        ^H8A;
        ^H18;
        ^H69;
        LI^P1C^P2C;
        ^H85;
        LBASE;
        ^H98;
        ^H69;
        HI^P1C^P2C;
        ^H85;
        HBASE;

        ^HA0;
        ^H01;
        ^HB9;
        LI^P5C^P6C;
        ^H91;
        LBASE;
        ^H88;

        ^HF0;
        ^N0S^N8-^!0C;
:$$=$$+$$;
        ^HA5;
        ^P3C^P4C;
        ^H18;
        ^H65;
        ^P5C^P6C;
        ^H85;
        ^P1C^P2C;
:$$=$$-$$;
        ^HA5;
        ^P3C^P4C;
        ^H38;
```

*(Continued on page 24, column 3)*

```
:I$$=I$$-I$$;
        ^HAD;
        LI^P3C^P4C;
        ^H38;
        ^HED;
        LI^P5C^P6C;
        ^H8D;
        LI^P1C^P2C;
        ^HAD;
        HI^P3C^P4C;
        ^HED;
        HI^P5C^P6C;
        ^H8D;
        HI^P1C^P2C;
:I$$=I$$*I$$;
        ^H20;
        IMUL;


        LI^P3C^P4C;
        LI^P5C^P6C;
        ^H8D;
        LI^P1C^P2C;
        ^H8C;
        HI^P1C^P2C;
:I$$=I$$/I$$;
        ^H20;
        IDIV;
        LI^P3C^P4C;
        LI^P5C^P6C;


        ^H8D;
        LI^P1C^P2C;
        ^H8C;
        HI^P1C^P2C;
:$$=$$&$$;
        ^HA5;
        ^P3C^P4C;
        ^H25;
        ^P5C^P6C;
        ^H85;
        ^P1C^P2C;
:$$=$$?$$;
        ^HA5;
        ^P3C^P4C;

        ^H05;
        ^P5C^P6C;
        ^H85;
        ^P1C^P2C;
:$$=!$$;
        ^HA5;
        ^P3C^P4C;
        ^H49;
        ^HFF;
        ^H85;
        ^P1C^P2C;
:$$=$$!=$$;
        ^HA2;
        ^H00;
```

*(Continued on page 24, column 4)*

```
        ^HD9;
        LI^P5C^P6C;
        ^HF0;
        ^H02;
        ^HA2;
        ^HFF;
        ^H88;
        ^H10;
        ^N0S^N1S^N3*^!0-^!0C;
        ^H86;


        ^P1C^P2C;
:$$=I$$==I$$;
        ^HA2;
        ^HFF;
        ^HA0;
        ^H01;
        ^HB9;
        LI^P3C^P4C;
        ^HD9;
        LI^P5C^P6C;
        ^HF0;
        ^H02;
        ^HA2;


        ^H00;
        ^H88;
        ^H10;
        ^N0S^N1S^N3*^!0-^!0C;
        ^H86;
        ^P1C^P2C;
:$$=I$$<!I$$;
        ^HA2;
        ^HFF;
        ^HAD;
        HI^P3C^P4C;
        ^HCD;
        HI^P5C^P6C;


        ^H4D;
        HI^P5C^P6C;
        ^HF0;
        ^H09;
        ^H10;
        ^H0D;
        ^HAD;
        HI^P5C^P6C;
        ^H10;
        ^H0A;
        ^H30;
        ^H08;
        ^HAD;
        LI^P3C^P4C;
        ^HCD;
        LI^P5C^PC6;
        ^H90;
        ^H02;
        ^HA2;
        ^H00;
```

*(Continued on next page)*

# Pidgin (Listing continued, text on pages 10-14)

```
        ^H86;
        ^P1C^P2C;
:$$=I$$<=I$$;
        ^HA2;
        ^H00;
        ^HAD;


        HI^P5C^P6C;
        ^HCD;
        HI^P3C^P4C;
        ^H4D;
        ^HI^P3C^P4C;
        ^HF0;
        ^H09;
        ^H10;
        ^H0D;
        ^HAD;
        HI^P3C^P4C;
        ^H10;
        ^H0A;
        ^H30;
        ^H08;
        ^HAD;
        LI^P5C^P6C;
        ^HCD;
        LI^P3C^P4C;
        ^H90;
        ^H02;
        ^HA2;
        ^HFF;
        ^H86;
        ^P1C^P2C;


:SUB $$;
:GOSUB $$;
        ^H20;
        S^P1C^P2C;
:CALL I$$;
        ^HAD;
        LI^P1C^P2C;
        ^H85;
        LBASE;
        ^HAD;
        HI^P1C^P2C;
        ^H85;
        HBASE;
        ^H20^H65^H96;
:RETURN;
        ^H60;
:ENDSUB;
        ^H60;
:GOTO $$;
        ^H4C;
        U^P1C^P2C;
:LOC $$;
:STOP $;
        ^HA9;
        ^P1C;
        C^S0N;
:CASE $$;
        ^!0P
        ^H4C;
        C^S0N;
        ^HA5;
        ^P1C^P2C;
        ^HC5;
        SWT;
        ^HF0;
        ^H03;
        ^U0S
        ^H4C;
        C^S0N;


:DEFAULT;
        ^!0P
        ^H4C;
        C^S0N;
        ^U0S
:ENDCHOOSE;
        ^!0P
        ^!0P
:MS '$$$$$$$$$';
        ^H20;
        IMSG;
        ^P1C^P2C^P
        ^P4C^P5C^P
        ^P7C^P8C^P9C;
:WRITE $$ INTO $$;
        ^HA9;
        L^P3C^P4C;
        ^HA0;
        H^P3C^P4C;
        ^HA6;
        ^P1C^P2C;


        ^H20;
        PTCH;
:WRITE $$;
        ^HA5;
        ^P1C^P2C;
        ^H20;
        WRIT;
:WRITEBF $$;
        ^HA9;
        L^P1C^P2C;
        ^HA0;
        H^P1C^P2C;
        ^H20;
        BFRF;
        ^H20;
        WRBF;
:READBF $$;
        ^HA0;
        L^P1C^P2C;
        ^HA0;
        H^P1C^P2C;
        ^HA5;
        ^P3C^P4C;
        ^H8D;
        HI^P1C^P2C;
:;
```

## Second Macro Set

```
U:;$^
:BYTE $$;
        )^P1C^P2C(;
        @@^P0C;
        ^P0S^N1+^!0P
:INT I$$;
        ^P9S^N2-^!9P
        )LI^P1C^P2C(;
        @@^P9C@@^P9H;
        )HI^P1C^P2C(;
        ^P9S^N1+@@^S0C@@^!0H;


:BYTE $$($$$);
        )^P1C^P2C(;
        @@^P0C;
        ^P9S
        ^V3S^V4*^V5*^!0-^!9P
        ^N4+
        ^P0S^N1+^!0P
        ^N4+
        ^P0S^N1+^!0P



:INT I$$($$$);
        )I^P1C^P2C(;
        @@^P0C;
        ^P9S
        ^V3S^V4*^V5*^S0+^!0-
        ^!9P
        ^N4+
        ^P0S^N1+^!0P
        ^N4+
        ^P0S^N1+^!0P



:BYTE $$($$$$$);
        ^P9S
        ^V3S^V4*^V5*^V6*^V7*^!0-
        ^!9P
        )L^P1C^P2C(;
        @@^P9C;
        )H^P1C^P2C(;
        @@^P9H;


:INT I$$($$$$$);
        ^P9S
        ^V3S^V4*^V5*^V6*^V7*^S0+^!0-
        ^!9P
```

```
:$$=$$+$$;
        ^N7+
:$$=$$-$$;
        ^N7+
:$$++;
        ^N2+
:$$--;
        ^N2+


:I$$++;
        ^N8+
:I$$--;
        ^N9+
        ^N4+
:I$$=-I$$;
        ^N9+
        ^N8+
:I$$=I$$+I$$;
        ^N1+
        ^N9+
        ^N9+
:I$$=I$$-I$$;
        ^N1+
        ^N9+
        ^N9+
:I$$=I$$*I$$;
        ^N9+
        ^N4+
:I$$=I$$/I$$;
        ^N9+
        ^N4+


:$$=$$&$$;
        ^N6+
:$$=$$?$$;
        ^N6+
:$$=!$$;
        ^N6+
:$$=$$!=$$;
        ^N8+
        ^N4+
:$$=$$==$$;
        ^N6+
        ^N6+
:$$=$$<!$$;
        ^N8+
        ^N4+
:$$=$$<=$$;
        ^N6+
        ^N6+
:$$=I$$!=I$$;
        ^N9+
        ^N9+
        ^N1+
:$$=I$$==I$$;
        ^N9+
        ^N9+
        ^N1+
```

```
        ^HAE;
        SPSV;
        ^H9A;
        ^H60;
:TOP;
:HIMEM=$$$$$;

        ^V1S^V2*^V3*^V4*^V5*^!9P
:LOMEM=$$$$$;
:REGISTER=$$$;
        ^V1S^V2*^V3*^!0P
:BOTTOM;
        ^HFF;
:BEGINMAIN(AC,IAV);
        ^HBA;
        ^H8E;
        SPSV;
:ENDMAIN;
        ^H60;
:IF $$;
        ^HA5;
        ^P1C^P2C;
        ^HD0;
        ^H03;
        ^U0S
        ^H4C;
        I^S0N;
:ENDIF;
        ^!0P
:ELSE;
        ^!0P
        ^U0S

        ^H4C;
        I^S0N;
:WHILE;
        ^U0S
:ON $$;
        ^HA5;
        ^P1C^P2C;
        ^HD0;
        ^H03;
        ^U0S
        ^H4C;
        W^S0N;
:ENDWHILE;
        ^!0P
        ^H4C;
        W^S0N;
        ^!0P
:CHOOSE ON $$;
        ^HA5;
        ^P1C^P2C;
        ^H85;
        SWT;
        ^U0S
        ^U1S
        ^H4C;
```

*(Continued on page 26, column 2)*

```
        ^H20;
        BFRF;
        ^H20;
        RDBF;
:READ $$ FROM $$;
        ^HA9;
        L^P3C^P4C;
        ^HA0;
        H^P3C^P4C;
        ^H20;
        GTCH;
        ^H85;
        ^P1C^P2C;
:READ $$;

        ^H20;
        READ;
        ^H85;
        ^P1C^P2C;
:OPEN $$ FOR $$ AT I$$;
        ^HAC;
        HI^P5C^P6C;
        ^HAD;
        LI^P5C^P6C;
        ^H20;
        BDTS;
        ^HA9;
        L^P1C^P2C;
        ^HA0;
        H^P1C^P2C;
        ^H20;
        BFRF;
        ^HA5;
        ^P3C^P4C;
        ^H20;
        OPEN;

:CLOSE $$;
        ^HA9;
        L^P1C^P2C;
        ^HA0;
        H^P1C^P2C;
        ^H20;
        CLOS;
:UNPACK(I$$,$$,$$);
        ^HAD;
        HI^P1C^P2C;
        ^H85;
        ^P3C^P4C;
        ^HAD;
        LI^P1C^P2C;
        ^H85;
        ^P5C^P6C;
:PACK(I$$,$$,$$);
        ^HA5;
        ^P5C^P6C;
        ^H8D;
        LI^P1C^P2C;
```

*(Continued on page 26, column 3)*

```
        )LI^P1C^P2C(;
        @@^P9C;
        )HI^P1C^P2C(;
        @@^P9H;



:$$='$';
        ^N4+
:$$=+$$$;
        ^N4+
:I$$=+$$$$$;
        ^N7+
        ^N3+
:$$=$$;
        ^N4+
:I$$=$$;
        ^N7+
        ^N3+
:$$=I$$;
        ^N5+
:I$$=I$$;
        ^N9+
        ^N3+
:$$=$$($$);
        ^N6+
:I$$=I$$($$);
        ^N9+

        ^N6+
:$$=$$(I$$);
        ^N9+
        ^N6+
        )L^U0N(;
        ^S0S^N1-@@^S0C@@^!0H;
        ^N2+
:I$$=I$$(I$$);
        ^N9+



        ^N7+
        ^N9+
        ^N6+
:$$($$)=$$;
        ^N6+
:I$$($$)=I$$;
        ^N7+
        ^N8+
:$$(I$$)=$$;
        ^N9+
        ^N8+
        )L^U0N(;
        ^S0S^N1-@@^S0C@@^!0H;
:I$$(I$$)=I$$;
        ^N9+
        ^N8+
        ^N9+
        ^N5+
```

*(Continued on page 26, column 4)*

```
:$$=I$$<!I$$;
        ^N9+
        ^N9+
        ^N9+
        ^N7+
:$$=I$$<=I$$;
        ^N9+
        ^N9+
        ^N9+
        ^N7+
:SUB $$;
        )S^P1C^P2C(;
        @@^S0C@@^S0H;


:GOSUB $$;
        ^N3+
:CALL I$$;
        ^N9+
        ^N4+

:RETURN;
        ^N1+
:ENDSUB;
        ^N1+
:GOTO $$;
        ^N3+
:LOC $$;
        )U^P1C^P2C(;
        @@^SOC@@^S0H;
:STOP $;
        ^N7+
:TOP;


        X)($XX;
        )SWT(;
        @@^H0D;
        )LBASE(;
        @@^H0E;
        )HBASE(;
        @@^H0F;
        )ER(;
        @@^H0B;
        )PAG(;
        @@^H03;
        )COL(;
        @@^H05;
        )IMSG(;
```

*(Continued on next page)*

# Pidgin (Listing continued, text on pages 10-14)

```
        @@^H00@@^H90;
        )IMUL(;
        @@^H55@@^H90;
        )IDIV(;
        @@^H2A@@^H90;
        )BDTS(;
        @@^HBF@@^H90;
        )BFRF(;
        @@^HC2@@^H90;
        )OPEN(;
        @@^HC5@@^H90;
        )CLOS(;
        @@^HC8@@^H90;
        )RDBF(;
        @@^HCB@@^H90;
        )GTCH(;
        @@^HCE@@^H90;
        )WRBF(;
        @@^HD1@@^H90;
        )PTCH(;
        @@^HD4@@^H90;
        )READ(;
        @@^HD7@@^H90;
        )WRIT(;
        @@^HDA@@^H90;


:HIMEM=$$$$$;
        ^V1S^V2*^V3*^V4*^V5*^!9P
:LOMEM=$$$$$;
        ^V1S^V2*^V3*^V4*^V5*
:REGISTER=$$$;
        ^V1S^V2*^V3*^!0P
:BOTTOM;
:BEGINMAIN(AC,IAV);
        )AC(;
        @@^H0C;
        )IAV(;
        @@^H09;
        ^P9S^N1-^!9P
        )SPSV(;
        @@^P9C@@^P9H;
        ^N4+


:ENDMAIN;
        ^N1+
:IF $$;
        ^N4+
        ^!8P^U0S^P8S
        ^N3+
:ENDIF;
        ^!9P)I^S0N(^P9S;
        @@^S0C@@^S0H;
        ^!8P^!9P^P8S
:ELSE;
        ^!8P^!9P^P8S
        ^N3+
        )C^P9N(;
        @@^S0C@@^S0H;
        ^N6+
        ^!8P^U0S^P8S
        ^N3+
:DEFAULT;
        ^!8P^!9P^P8S
        ^N3+
        )C^P9N(;
        @@^S0C@@^S0H;


        ^!8P^U0S^P8S
:ENDCHOOSE;
        ^!8P^!9P^P8S
        ^!9P)C^S0N(^P9S;
        @@^S0C@@^S0H;
        ^!8P^!9P^P8S
:MS '$$$$$$$$$';
        ^N3+
        ^N9+
:WRITE $$ INTO $$;
        ^N9+
:WRITE $$;
        ^N5+
:WRITEBF $$;
        ^N5+
        ^N5+
:READBF $$;
        ^N5+
        ^N5+
:READ $$ FROM $$;
        ^N9+
:READ $$;
        ^N5+
:OPEN $$ FOR $$ AT I$$;
        ^N3+
        ^N9+
        ^N9+
:CLOSE $$;
        ^N7+
:UNPACK(I$$,$$,$$);
        ^N5+
        ^N5+
:PACK(I$$,$$,$$);
        ^N5+
        ^N5+
:;
```

**End Macro Set Listings**

```
BYTE HF;'F'
BYTE LE;END OF LIST
BYTE LS(09000);LIST OF MACRO DEFINITIONS
BYTE MF;MACRO REPLACEMENT OPER FLAG
BYTE ML;MACRO LENGTH
BYTE MM;MINIMUM MACRO LEN

BYTE ND;NUMBER OF DIGITS- USED IN SUB SD FOR NUMBER OUTPUT
BYTE NL;NEWLINE
BYTE O1;FETCH CODE
BYTE O2;INDEX CODE
BYTE O3;DISPOSE CODE
BYTE OA;'+' ADD OPERATOR
BYTE OB;'!' POP STACK OPERATOR
BYTE OC;'C' CHARACTER DISPOSE
BYTE OD;'V' DIGIT CONVERSION FETCH
BYTE OE;ESCAPE CHARACTER
BYTE OG;IGNORE CHARACTER
BYTE OH;'H' HEX CONVERSION FETCH
BYTE OL;'L' LITERAL FETCH
BYTE OM;'*' MULTIPLY DISPOSE
BYTE ON;'N' NUMERIC LITERAL FETCH

BYTE OP;'P' PARAMETER FETCH OR DISPOSE
BYTE OR;'-' REDUCE (SUBTRACT) DISPOSE
BYTE OS;'S' STACK FETCH OR DISPOSE
BYTE OT;TRACE FLAG TURN ON
BYTE PP;POINTER INTO IPR
BYTE RB;BEGIN DEFINITION FLAG
BYTE RC;(COMMENT) END-LINE FLAG
BYTE SF;SUBSTITUTION (PARAMETER) FLAG
BYTE SP;STACK POINTER
BYTE TR;TRUE IF TO TRACE
BYTE UG;USE IGNORE; TRUE UNLESS OG IS 'X'

BYTE UN;NOT X-- FLAG FOR NOT SUPPRESSING NEWLINES ON OUTPUT
BYTE UO;USE OPERATIONS-- TRUE UNLESS MF IS 'X'
BYTE UT;USE TRACE  TRACE MODE IS ON
BYTE ZR;CHARACTER ZERO
INT I00;CONSTANT 0
INT I01;CONSTANT 1
INT I09;CONSTANT 9
INT I10;CONSTANT 10
INT I16;CONSTANT 16
INT IAA;WORK
INT IBB; WORKING STORAGE
INT IBC;BUCKET NUMBER
INT IDP;DEFINITION POINTER WHILE MATCH
INT IED;POINTS TO END OF DEFINITIONS
INT III; POINTER TO L WHILE READING
INT IJJ; POINTER TO L READING CODE
INT ILM;MAXIMUM LIMIT FOR STORING IN L

INT ILP(01000);POINTERS TO MACROS
INT IMP;MACRO POINTER DURING EXPANSION
INT INM;NUMBER OF MACROS
INT IPR(010);PARAMETER VALUES
```

```
        ^!8P^U0S^P8S
        ^N3+
        )I^P9N(;
        @@^S0C@@^S0H;
:WHILE;
        ^!8P^U0S^P8S
        ^!9P)W^S0N(^P9S;
        @@^S0C@@^S0H;
:ON $$;
        ^N4+
        ^!8P^U0S^P8S
        ^N3+
:ENDWHILE;
        ^!8P^!9P^P8S
        ^N3+
        )W^P9N(;
        @@^S0C@@^S0H;
        ^!8P^!9P^P8S
:CHOOSE ON $$;
        ^N4+
        ^!8P^U0S^P8S
        ^!9P^U1S^P9S
        ^N3+
:CASE $$;
        ^!8P^!9P^P8S
```

*(Continued on page 28, column 2)*

## Tincmp Listing

```
TOP;TINCMP COPYRIGHT (C) 1981 W.A.GALE
LOMEM=16384;$4000
HIMEM=36860;$9000
REGISTER=086
BYTE AA;WORK
BYTE BB;WORK

BYTE BF(080);EXPANSION BUFFER
BYTE BL;BLANK
BYTE BP;POINTER INTO BF
BYTE C0;CONSTANT ZERO
BYTE C1;CONSTANT ONE
BYTE C2;CONSTANT TWO
BYTE C3;CONSTANT 3
BYTE C4;CONST 40
BYTE C8;CONSTANT EIGHTY
BYTE C9;CHARACTER! NINE
BYTE CC; INPUT CHARACTER
BYTE CX;CONSTANT TEN

BYTE DG;DIGIT FROM PARAMETER TREATMENT DEFINITION
BYTE DS(010);DIGIT STACK FOR SUB SD
BYTE EF;END OF FILE CHAR
BYTE F1(00275);INPUT BUFFER
BYTE F2(00275);OUTPUT BUFFER SIZES ARE MACHINE DEPENDENT
BYTE HA;'A'
```

*(Continued on top of page 28)*

```
INT ISS(040); INT TO HOLD NUMBERS-MAIN STACK
INT ITU;VALUE OF PARAMETER TO USE
INT IUU;SYMBOL GENERATOR(UNIQUE)
INT IXX;WORK
INT IYY;WORK
BEGINMAIN(AC,IAV)
NL=+141;!!!!!!!!!!!!!!MACHINE DEPENDENT
MS 'COPYRIGHT'
MS ' (C) 1981'
MS ' W.A.GALE'
WRITE NL
GOSUB IN;INITIALIZE INCL READ FLAG LINE
GOSUB RM;READ MACROS INTO TABLE
LOC 00;EXPANSION
WHILE
        READ CC FROM F1
        AA=ER==C0
ON AA;THAT IS, UNTIL EOF IS REACHED ON INPUT

      IF UG
              WHILE
                      AA=CC==0G
              ON AA;IGNORE LEADING IGNORE CHARACTERS
                      READ CC FROM F1
              ENDWHILE
      ENDIF
      BP=C1;BUF POINTER
      BF(C0)=CC
      WHILE
              READ CC FROM F1
              AA=CC!=NL
              BB=BP!=C8
              AA=AA&BB
      ON AA;WHILE LESS THAN 80 CHAR AND NOT NEWLINE
              BF(BP)=CC;THEN PUT IT IN A BUFFER FOR MULTIPLE COMPARISON
              BP++


       ENDWHILE
       WHILE
       AA=CC!=NL
       ON AA


               READ CC FROM F1
       ENDWHILE;HERE WE ARE DUMPING A LONG INPUT LINE
       BF(BP)=RC
       BP++
       BF(BP)=NL
       LE=BP
       AA=BP<=MM
       IF AA;TO SHORT TO MATCH
              ML=+000
              GOTO 17
       ELSE
              ML=+001
       ENDIF
       IDP=I00
       PP=C0
       IJJ=I00
```

*(Continued on next page)*

# Pidgin (Listing continued, text on pages 10-14)

```
        INM=C0
        WHILE
                AA=IDP<!IED;DEF PTR < END OF DEFINITIONS
        ON AA
                BP=C0
                WHILE
                        AA=BP<=LE
                ON AA
                        AA=LS(IJJ)
                        AA=AA==RC
                        03=BF(BP)
                        03=03==RC
                        AA=AA&03;IF BOTH THE BUFFER AND THE TARGET HAVE
;TEMPLATE END OF LINE, THEN WE HAVE MATCHED
                        IF AA
                                GOSUB DM;DO MACRO EXPANSION
                                GOTO 00
                        ELSE
                        AA=BF(BP)
                        BB=LS(IJJ)
                        AA=AA==BB
                        IF AA
                                GOTO 01;MATCHING
                        ELSE
                        AA=BB!=SF;NOT A TEMPLATE PARAMETER FLAG
                        IF AA
                                GOTO 10;MISMATCHED
                        ELSE;THIS IS A PARAMETER
                                PP++
                                AA=BF(BP)
                                IAA=AA
                                IPR(PP)=IAA
                        ENDIF
                        ENDIF
                        ENDIF
                LOC 01

                        BP++
                        IJJ++
                ENDWHILE
        LOC 10
                PP=C0
                INM++
                IDP=ILP(INM)
                IJJ=IDP
        ENDWHILE
        ;NO MATCH
LOC 17
        BP=C0
        WHILE
                CC=BF(BP)
                01=BP+C1
                AA=BF(01)
                AA=AA!=NL
        ON AA
                IF ML;THEN ALSO WRITE
                        WRITE CC
                ENDIF
                AA=I00<!IAA
ON AA
                ND--
                AA=DS(ND)
                WRITE AA
ENDWHILE
WRITE BL
ENDSUB


                                SUB CD;CONVERT AA AS DECIMAL DIGIT
BB=ZR<=AA
CC=AA<=C9
BB=BB&CC
IF BB


        AA=AA-ZR
        RETURN


ENDIF
AA=C0
ENDSUB

                                SUB CH;CONVERT AA AS HEX DIGIT
;ASSUMES THAT A-F ARE CONSECUTIVE

BB=ZR<=AA
CC=AA<=C9
BB=BB&CC
IF BB
        AA=AA-ZR
        RETURN
ENDIF
BB=HA<=AA
CC=AA<=HF
BB=BB&CC
IF BB
        AA=AA-HA
        AA=AA+CX


        RETURN
ENDIF
AA=C0
ENDSUB

                                SUB IN;INITIALIZE
ILM=+08920;DIMENSION OF LS -80
I00=+00000
I01=+00001
I10=+00010
I09=+00009
C0=+000
C1=+001
C2=+002
C3=+003
```

```
                WRITE CC INTO F2
                BP++
        ENDWHILE
        IF ML
                WRITE NL
        ENDIF
        IF UN;ONLY IF NOT SUPPRESSING
                WRITE NL INTO F2
        ENDIF
ENDWHILE
CLOSE F1
CLOSE F2
ENDMAIN
                SUB SD;THIS SUB CONVERTS "TOUSE" TO A NUMBER WITHOUT LEAD
AA=ITU<!I00
IF AA
        BB=+001
        ITU=-ITU
ELSE
        BB=+000
ENDIF
AA=ITU==I00
IF AA
        ND=C1
        DS(C0)=ZR
ELSE
        ND=C0
        WHILE
                AA=I00<!ITU
        ON AA
                IYY=ITU/I10
                IAA=I10*IYY
                IXX=ITU-IAA
                ITU=IYY
                AA=IXX
                AA=AA+ZR
                DS(ND)=AA
                ND++

        ENDWHILE
ENDIF
DS(ND)=OR
ND=ND+BB;IE INCR FOR NEG ONLY
ENDSUB
                        SUB WN;WRITE A NUMBER INTO F2
GOSUB SD;STACK THE DIGITS
WHILE;NOW WRITE OUT THE DIGITS FIRST TO LAST
        IAA=ND
        AA=I00<!IAA
ON AA
        ND--
        AA=DS(ND)
        WRITE AA INTO F2
ENDWHILE
ENDSUB
                        SUB PN;WRITE THE NUMBER ON THE TERMINAL
GOSUB SD;STACK THE DIGITS
WHILE
        IAA=ND
```

*(Continued on top of page 30)*

```
EF=+255;END OF FILE CHAR
C4=+040
C8=+080
I16=+00016
SP=+000
C9='9'
ZR='0'
BL=' ';BLANK
HF='F'
HA='A'
CX=+010

;READ CHARACTER FLAGS
IBC=IAV(C1)
TR='R'
CLOSE F1
OPEN F1 FOR TR AT IBC
IBC=IAV(C3)
TR='W'
CLOSE F2
OPEN F2 FOR TR AT IBC
READ AA FROM F1;X SUPPRESSES NEW LINE OUTPUT

OT='T'
UT=+000
BB='X'
UN=AA!=BB;UN SAYS THE CHAR WAS NOT X, SO DON'T SUPPRESS
READ RB FROM F1;BEGINNING OF DEFN FLAG
READ RC FROM F1;COMMENT AND END OF LINE FLAG
READ SF FROM F1;TEMPLATE PARAMETER FLAG
READ MF FROM F1;EXPANSION OPERATION FLAG

BB='X'
AA=MF==BB
IF AA
        UO=C0
ELSE
        UO=C1

ENDIF
OP='P';PARAMETER DESIGNATOR IN OPERATION SEQUENCE
OE='@';ESCAPE CHARACTER
OD='V';CONVERT PARAMETER TO DIGIT IN ACTION SEQUENCE
OB='!';POP STACK DESIGNATOR IN OPERATION SEQUENCE
OS='S';STACK DESIGNATOR IN OPERATION SEQUENCE
OH='H';HEX CONSTANT FETCH AND WRITE


;ALSO FOR HIGH BYTE OUTPUT
ON='N';LITERAL NUMERIC FETCH
OL='L';LITERAL BYTE FETCH
OC='C';CHARACTER OUT DESIGNATION
OA='+';ADD TO STACK DESIGNATION
OR='-';SUBTRACT (REDUCE) FROM STACK
OM='*'; MULTIPLY STACK BY BASE AND ADD
READ OG FROM F1;IGNORE CHARACTER
AA='X'
```

*(Continued on next page)*

# Pidgin (Listing continued, text on pages 10-14)

```
BB=AA==OG
IF BB
        UG=+000
ELSE
        UG=+001
ENDIF
READ CC FROM F1;NEWLINE
AA=NL!=CC;NL IS NEWLINE
IF AA
        MS 'FLAG LINE'
        STOP 1
ENDIF
IUU=+00100
ENDSUB;IN
                        SUB RM;READ MACROS
III=I00
INM=C0
MM=+127
WHILE
        READ CC FROM F1
        AA=ER==C0


ON AA
        CHOOSE ON CC
        CASE OE;ACCEPT THE NEXT CHARACTER UNCRITICALLY
                READ CC FROM F1
                GOTO 77
        CASE RB;BEGIN A DEFFINITION
                ILP(INM)=III
                INM++
                ML=+000

        CASE NL;IGNORE
        CASE RC;IGNORE FOLLOWING COMMENTS AND MARK THE LINE END
                LS(III)=RC
                III++
                AA=ML<!MM
                IF AA;THIS LINE SHORTEST YET
                        MM=ML


                ENDIF
                WHILE
                        READ CC FROM F1
                        AA=CC!=NL


                ON AA
                ENDWHILE
        CASE OG;IF USING IGNORE, IGNORE
                IF UG
                ELSE
                        GOTO 77


               ENDIF
        DEFAULT;
        LOC 77
          IMP++
          AA=LS(IMP)
          O2=AA
          GOSUB CD;FOR DIGIT CONVERSION
          DG=AA
          IMP++
          O3=LS(IMP); DESTINATION INDICATOR
          IF UT
                  WRITE O1
                  WRITE O2
                  WRITE O3
          ENDIF
          CHOOSE ON O1
          CASE OP;FETCH PARAMETER
                  ITU=IPR(DG);
          CASE OD;CONVERT FROM CHAR TO DIGIT
                  IAA=IPR(DG);
                  AA=IAA
                  GOSUB CD;DIGIT CONVERSION
                  ITU=AA
          CASE OB;POP STACK
                  ITU=ISS(SP)
                  AA=SP<=C0
                  IF AA
                          MS 'S STACKER'
                          WRITE NL
                          SP=C1
                  ENDIF
                  SP--;
          CASE OS;FETCH FROM STACK WITHOUT POP
                  ITU=ISS(SP)
          CASE OH;FETCH AND WRITE HEX CONSTANT BYTE
                  AA=O2
                  GOSUB CH
                  IAA=AA
                  IAA=IAA*I16
                  AA=O3
                  GOSUB CH
                  IBB=AA
                  ITU=IAA+IBB
                  O3=OC

          CASE OL;LITERAL BYTE FETCH
                  ITU=O2
          CASE ON;LITERAL DIGIT FETCH
                  AA=O2
                  GOSUB CD
                  ITU=AA
          CASE OT;TURN ON TRACE MODE
                  UT=+001
          DEFAULT;FETCH A UNIQUE NUMBER
                  ITU=IUU
                  IUU++

          ENDCHOOSE
          IF UT
                  III=ITU
                  GOSUB PN
```

```
                LS(III)=CC
                III++
                AA=ILM<!III
                IF AA

                        MS 'MACMEMXST'
                        WRITE NL
                        CLOSE F1
                        STOP 5


                ENDIF
                ML++
        ENDCHOOSE

ENDWHILE
AA=CC!=EF
IF AA
        MS 'DEFN READ'
        STOP 2
ENDIF
CLOSE F1
IBC=IAV(C2)
TR='R'
OPEN F1 FOR TR AT IBC
IED=III;END OF DEFINITIONS
MS 'LOADED...'
ITU=III
GOSUB PN
MS '.BYTES FO'
MS 'R DEFINES'
WRITE NL
ILP(INM)=III
ITU=INM
GOSUB PN
MS '.MACROS..'
ITU=MM
GOSUB PN
MS ' MIN LEN.'
WRITE NL
ENDSUB;RM
                        SUB DM;DO MACRO EXPANSION


IMP=IJJ+I01
INM++
IDP=ILP(INM)
WHILE

        AA=IMP<!IDP
ON AA;  UNTIL WE HAVE READ UP TO THE NEXT MACRO DEFINITION
        AA=LS(IMP)
        IF UO
                AA=AA==MF
        ELSE
                AA=C0
        ENDIF
        IF AA;OPERATION CODE
                IMP++
                O1=LS(IMP);FROM  INDICATOR
```

*(Continued on top of page 32)*

```
                        ITU=ISS(SP)
                        GOSUB PN
                        ITU=SP
                        GOSUB PN
                        ITU=III
                        WRITE NL
                ENDIF
                CHOOSE ON O3
                CASE OC;CHARACTER OUTPUT
                        AA=ITU
                        WRITE AA INTO F2
                CASE OS;PUT ON THE STACK
                        SP++
                        AA=C4<=SP
                        IF AA
                                MS 'S OVERFLW'
                                WRITE NL
                                SP=C4
                        ENDIF
                        ISS(SP)=ITU
                CASE OP;PUT INTO PARAMETER LOC
                        IPR(DG)=ITU
                CASE OA;ADD TO STACK
                        IAA=ISS(SP)
                        IAA=IAA+ITU
                        ISS(SP)=IAA
                CASE OR;SUBTRACT FROM STACK
                        IAA=ISS(SP)
                        IAA=IAA-ITU
                        ISS(SP)=IAA
                CASE OM; MULTIPLY BY BASE AND ADD
                        IAA=ISS(SP)
                        IAA=IAA*I10
                        IAA=IAA+ITU
                        ISS(SP)=IAA
                CASE OH; OUTPUT HIGH BYTE
                        UNPACK(ITU,AA,BB)
                        WRITE AA INTO F2
                DEFAULT;WRITE OUT AS A DECIMAL NUMBER
                        GOSUB WN
                ENDCHOOSE
        ELSE;END OF ACTION SECTION
        AA=LS(IMP)
        IF UN
                BB=AA!=RC
        ELSE
                BB=C1
        ENDIF
        IF BB
                WRITE AA INTO F2
        ELSE
                WRITE NL INTO F2
        ENDIF
        ENDIF
        IMP++
ENDWHILE
UT=+000;TURN OFF TRACE AT END OF MACRO
ENDSUB;DM
BOTTOM
```

**end**

# BASIC Rubik's Cube (Text on pages 19-21)

```
1000 PRINT :PRINT :PRINT
1010 PRINT "      RUBIK'S CUBE PLAYER          APRIL 27, 1981"
1020 PRINT
1030 PRINT "            J. W. PEHOUSHEK"
1040 PRINT "            6038 CRITTENDEN DR."
1050 PRINT "            CINCINNATI, OHIO 45244"
1060 PRINT
1070 REM  THE FOLLOWING CONVENTIONS ARE OBSERVED:
1080 REM       FACES ARE LABELED L, R, U, D, F, B
1090 REM       ROTATION IS CLOCKWISE = +,  CCW = -
1100 REM
1110 REM  THEREFORE, A CW ROTATION OF F FACE IS ENTERED AS  +F
1120 REM
1130 REM  ENTER MOVES IN RESPONSE TO THE PROMPT "COMMAND ?"
1140 REM  ENTER ONLY ONE MOVE AT EACH PROMPT.
1150 REM


1160 REM  COMMAND = M  SAVES ARRANGEMENT IN MEMORY
1170 REM  COMMAND = R  RECALLS ARRANGEMENT FROM MEMORY
1180 REM

1190 REM  TO DISPLAY CURRENT POSITION, ANSWER PROMPT WITH:
1200 REM       V  TO DISPLAY ON VIDEO
1210 REM       P  TO PRINT ON PRINTER
1220 REM  TWO VIEWS OF CUBE ARE PRESENTED.  THE LEFT HAND VIEW
1230 REM  IS GIVEN FACING FACE "F", THE RIGHT HAND
1240 REM  FACING FACE "B".
1250 REM  LOWER CASE LETTERS ARE USED TO REPRESENT
1260 REM  THE COLORS ON EACH FACELET.

1270 REM
1280 REM            u u u                     u u u
1290 REM            u U u                     u U u
1300 REM            u u u                     u u u
1310 REM
1320 REM  l l l   f f f   r r r   r r r   b b b   l l l
1330 REM  l L l   f F f   r R r   r R r   b B b   l L l
1340 REM  l l l   f f f   r r r   r r r   b b b   l l l
1350 REM
1360 REM            d d d                     d d d
1370 REM            d D d                     d D d
1380 REM            d d d                     d d d
1390 REM

1400 REM  VARIABLES USED:
1410 REM       A$ = GENERAL PURPOSE STRING
1420 REM       C$ = COMMANDS EXECUTED SINCE LAST PRINT
1430 REM        D = ROTATION INDEX:
1440 REM              1 = CW
1450 REM              0 = CCW
1460 REM             -1 = ERROR CONDITION
1470 REM              2 = INITIALIZE
1480 REM        S = FACE INDEX
1490 REM        L = NR. CHARACTERS IN C$
1500 REM        R = ROW INDEX
1510 REM        C = COLUMN INDEX

1520 REM       L( ),R( ),U( ),D( ),F( ), & B( ) ARE ARRAYS
1530 REM       CONTAINING CUBIE FACELETS AND ADJACENT EDGES.
1540 REM       L1( ) - - - B1( ) ARE MEMORY ARRAYS
1550 REM       T1( ),T2( ) = TEMPORARY ARRAYS
1560 REM
1570 REM
1580 REM
```

```
2190 F(4,4)=D(4,5):R(4,1)=D(4,5)
2200 B(4,1)=D(5,2):R(4,4)=D(5,2)
2210 R(4,3)=D(5,3)
2220 F(4,5)=D(5,4):R(4,2)=D(5,4)
2230 B(5,4)=D(2,2):L(5,2)=D(2,2)
2240 L(5,3)=D(2,3)
2250 F(5,2)=D(2,4):L(5,4)=D(2,4)

2260 B(5,3)=D(3,2)
2270 F(5,3)=D(3,4)
2280 B(5,2)=D(4,2):R(5,4)=D(4,2)
2290 R(5,3)=D(4,3)
2300 F(5,4)=D(4,4):R(5,2)=D(4,4)
2310 RETURN

2320 REM
2330 REM
2340 REM  Uface rotate
2350 IF D=2 THEN GOTO 2550
2360 FOR R=1 TO 5:FOR C=1 TO 5
2370 T2(R,C)=U(R,C):NEXT C:NEXT R
2380 IF D=1 THEN GOSUB 3950
2390 IF D=0 THEN GOSUB 4010
2400 FOR R=1 TO 5:FOR C=1 TO 5
2410 U(R,C)=T1(R,C):NEXT C:NEXT R
2420 REM  Uface (5x5) has been rotated.  Now equate edges.
2430 L(2,1)=U(1,2):B(2,4)=U(1,2)

2440 B(2,3)=U(1,3)
2450 R(2,5)=U(1,4):B(2,2)=U(1,4)
2460 L(2,2)=U(2,1):B(2,5)=U(2,1)
2470 R(2,4)=U(2,5):B(2,1)=U(2,5)
2480 L(2,3)=U(3,1)
2490 R(2,3)=U(3,5)
2500 L(2,4)=U(4,1):F(2,1)=U(4,1)
2510 R(2,2)=U(4,5):F(2,5)=U(4,5)
2520 L(2,5)=U(5,2):F(2,2)=U(5,2)
2530 F(2,3)=U(5,3)
2540 R(2,1)=U(5,4):F(2,4)=U(5,4)
2550 L(1,2)=U(2,2):B(1,4)=U(2,2)
2560 B(1,3)=U(2,3)
2570 R(1,4)=U(2,4):B(1,2)=U(2,4)
2580 L(1,3)=U(3,2)
2590 R(1,3)=U(3,4)
2600 L(1,4)=U(4,2):F(1,2)=U(4,2)
2610 F(1,3)=U(4,3)
2620 R(1,2)=U(4,4):F(1,4)=U(4,4)
2630 RETURN
2640 REM
2650 REM

2660 REM  Bface rotate
2670 IF D=2 THEN GOTO 2870
2680 FOR R=1 TO 5:FOR C=1 TO 5
2690 T2(R,C)=B(R,C):NEXT C:NEXT R
2700 IF D=1 THEN GOSUB 3950
2710 IF D=0 THEN GOSUB 4010
2720 FOR R=1 TO 5:FOR C=1 TO 5
2730 B(R,C)=T1(R,C):NEXT C:NEXT R

2740 REM  Bface (5x5) has been rotated   Now equate edges.
2750 R(1,4)=B(1,2):U(2,4)=B(1,2)
2760 U(2,3)=B(1,3)
2770 L(1,2)=B(1,4):U(2,2)=B(1,4)
2780 R(2,4)=B(2,1):U(2,5)=B(2,1)
```

```
1590 DIM R(5,5), L(5,5), F(5,5), B(5,5), U(5,5), D(5,5)
1600 DIM R1(5,5),L1(5,5),F1(5,5),B1(5,5),U1(5,5),D1(5,5)
1610 DIM T1(5,5), T2(5,5), C$(30)
1620 PRINT :PRINT
1630 LINE INPUT "DO YOU WANT TO ARRANGE CUBE YOURSELF ? ";A$
1640 IF A$="Y" THEN GOSUB 5030:GOTO 1700
1650 REM  INITIALIZE FACE VALUES
1660 FOR R=2 TO 4:FOR C=2 TO 4
1670 F(R,C)=102:L(R,C)=108:U(R,C)=117
1680 B(R,C)=98:R(R,C)=114:D(R,C)=100
1690 NEXT C:NEXT R
1700 GOSUB 4920

1710 PRINT :PRINT :PRINT :PRINT TAB(25);"INITIAL POSITION"
1720 GOSUB 4160
1730 LINE INPUT "COMMAND = ? ";A$
1740 IF A$="V" THEN GOSUB 4160:GOTO 1730
1750 IF A$="M" THEN GOSUB 5420:GOTO 1730
1760 IF A$="P" THEN GOSUB 4080:GOTO 1730
1770 IF A$="R" THEN GOSUB 5610:GOTO 1730
1780 D=-1
1790 IF LEFT$(A$,1)="+" THEN D=1
1800 IF LEFT$(A$,1)="-" THEN D=0
1810 IF D <> -1 THEN GOTO 1840
1820 PRINT "ERROR - INPUT AGAIN"
1830 GOTO 1730
1840 S=0

1850 A$=LEFT$(A$,2)
1860 IF RIGHT$(A$,1)="D" THEN S=1
1870 IF RIGHT$(A$,1)="U" THEN S=2
1880 IF RIGHT$(A$,1)="B" THEN S=3
1890 IF RIGHT$(A$,1)="F" THEN S=4
1900 IF RIGHT$(A$,1)="L" THEN S=5
1910 IF RIGHT$(A$,1)="R" THEN S=6
1920 IF S=0 THEN GOTO 1820
1930 C$=C$+A$+CHR$(32):L=LEN(C$)
1940 IF L >= 30 THEN GOSUB 4080
1950 ON S GOSUB 2030,2350,2670,2990,3310,3630


1960 GOTO 1730
1970 END
1980 REM
1990 REM
2000 REM  SUBROUTINES
2010 REM

2020 REM  Dface rotate
2030 IF D=2 THEN GOTO 2230
2040 FOR R=1 TO 5:FOR C=1 TO 5
2050 T2(R,C)=D(R,C):NEXT C:NEXT R
2060 IF D=1 THEN GOSUB 3950
2070 IF D=0 THEN GOSUB 4010
2080 FOR R=1 TO 5:FOR C=1 TO 5
2090 D(R,C)=T1(R,C):NEXT C:NEXT R
2100 REM  Dface (5x5) has been rotated.  Now equate edges
2110 B(4,5)=D(1,2):L(4,2)=D(1,2)
2120 L(4,3)=D(1,3)


2130 F(4,1)=D(1,4):L(4,4)=D(1,4)
2140 B(4,4)=D(2,1):L(4,1)=D(2,1)
2150 F(4,2)=D(2,5):L(4,5)=D(2,5)
2160 B(4,3)=D(3,1)
2170 F(4,3)=D(3,5)
2180 B(4,2)=D(4,1):R(4,5)=D(4,1)
```

*(Continued on top of page 34)*

```
2790 L(2,2)=B(2,5):U(2,1)=B(2,5)
2800 R(3,4)=B(3,1)
2810 L(3,2)=B(3,5)
2820 R(4,4)=B(4,1):D(5,2)=B(4,1)
2830 L(4,2)=B(4,5):D(1,2)=B(4,5)
2840 R(5,4)=B(5,2):D(4,2)=B(5,2)
2850 D(3,2)=B(5,3)
2860 L(5,2)=B(5,4):D(2,2)=B(5,4)
2870 R(2,5)=B(2,2):U(1,4)=B(2,2)
2880 U(1,3)=B(2,3)


2890 L(2,1)=B(2,4):U(1,2)=B(2,4)
2900 R(3,5)=B(3,2)
2910 L(3,1)=B(3,4)
2920 R(4,5)=B(4,2):D(4,1)=B(4,2)
2930 D(3,1)=B(4,3)
2940 L(4,1)=B(4,4):D(2,1)=B(4,4)
2950 RETURN
2960 REM
2970 REM


2980 REM   Fface rotate
2990 IF D=2 THEN GOTO 3190
3000 FOR R=1 TO 5:FOR C=1 TO 5
3010 T2(R,C)=F(R,C):NEXT C:NEXT R
3020 IF D=1 THEN GOSUB 3950
3030 IF D=0 THEN GOSUB 4010
3040 FOR R=1 TO 5:FOR C=1 TO 5
3050 F(R,C)=T1(R,C):NEXT C:NEXT R
3060 REM  Fface (5x5) has been rotated.  Now equate edges.
3070 L(1,4)=F(1,2):U(4,2)=F(1,2)
3080 U(4,3)=F(1,3)


3090 R(1,2)=F(1,4):U(4,4)=F(1,4)
3100 L(2,4)=F(2,1):U(4,1)=F(2,1)
3110 R(2,2)=F(2,5):U(4,5)=F(2,5)
3120 L(3,4)=F(3,1)
3130 R(3,2)=F(3,5)
3140 L(4,4)=F(4,1):D(1,4)=F(4,1)
3150 R(4,2)=F(4,5):D(5,4)=F(4,5)
3160 L(5,4)=F(5,2):D(2,4)=F(5,2)
3170 D(3,4)=F(5,3)
3180 R(5,2)=F(5,4):D(4,4)=F(5,4)
3190 L(2,5)=F(2,2):U(5,2)=F(2,2)
3200 U(5,3)=F(2,3)
3210 R(2,1)=F(2,4):U(5,4)=F(2,4)
3220 L(3,5)=F(3,2)
3230 R(3,1)=F(3,4)
3240 L(4,5)=F(4,2):D(2,5)=F(4,2)
3250 D(3,5)=F(4,3)
3260 R(4,1)=F(4,4):D(4,5)=F(4,4)
3270 RETURN
3280 REM
3290 REM


3300 REM  Lface rotate
3310 IF D=2 THEN GOTO 3510
3320 FOR R=1 TO 5:FOR C=1 TO 5
3330 T2(R,C)=L(R,C):NEXT C:NEXT R
3340 IF D=1 THEN GOSUB 3950
3350 IF D=0 THEN GOSUB 4010
3360 FOR R=1 TO 5:FOR C=1 TO 5
3370 L(R,C)=T1(R,C):NEXT C:NEXT R
3380 REM  Lface (5x5) has been rotated.  Now equate edges.
```

*(Continued on next page)*

# BASIC Rubik's Cube (Listing continued, text on pages 19-21)

```
3390 B(1,4)=L(1,2):U(2,2)=L(1,2)
3400 U(3,2)=L(1,3)
3410 F(1,2)=L(1,4):U(4,2)=L(1,4)
3420 B(2,4)=L(2,1):U(1,2)=L(2,1)
3430 F(2,2)=L(2,5):U(5,2)=L(2,5)
3440 B(3,4)=L(3,1)
3450 F(3,2)=L(3,5)
3460 B(4,4)=L(4,1):D(2,1)=L(4,1)
3470 F(4,2)=L(4,5):D(2,5)=L(4,5)
3480 B(5,4)=L(5,2):D(2,2)=L(5,2)
3490 D(2,3)=L(5,3)
3500 F(5,2)=L(5,4):D(2,4)=L(5,4)
3510 B(2,5)=L(2,2):U(2,1)=L(2,2)
3520 U(3,1)=L(2,3)
3530 F(2,1)=L(2,4):U(4,1)=L(2,4)
3540 B(3,5)=L(3,2)
3550 F(3,1)=L(3,4)
3560 B(4,5)=L(4,2):D(1,2)=L(4,2)
3570 D(1,3)=L(4,3)
3580 F(4,1)=L(4,4):D(1,4)=L(4,4)
3590 RETURN
3600 REM
3610 REM
3620 REM  Rface rotate
3630 IF D=2 THEN GOTO 3830
3640 FOR R=1 TO 5:FOR C=1 TO 5
3650 T2(R,C)=R(R,C):NEXT C:NEXT R
3660 IF D=1 THEN GOSUB 3950
3670 IF D=0 THEN GOSUB 4010
3680 FOR R=1 TO 5:FOR C=1 TO 5
3690 R(R,C)=T1(R,C):NEXT C:NEXT R

3700 REM  Rface (5x5) has been rotated.  Now equate edges.
3710 F(1,4)=R(1,2):U(4,4)=R(1,2)
3720 U(3,4)=R(1,3)
3730 B(1,2)=R(1,4):U(2,4)=R(1,4)
3740 F(2,4)=R(2,1):U(5,4)=R(2,1)
3750 B(2,2)=R(2,5):U(1,4)=R(2,5)
3760 F(3,4)=R(3,1)
3770 B(3,2)=R(3,5)

3780 F(4,4)=R(4,1):D(4,5)=R(4,1)
3790 B(4,2)=R(4,5):D(4,1)=R(4,5)
3800 F(5,4)=R(5,2):D(4,4)=R(5,2)
3810 D(4,3)=R(5,3)
3820 B(5,2)=R(5,4):D(4,2)=R(5,4)
3830 F(2,5)=R(2,2):U(4,5)=R(2,2)
3840 U(3,5)=R(2,3)

3850 B(2,1)=R(2,4):U(2,5)=R(2,4)
3860 F(3,5)=R(3,2)
3870 B(3,1)=R(3,4)
3880 F(4,5)=R(4,2):D(5,4)=R(4,2)
3890 D(5,3)=R(4,3)
3900 B(4,1)=R(4,4):D(5,2)=R(4,4)
3910 RETURN
3920 REM
3930 REM

3940 REM  ROTATE ARRAY CW SUBROUTINE
3950 FOR R=1 TO 5:FOR C=1 TO 5
3960 T1(C,6-R)=T2(R,C):NEXT C:NEXT R
3970 RETURN
3980 REM
```

```
4590 IF A=2 THEN PRINT " ";CHR$(F(4,C));" ";
4600 IF A=3 OR A=4 THEN PRINT " ";CHR$(R(4,C));" ";
4610 IF A=5 THEN PRINT " ";CHR$(B(4,C));" ";
4620 IF A=6 THEN PRINT " ";CHR$(L(4,C));" ";
4630 NEXT C:PRINT SPC(1);
4640 IF A=3 THEN PRINT SPC(2);
4650 NEXT A
4660 PRINT :PRINT
4670 PRINT TAB(11);
4680 FOR R=2 TO 4
4690 PRINT " ";CHR$(D(R,4));" ";
4700 NEXT R
4710 PRINT TAB(43);
4720 FOR R=4 TO 2 STEP -1
4730 PRINT " ";CHR$(D(R,2));" ";

4740 NEXT R
4750 PRINT :PRINT TAB(11);
4760 PRINT " ";CHR$(D(2,3));"  D  ";CHR$(D(4,3));" ";
4770 PRINT TAB(43);
4780 PRINT " ";CHR$(D(4,3));"  D  ";CHR$(D(2,3));" ";
4790 PRINT :PRINT TAB(11);
4800 FOR R=2 TO 4
4810 PRINT " ";CHR$(D(R,2));" ";
4820 NEXT R
4830 PRINT TAB(43);
4840 FOR R=4 TO 2 STEP -1
4850 PRINT " ";CHR$(D(R,4));" ";
4860 NEXT R
4870 PRINT :PRINT :PRINT
4880 RETURN
4890 REM
4900 REM

4910 REM  SUBROUTINE TO INIT 5x5 ARRAYS
4920 D=2
4930 GOSUB 2030
4940 GOSUB 2350
4950 GOSUB 2670
4960 GOSUB 2990
4970 GOSUB 3310
4980 GOSUB 3630
4990 RETURN
5000 REM
5010 REM

5020 REM USER ARRANGED INPUT SUBROUTINE
5030 PRINT :PRINT "FACELETS ARE LABELED AS FOLLOWS:"
5040 PRINT "      F ( 2,2 ) = Upper left corner"
5050 PRINT "      F ( 2,4 ) = Upper right corner"
5060 PRINT "      F ( 4,2 ) = Lower left corner"
5070 PRINT "            etc."

5080 PRINT :PRINT "ENTER LETTER FOR COLOR":PRINT
5090 PRINT :PRINT :
5100 FOR R=2 TO 4:FOR C=2 TO 4
5110 PRINT "FACE F (";R;C;") = ";:LINE INPUT ;A$
5120 F(R,C)=ASC(A$):NEXT C:NEXT R
5130 FOR R=2 TO 4:FOR C=2 TO 4
5140 PRINT "FACE B (";R;C;") = ";:LINE INPUT ;A$
5150 B(R,C)=ASC(A$):NEXT C:NEXT R
5160 FOR R=2 TO 4:FOR C=2 TO 4
5170 PRINT "FACE L (";R;C;") = ";:LINE INPUT ;A$
5180 L(R,C)=ASC(A$):NEXT C:NEXT R
```

```
3990 REM
4000 REM  ROTATE ARRAY CCW SUBROUTINE
4010 FOR R=1 TO 5:FOR C=1 TO 5
4020 T1(6-C,R)=T2(R,C)
4030 NEXT C:NEXT R
4040 RETURN
4050 REM
4060 REM
4070 REM  PRINTER SUBROUTINE
4080 GOSUB 5360
4090 PRINT "MOVES SINCE LAST PRINTOUT:":PRINT C$:C$=""
4100 GOSUB 4160
4110 GOSUB 5560
4120 RETURN
4130 REM
4140 REM

4150 REM  PRINT BOTH VIEWS OF CUBE SUBROUTINE
4160 PRINT
4170 PRINT TAB(11);
4180 FOR C=2 TO 4
4190 PRINT " ";CHR$(U(2,C));" ";
4200 NEXT C
4210 PRINT TAB(43);
4220 FOR C=4 TO 2 STEP -1
4230 PRINT " ";CHR$(U(4,C));" ";
4240 NEXT C
4250 PRINT :PRINT TAB(11);
4260 PRINT " ";CHR$(U(3,2));"  U  ";CHR$(U(3,4));" ";
4270 PRINT TAB(43);
4280 PRINT " ";CHR$(U(3,4));"  U  ";CHR$(U(3,2));" ";
4290 PRINT :PRINT TAB(11);
4300 FOR C=2 TO 4

4310 PRINT " ";CHR$(U(4,C));" ";
4320 NEXT C
4330 PRINT TAB(43);
4340 FOR C=4 TO 2 STEP -1
4350 PRINT " ";CHR$(U(2,C));" ";
4360 NEXT C
4370 PRINT :PRINT
4380 FOR A=1 TO 6
4390 FOR C=2 TO 4

4400 IF A=1 THEN PRINT " ";CHR$(L(2,C));" ";
4410 IF A=2 THEN PRINT " ";CHR$(F(2,C));" ";
4420 IF A=3 OR A=4 THEN PRINT " ";CHR$(R(2,C));" ";
4430 IF A=5 THEN PRINT " ";CHR$(B(2,C));" ";
4440 IF A=6 THEN PRINT " ";CHR$(L(2,C));" ";
4450 NEXT C:PRINT SPC(1);
4460 IF A=3 THEN PRINT SPC(2);
4470 NEXT A
4480 PRINT

4490 PRINT " ";CHR$(L(3,2));"  L  ";CHR$(L(3,4));SPC(3);
4500 PRINT CHR$(F(3,2));"  F  ";CHR$(F(3,4));SPC(3);
4510 PRINT CHR$(R(3,2));"  R  ";CHR$(R(3,4));SPC(4);
4520 PRINT " ";CHR$(R(3,2));"  R  ";CHR$(R(3,4));SPC(3);
4530 PRINT CHR$(B(3,2));"  B  ";CHR$(B(3,4));SPC(3);
4540 PRINT CHR$(L(3,2));"  L  ";CHR$(L(3,4));

4550 PRINT
4560 FOR A=1 TO 6
4570 FOR C=2 TO 4
4580 IF A=1 THEN PRINT " ";CHR$(L(4,C));" ";
```

*(Continued on top of page 36)*

```
5190 FOR R=2 TO 4:FOR C=2 TO 4
5200 PRINT "FACE R (";R;C;") = ";:LINE INPUT ;A$
5210 R(R,C)=ASC(A$):NEXT C:NEXT R
5220 FOR R=2 TO 4:FOR C=2 TO 4
5230 PRINT "FACE U (";R;C;") = ";:LINE INPUT ;A$
5240 U(R,C)=ASC(A$):NEXT C:NEXT R
5250 REM  D-FACE HAS DIFFERENT SEQUENCE FROM OTHERS
5260 FOR R=2 TO 4:FOR C=2 TO 4
5270 PRINT "FACE D (";R;C;") = ";:LINE INPUT ;A$
5280 D(C,6-R)=ASC(A$):NEXT C:NEXT R:PRINT
5290 RETURN
5300 REM
5310 REM
5320 REM  SET UP PRINTER SUBROUTINE
5330 REM  ***************** NOTE **********************
5340 REM  NEXT TWO LINES SET UP H-8 & IDS 460 PRINTER
5350 REM  *********************************************
5360 PORT OUT 216,9600
5370 PRINT CHR$(31);CHR$(1);CHR$(5);CHR$(6);CHR$(13);
5380 RETURN
5390 REM
5400 REM

5410 REM  MEMORY SUBROUTINE
5420 LINE INPUT "SURE ? ";A$
5430 IF A$()"Y" THEN RETURN
5440 FOR R=2 TO 4:FOR C=2 TO 4
5450 F1(R,C)=F(R,C):L1(R,C)=L(R,C):U1(R,C)=U(R,C)
5460 B1(R,C)=B(R,C):R1(R,C)=R(R,C):D1(R,C)=D(R,C)
5470 NEXT C:NEXT R
5480 GOSUB 4080
5490 GOSUB 5360

5500 PRINT "THE ARRANGEMENT ABOVE HAS BEEN SAVED"
5510 GOSUB 5560
5520 RETURN
5530 REM
5540 REM

5550 REM  RESET PRINT OUTPUT TO H-8 VIDEO SUBROUTINE
5560 PORT OUT 250
5570 RETURN
5580 REM
5590 REM

5600 REM  RESTORE CUBE TO MEMORIZED ARRANGEMENT SUBROUTINE
5610 LINE INPUT "SURE ? ";A$
5620 IF A$()"Y" THEN RETURN
5630 FOR R=2 TO 4:FOR C=2 TO 4
5640 F(R,C)=F1(R,C):L(R,C)=L1(R,C):U(R,C)=U1(R,C)
5650 B(R,C)=B1(R,C):R(R,C)=R1(R,C):D(R,C)=D1(R,C)
5660 NEXT C:NEXT R
5670 GOSUB 4920
5680 GOSUB 5360
5690 PRINT :PRINT

5700 PRINT "THE FOLLOWING ARRANGEMENT HAS BEEN RESTORED"
5710 GOSUB 5560:C$="":GOSUB 4080
5720 RETURN
```

**end**

# Of Interest

by Dave Cortesi

## SCRIBBLE and MINCE Group

Sounds like a club for subway vandals, doesn't it? Or possibly one for cookbook authors. It's really the **Amethyst Users Group,** an organization for those sharing a common interest in the Amethyst editor-formatter package described so enthusiastically in *Dr. Dobb's Journal* #54. Annual dues are $6, which is also the tentative price for diskettes of club programs. The latter will be made available in a number of formats.

## Plan Well Ahead

Here are items you might want to write into your calendar. The New York Computer Expo **(NYCE)** is scheduled for New York City, August 12 to 15. The organizers (who claim credit for the former National Small Computer Show) plan to charge $10 a day, or $200 for a business person's tutorial and five day's admission.

The IEEE Computer Society will hold its **COMPCON '81** September 14-17 in Washington, DC. Conference concerns will center on productivity issues.

Those whose concerns center more on buying and selling will want to know about the second annual **NJ Micro Show** and fleamarket. That's to be held at the Holiday Inn (north) Convention Center, Newark NJ International Airport, on October 24th and 25th. Exhibit space is available; $150 gets you a booth, electricity, a table and two chairs.

The ACM and the US Dept. of Commerce are planning what sounds like a very interesting conference in March 1982. The conference title is "Human Factors in Computer Systems" and **Michael L. Schneider,** program chairman, is looking for papers now. Two of the general categories are "Human-Computer Interface Design" and "Impact of Computers on Human Behaviour." People who work with personal computers should have a **lot** to say under those headings, and "papers which present practical experience . . . will be considered."

## Settle Down for a Good Read

The mail brought blurbs for several hardware books this month. One is *PET Interfacing* by Downey and Rogers (262 pp, $17, from **Group Technology, Ltd.**). It claims to cover each of the PET's interfaces – user port, memory expansion port, and IEEE-488 bus port – in detail, giving a breadboard circuit and a number of experiments for each.

Working engineers and serious hobbyists may want the *Electronic Circuits Notebook* from McGraw-Hill (344 pp, $32.50). A compilation by the staff of *Electronics* magazine, the book is said to contain 268 circuits chosen to emphasize "fresh ideas and new approaches."

The same publisher has released the *User's Guidebook to Digital CMOS Integrated Circuits* by Eugene R. Hnatek (339 pp, $24.50). The work is said to cover the entire CMOS field with emphasis on LSI circuits, to be practical in approach, and to give specific applications.

**RCA** has what should be a good companion volume to the one above: the latest edition of 688-page *COS/MOS Integrated Circuits* data book. SSD-250B is their order number; it costs just $7.

## Settle Down for a Free Read

The mail also brought a number of offers of free catalogs. **DEC** offers engineers their Software Referral Catalog, listing 300 software packages from independent vendors. The **COMAL Information Exchange** wants to send you an information package on that language, currently all the rage in Europe. **ENTELEK** (could that mean "mind" in Esperanto?) is, apparently, a book dealer specializing in computer books and computer programs. They offer a catalog of micro-oriented books, many accompanied by program disks for TRS-80 and Apple.

We got catalogs from two dealers in supplies for small computers: **MISCO** and **Inmac.** We compared prices on a couple of products; MISCO's were slightly lower. On the other hand, Inmac doesn't charge shipping on prepaid orders. Inmac is running a contest with an Apple II as first prize.

## Two Unusual Periodicals

We've now received volume 1, number 1 of **Software News,** a monthly tabloid devoted to computer software ($20 per year). It certainly wouldn't be fair to judge a periodical on the content of its first issue, but we wonder a bit at the paper's format. It uses five columns on newsprint, resulting in the dim gray look standardized by such well-known industry tabloids as *Computerworld.* We'd hoped for something more attractive; the recent improvements in *InfoWorld* show what can be done with the tabloid format.

When you read a computer magazine, do you spend more time on the display ads than you do on the articles? Don't blush, many do. **Gordon Publications** has the answer. They publish *Computer Products,* a bi-monthly magazine that eliminates all that annoying editorial matter; it contains nothing but glossy display ads, page after slick, 11" by 14" page of them. $6 per issue.

## TRS-80, Speech Impediment Removed . . .

A package from **Baysik Speech** adds the SAY verb to TRS-80 BASIC. Having loaded it (it's self-relocating and won't disturb any other special drivers, they claim) users of the Radio Shack voice synthesizer can code SAY much the way they code PRINT. Coding SAY "7/4="; 7/4 is supposed to produce the words "seven divided by four equals one point seven five" from the synthesizer. The product allows words to be programmed with English letters rather than special characters: "BAYSIK" instead of "B"+CHR$(94)+"S#K". The package costs $125.

## . . . Writes for Other CPUs . . .

**Allen Ashley** has produced a set of six matched cross-assemblers for the TRS-80. Each package lets your TRS-80 become a development system for one of: the Intel 8048, the RCA 1802, the National COP400, the Zilog Z-8, the Fairchild 3870, or the AMI S2000. Each assembler uses, "with minor exceptions," the mnemonics of the target CPU, and costs $75 on diskette or tape. Once a program is assembled it must be sent to the target machine. Allen Ashley suggests sending the data through a CPU port direct. However . . .

## . . . Burns PROMs for Them, Too

. . . **Orion Instruments** has an EPROM programmer for the TRS-80, so you could put the cross-assembled program right onto a chip. The "Developmate 81" plugs into the expansion port, can handle just about any current chip, and comes with software for $329. The same box contains a Z80 in-circuit emulator. A 40-pin connector mates the Z80 socket of a target system to the TRS-80 bus so that the target can be completely controlled

and checked out from a program in the TRS-80.

## Apple II Bus Stretches Out

Great ideas are always simple, yes? Our vote for clever product of the month goes to **Passport Designs** for the Softbus. It's simply an Apple II bus card with nothing on it but the heads of three 16-conductor ribbon cables. The three-foot cables terminate at their other ends in 16-pin connectors so they can be pushed into a circuit breadboard. The effect is to bring one position on the Apple bus out to the top of the bench and spread it on the breadboard where you can build circuits around it.

## Oh Boy, A New Acronym

Just when we thought we'd figured out the difference between PROM, EPROM, and EEPROM, along came **Xicor** to announce NOVRAM. That stands for "non-volatile RAM." What it means is a single chip with some RAM and some EEPROM. The RAM is used as it normally would be, but on receipt of a TTL signal the chip copies the contents of RAM into the EEPROM, where it can remain "indefinitely." A different signal loads the saved data back to the RAM. A NOVRAM with 256 by 4 bits is to be in production this month, with a quantity price of $12.

## A CP/M Software Index

The **Small Systems Group** has released its second Index of CP/M Software, listing 740 programs from 248 vendors. The press release comments that the software industry must be fairly healthy since only 9 vendors of 101 listed in the prior Index have gone out of business. The index can be had for $6, or $8 outside North America.

## A Diskette Conversion Service

**Prestige Services** writes to tell us that they are prepared to move data between the diskettes used by any pair selected from the following: IBM 3740, Apple II, DEC RT-11, TRS-80, and CP/M exchange format. They don't indicate what the charge would be nor what volume they can handle, but software vendors might like to check them out.

## MP/M-86 Is Out

**Digital Research** has finally dropped the other shoe, announcing MP/M-86 as a companion system for CP/M-86. We expected that the two dash-86 systems would be more compatible with each other than were CP/M and MP/M. The 8086 hardware architecture relieves the restrictions that (we thought) forced MP/M into such contorted design tricks as page-relocation. It seems that this will be the case, for the release claims that "CP/M-86 programs can be operated immediately under MP/M-86," something no one could have said about the 8080 systems.

## OASIS Spreads Out

**Phase One Systems** sent us releases announcing that they now have versions of OASIS for North Star machines and for the Cromemco System 3. Owners of those machines now have a choice of three operating systems each: the manufacturer's DOC, CP/M, and OASIS. Would someone like to write a side-by-side-by-side review for *DDJ*?

■■

Engineers, scientists, systems programmers, and business owners... All these read *Dr. Dobb's Journal*. Reach them with cost-effective advertising. (415) 323-3111.

## Contact Points

**Allen Ashley,** 395 Sierra Madre Villa, Pasadena, CA 91107; (213) 793-5748.

**Amethyst Users Group,** Barry A. Dobyns, POB 8173, University Station, Austin, TX 78712; (512) 441-9466.

**Baysik Speech,** Suite 289, 1259 El Camino Real, Menlo Park, CA 94025.

**COMAL Information Exchange,** 5501 Groveland Terrace, Madison, WI 53716.

**COMPCON '81,** POB 639, Silver Spring, MD 20901; (301) 589-3386.

**DEC** Engineering Systems Group, MR1-1/M75, 200 Forest St., Marlboro, Mass. 01752, attention SRC Manager.

**Digital Research,** POB 579, Pacific Grove, CA 93950; (408) 649-3896.

**ENTELEK,** Ward-Whidden House, The Hill, POB 1303, Portsmouth, NH 03801,

**Gordon Publications,** POB 2106R, 20 Community Pl., Morristown, NJ 07690; (201) 267-6040.

**Group Technology, Ltd.,** POB 87, Check, VA 24072; (703) 651-3153.

**Inmac** Department 10, 2465 Augustine Drive, Santa Clara, CA 95051.

**Michael L. Schneider,** Program Chairman, c/o Sperry-Univac, POB 200, Blue Bell, PA 19424; (215) 542-5808.

**MISCO,** POB 399NR, Holmdel, NJ 07733; toll-free 800-631-2227 outside NJ.

**NJ Micro Show,** c/o Kengore Corporation, 3001 Route 27, Franklin Park, NJ 08823; (201) 297-2526.

**NYCE,** 110 Charlotte Place, Englewood Cliffs, NJ 07632; (201) 569-8542.

**Orion Instruments,** 172 Otis Ave., Woodside, CA 94062; (415) 851-1172.

**Passport Designs,** POB 478, La Honda, CA 94020.

**Phase One Systems,** 7700 Edgewater Drive Suite 830, Oakland, CA 94621; (415) 562-8085.

**Prestige Services Division,** 909 N. Coliseum Blvd., Fort Wayne, IN 46805; (219) 423-1571.

**RCA** Solid State Division, POB 3200, Somerville, NJ 08876; (201) 685-6423.

**Small Systems Group** CP/M Software Indes, POB 5429, Santa Monica, CA 90405; (213) 392-1234.

**Software News,** Technical Publishing Co., 666 Fifth Ave., NY, NY 10103.

**Xicor** Inc. – only a phone number given: (408) 734-3041.

# Dr. Dobb's Clinic

by D. E. Cortesi, Resident Intern

## The Doctor is Overwhelmed

The first Clinic produced a very gratifying response. More than 60 readers took the trouble to write, and PCC thanks all of them. We're obviously onto something good with the Clinic and we'll try to keep it going (but see the exhortation at the end).

## The Most and the Least

Many readers sent methods of defining MIN and MAX in BASIC. Although there were variations in the syntax the solutions could be boiled down to four, two based on arithmetic and two based on boolean operations. We'll give only the MIN solution here; MAX falls out from MIN in each case by changing a plus to a minus or vice versa.

The popular choice was an expression that was explained most succinctly by Philip Bacon of Gainesville, FL. His letter, in full, said "The minimum of A and B is their average minus half the distance between them:"

```
MIN(A,B) = (A+B - ABS(A - B))/2
```

A smaller group took essentially the same route but using SGN rather than ABS:

```
MIN(A,B) = ((A+B) -
(A-B)*SGN(A-B))/2
```

Several correspondents noted that such expressions are unreliable when the difference between A and B is near the limit of precision of the floating point representation of the system.

About half the writers made use of the fact that a relational expression returns a numeric value. In most Microsoft BASIC's, – 1 means "true" and zero means "false". Multiplying by a relationship is a way of selecting or not selecting a value based on the truth of the relationship. Therefore we have:

```
MIN(A,B) = -A*(A <=B)
- B*(B < A)
```

The minus signs adjust for "true" being represented as –1; if your BASIC returns positive 1 for "true", change both minuses to plusses. Note that one of the relations must include an equal sign. If it's absent the expression yields zero when A=B.

Harry Saal of Palo Alto, CA and Dale Schafer of Bryan, TX were the only readers to come up with this elegant use of the relational:

```
MIN(A,B) = A + (A-B)*(A > B)
```

This again assumes that "true" is –1. Both expressions could be made independent of the truth value and thus fully portable, but only at some cost in performance. The trick is to use the value to adjust for its own sign:

```
MIN(A,B) = (1=1)*A*(A <=B)
+ (1=1)*B*(B < A)
```

```
MIN(A,B) = A + (-1=1)*(A-B)
*(A > B)
```

Allen Ashley of Pasadena, CA pointed out that if MIN is limited to binary integers, then boolean operations can replace the arithmetic ones:

```
MIN(J,K) = J AND (J < K) OR
K AND (J=> K)
```

Bob Goodman of Dublin, CA, extended selection by truth value to give a MIN function for character variables! Here it is:

```
MIN(A$,B$) = LEFT$ (A$, ABS
(A$ < B$) *LEN (A$)) + LEFT$
(B$,ABS (A$=> B$) *LEN (B$))
```

All in all, the Doctor's problems have been well and truly minimized.

## The Matter of CP/M Comments

We stumbled onto the fact that a CP/M command line that started with a colon was treated as a comment (to give belated credit where it's due, we first saw that use of the colon in a submit file that's part of the Pascal/Z compiler from Ithaca Intersystems). We asked if that was a reliable feature of CP/M and its companion systems.

John Pierce of Digital Research wrote to us as follows:

> "In fact, CP/M systems consider ";" (semicolon) to be the proper comment delimiter on command lines. This not only holds true for all existing systems (including 1.4, 2.2, MP/M, and CP/M-86), but will continue to be supported in future versions. The present CCP ignores lines that start with most terminators, but we will not guarantee that this will always work."

Our thanks to John for the clarification, and apologies to readers who've been putting colonized comments in their submit files. We hope to see lots more contributions from John and his colleagues in Pacific Grove.

## Approximately Right

In the May Clinic we gave an approximation to the hypotenuse of a right triangle:

```
happrox = max(dh,dv) +
min(dh,dv)/2
```

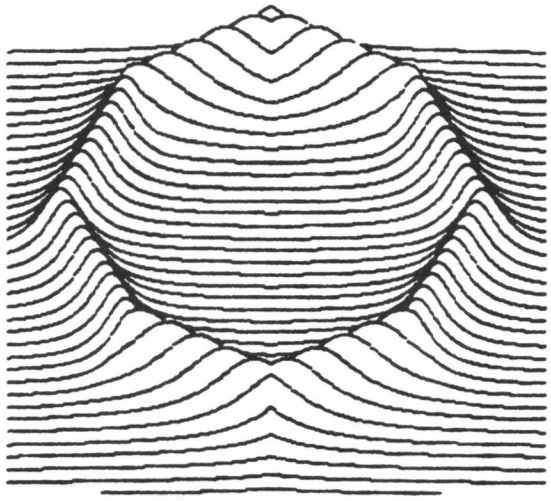
or, the hypotenuse is approximately the longer side plus half the shorter. Bob Hoolko of Grants Pass, OR responded as follows.

> I have seen your "max(dx,dy) + min(dx,dy)/2" handy approximation for SQR (dx**2 + dy**2) used in digital radar signal processing to calculate the magnitude of complex numbers. This method is easy to implement in high speed hardware. Unfortunately if you use it to try to plot a circle, you get an octagon instead of a circle. The two plots of lunar craters below illustrate the effects of using the handy approximation (fig. 1) and the approximation 'dx +dy–dx*dy/(dx+dy+1)' (fig. 2). Figure 2 can be derived by applying one iteration of Newton's method to the first approximation dx+dy. The '+1' in the denominator is to prevent division by zero when both dx and dy are zero. This approximation costs four adds, one multiply, and one divide which is usually faster than the one add, two multiplies, and one square root of the exact method. And if you are willing to pay for one more floating multiply 'dx + dy – 1.17*dx*dy/(dx+dy+1)' looks even better.

Other readers improved on the approximation, too. For simplicity, assume that dh>dv; then we can dispense with max and min. The original is then

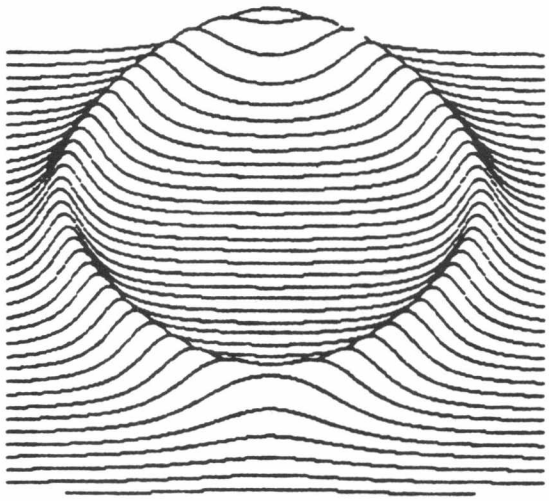
```
(1)   happrox = dh + dv * .5
```

Ed Wilson of Greenbelt, MD derived a much more accurate formula:



r = min (dx,dy) +max (dx,dy)/2

**Figure 1**



r = dx+dy−dx*dy/(dx+dy+1)

**Figure 2**

```
happrox = dh + (dv * dv)/(2 * dh)
```

and justified it three ways, from a Taylor expansion, a geometric argument, and from Newton's method. In a postscript he noted that using .4142135 in place of .5 in (1) would improve it.

Edmund Butler of E. Riverdale, MD took a different route to a similar number. He generalized (1) to the form

```
happrox = dh * A + dv * B
```

and, converting the problem to polar coordinates, analyzed it to find the A and B that would minimize the relative error. The worst error occurs when the included angle is 0 or pi/4 (a fact also noted by M. David Leonard of Leverett, MA). Butler's argument led him to best values of A=.9541 and B=.4127.

Gerald Edgar of Columbus OH proposed A=.960 and B=.398. In an informal bake-off, Wilson's formula showed the smallest error by far, while Butler's choices for A and B edged out Edgar's by a nose. The best balance of speed and accuracy is probably Butler's, while Wilson's expression would be preferable when accuracy is important.

## 2000 AD is, so, a Leap Year

All centuries divisible by 400 are leap years, as any fool knows. Except the fool preparing this column, and then only sometimes. The Intern's foolishness was revealed even more embarrasingly on the first of March this year. On that date, users of a certain popular APL interpreter signed on to their systems and were told that the date was 3/0/81. That unique date came from a piece of code that the Intern wrote some five years ago, a routine not greatly different from the MP/M date routine presented in this column. Let this be a warning to all – calendar routines don't get any safer with age. Who'd look for a bug to turn up in the month *after* February in the year *after* a leap year?

L. Barker of Chicago, and C. T. McDaniel of Los Gatos, CA both have the calendar under firm control. Both sent programs in BASIC that can handle any reasonable date with aplomb. McDaniel sent subroutines in North Star BASIC for converting Julian dates to Gregorian, and vice versa (Listing 1, page 42). Barker sent a delightful program that prints a month calendar for any chosen month of any year from 1900 to 2099 (Listing 2, page 42).

## The Clinic Needs Sponses

The heavy mail after the first clinic was very gratifying to everyone on the *DDJ* staff, as we said above. Unfortunately, almost all the letters that came in were *re*sponses to the questions posed. Those are certainly good – what would be the point of posing questions if readers didn't answer them? – but if the Clinic is to be healthy it must have more than just *re*sponses.

What's needed are the problems and discoveries that any serious computist makes weekly. Anytime a computing problem makes you think hard for an hour or more, that problem is material for the Clinic. Jot it down and send it along, even if you don't have the perfect solution, even if you don't have a solution at all. As should be clear from the material above, the *DDJ* readership is capable of finding not one but several solutions to any problem. The letters that will give this column real body are the ones that open with phrases like:

> "It isn't commonly known that you can . . ."
> "I am absolutely stumped. I want to . . ."
> "The documentation for X doesn't say so, but . . ."
> "I just found a way to . . ."

Keep the sponses and responses coming. ■■

LISTING ON PAGE 42

## LISTING I by C. T. McDaniel *(Text on page 41.)*

```
10000 REM THIS SUBROUTINE COMPUTES THE GREGORIAN
10050 REM   CALENDAR DATE CORRESPONDING TO
10100 REM   A GIVEN JULIAN CALENDAR DATE 'D'
10200 REM
10300 REM FOR EXAMPLE, IF THE JULIAN DATE D = 2440588,
10400 REM   THE YEAR  (I) = 1970
10500 REM   THE MONTH (J) = JANUARY = 1
10600 REM   THE DAY   (K) = FIRST OF THE MONTH (1)
10700 REM
10800 REM
10900 REM BEGIN COMPUTATION
11000 REM
11100 C1=146097 \ C2=2447
11200 L=D+68569
11300 N=INT(4*L/C1)
11400 L=L-INT((C1*N+3)/4)
11500 I=INT(4000*(L+1)/1461001)
11600 L=L-INT(1461*I/4)+31
11700 J=INT(80*L/C2)
11800 K=L-INT(C2*J/80)
11900 L=INT(J/11)
12000 J=J+2-12*L
12100 I=100*(N-49) +I+L
12200 L=D-7*INT(D/7)+I
12300 RETURN
20000 REM
20100 REM THIS FUNCTION DETERMINES THE DAY NUMBER D
20150 REM  ON THE JULIAN CALENDAR, USING
20200 REM  THE FOLLOWING GREGORIAN INFORMATION
20300 REM
20400 REM  I = YEAR
20500 REM  J = MONTH
20600 REM  K = DAY OF MONTH
20700 REM
20800 REM  WHERE I,J,K ARE GREGORIAN DATES AS INTEGERS.
20900 REM
21000 REM FOR EXAMPLE, IF
21100 REM  I = 1970, J = 1 (JANUARY) AND K = 1 (FIRST)
21200 REM  THEN D = 2440588
21300 REM
21400 REM THE INTERVAL BETWEEN TWO GREGORIAN DATES
21500 REM  CAN THUS BE COMPUTED BY SUBTRACTING
21550 REM  THE JULIAN DATES FOR EACH.
21600 REM  (NOTE: NORTH STAR BASIC MULTI-LINE FUNCTION)
21700 DEF FNA (I,J,K)
21800 L=-INT((14-J)/12)
21900 D=K-32075+INT(1461*INT(I+4800+L)/4)+
                        INT(367*INT(J-2-INT(12*L))/12)
22000 D=D-INT(3*INT(INT((I+4900+L)/100)/4))
22100 RETURN D
22200 FNEND
```

## LISTING II by L. Barker *(Text on page 41.)*

```
10 REM ZELLER'S CONGRUENCE
20 DEFINT A-Y
30 DEFSTR Z
40 INPUT"Month";M
50 INPUT"Year";Y
60 IF Y  1900 OR Y  2099 GOTO 40
70 IF M  1    OR M  12   GOTO 40
80 MONTH = M : YEAR = Y : GOSUB 280
90 DAY = D : YEAR = Y : MONTH = M +1
100 IF MONTH  13 GOTO 120 ELSE MONTH = 1
110 YEAR = YEAR + 1
120 GOSUB 280
130 D = 28 + D - DAY
140 IF D  28 THEN D = D + 7
150 FOR I = 1 TO M
160     READ ZMONTH
170 NEXT
180 LPRINT " "; ZMONTH; TAB(25);
190 LPRINT USING "####";Y
200 LPRINT " Sun Mon Tue Wed Thu Fri Sat"
210 LPRINT TAB(DAY * 4 +1);
220 FOR I = 1 TO D
230   LPRINT USING "  ##";I;
240   IF (I + DAY) MOD 7 = 0 AND I  D THEN LPRINT
250 NEXT
260 LPRINT
270 END
280 REM Compute the day of the week
290 MONTH = MONTH - 2
300 IF MONTH  0 GOTO 320 ELSE MONTH = MONTH + 12
310 YEAR = YEAR -1
320 CENTURY = YEAR\100
330 YEAR = YEAR MOD 100
340 D=((13*MONTH -1)\5 + 5*YEAR\4 + CENTURY\4 -
                                  2*CENTURY +1) MOD 7
350 RETURN
360 DATA January,February,March,April,May,June
370 DATA July,August,September,October
380 DATA November,December
```




**"This will never do! I can understand every single word!"**

# Book Reviews

**Programming the Z8000**
**By Richard Mateosian**
Published by **Sybex**
**$15.95, 297 pages**
Reviewed by **Terry R. Dettmann**

Sybex Books has gained a reputation for putting out good books on a variety of microcomputer-related subjects. This new book on the Z8000 processor is another fine addition to that line.

Mateosian covers all the essential details of the Z8000 processor for people who want to know what it is and how to use it. He starts out with a chapter on basic concepts dealing with programming, algorithms, flowcharts, and representations of information. This chapter proceeds rapidly from bits to stacks and makes a good review for the reader.

Chapters 2, 4, 5, and 7 provide descriptive information about the processor itself that would be useful to a design engineer who has no real interest in programming. Chapter 2 covers the architecture of the Z8000. If you haven't any prior familiarity with microprocessors at the hardware level, this chapter can be a little rough going, but anyone considering using the Z8000 should at least scan the chapter.

Chapter 4 covers the Z8000 instruction set in detail. The first part of the chapter deals with topics such as "Data Operations and Testing," "Data Transfers," "Pointer Setting," "Transfer of Control," "Input/Output," "CPU Control," "Block Operations," "Multi-Micro Synchronization," "Condition Codes," and "Instruction Formats." Part 2 of this chapter is a reference manual of the instructions themselves.

Chapter 5 covers Z8000 addressing modes (8 pages) and Chapter 7 covers Z8000 peripheral components such as the memory management unit, the serial communication controller and so forth.

The other chapters of this book are aimed more at the programmer. Chapter 3 covers elementary programming techniques by developing an algorithm to encrypt text and programming it on the Z8000.

Chapter 6 covers input/output techniques. Chapter 8 covers a variety of utility programming examples useful for the non-segmented version of the processor. Specifically, routines are developed and presented for an I/O initiator, ring buffer routines, line handling routines, translation routines, a terminal interaction package, input decoding, output formatting, cursor control, and bit tables.

Chapter 9 deals with a variety of advanced programming techniques such as storage management, timesharing, and stack management. Finally Chapter 10 covers the use of various program development tools such as assemblers and text editors. The discussion is general and not specifically related to one or another system.

Five appendices are provided: A, Answers to selected exercises; B, ASCII Character Set; C, Encodings for instruction fields; D, Control Register Formats; and E, Layout of the Program Status Area.

Spot checking the book against Zilog's Z8000 Product Specification and the Z8000 Family brochure turned up no errors. Overall I found the book easy to read and well worth the effort to do so.

**BASIC Scientific Subroutines Vol. I**
**by F. R. Ruckdeschel**
Published by **McGraw-Hill**
**$19.95, 317 pages**
Reviewed by **Terry R. Dettmann**

Are you using BASIC in your scientific work? Have you ever wanted a quick routine to do a little task such as plotting, complex variables, vector or matrix operations, random number generation, or series approximation? Well, this could be your source.

Ruckdeschel has compiled a series of useful routines in these areas for people who are using BASIC for their scientific work. This book is the first of a promised series of books that will compile routines in statistics, mathematics, engineering and the sciences for use by everyone who is applying BASIC to computations in these fields.

The dialect for the programs is close to North Star BASIC while observing some limits over what is possible with the language to make these routines compatible with more BASIC's. In addition, a special appendix is provided that covers translation of the routines to other BASIC's as well as a complete listing of the routines in Microsoft BASIC.

This book is more than just a collection of programs, however. It covers the routines, how to use them, and the theory behind them in a clear and concise manner. Admittedly, someone who hasn't had any exposure to some of the techniques might be taken aback by the discussions.

For example, in talking about least squares regressing, Ruckdeschel speaks about obtaining near mini-max approximations with Chebyshev polynomial expansions. The discussion is elementary to someone with experience, but less clear to those without. Given supplementary material expanding the theoretical treatments, even novice users could understand the techniques.

Even without understanding them, a novice user can take the routines and plug them into his programs where needed by following the simple instructions on the use of the routines.

The text identifies the variables used by the subroutines as well as those to be passed with parameters. All of the subroutines are given unique line numbers so that a complete library could be assembled and loaded as needed. Users with the necessary technical knowledge could write their own loader to resolve subroutine calls to this library and load only those routines needed for a program they are working on.

Since receiving the book, I've tried a number of the routines under Microsoft BASIC and found them reliable when used according to the instructions. I have trouble in general using libraries such as this since it is simply more fun to develop my own routines, but I've found myself going to this book despite that for routines I've needed because they are so simple to use when I want something fast.

I'm looking forward to seeing Volume 2 in the series.

**The PLL Synthesizer Cookbook**
**By Harold Kinley**
Published by **TAB**
**$7.95, 279 pages**
Reviewed by **Terry R. Dettmann**

Many Microcomputer hobbyists are also electronics hobbyists. In fact, that's how many people got into microcomputers in the first place. This book is for the electronics hobbyists among you.

Anything but elementary, this book picks up with an explanation of how PLL synthesizers work and then devotes a chapter each to 10 popular synthesizers. The work is heavily oriented to their use in CB radios and uses an example from CB for each of the ten synthesizers.

There is enough detail here to satisfy the potential experimenter on how to develop program control of a synthesizer, how to generate desired frequencies from a master oscillator, and so forth. At no point in the book does the explanation

ever get low enough for the introductory student, however.

This book thoroughly covers the topic of PLL synthesizers. It provides the kind of information on connections, use and troubleshooting that will please both professionals and hobbyists. If electronics is your business or your hobby, and generation of accurate frequencies is necessary to you, then I recommend this book as a good reference for your shelf.

**Conversational Languages**
**By I. Kupka and N. Wilsing**
Published by **John Wiley & Sons**
**$27.75, 117 pages**
Reviewed by **Doug MacDonald**

Computers are not friendly creatures. Despite the incredible technological advances since the first vacuum tube monsters, the man-machine dialogue is still strained at best. Computers are unforgiving, maddeningly logical, heartless creatures with whom you can't imagine (outside of *Star Wars*) having a friendly conversation. (Of course, to the basement hacker whose first programming was of the "finger-blistering" variety on a front panel of binary switches, *anything* is friendly!)

Nevertheless, a number of attempts have been made to produce conversational languages – languages and operating systems that provide a high degree of interaction between machine and user. The fact that these languages are not widely known or used is only an indication of how difficult the problem is.

Two German authors, Kupka and Wilsing, present a somewhat thick and scholastic outline of some of these conversational languages and the problems in developing and using them. The book is based on a series of lectures given at the University of Hamburg.

The authors begin by making the distinction between a programming language and a command language. The various so-called high-level languages such as FORTRAN, ALGOL and PL/1 are notoriously unfriendly, demanding of syntax and often awkward. But even when the necessary program has been designed and written, there is still the business of getting it into the computer, compiling or interpreting it into machine code and performing the run.

It is here – the time spent while the programmer is actually squared away at the CRT keyboard – that most of the effort has been made to unfreeze the heartless computer.

In the program-entry stage, various attempts have been made to assist the programmer with his development. The authors use ALGOL, the granddaddy of Pascal, as an example. With the computer's responses italicized, the dialogue might look like:

begin[
*illegal position of '['*
ifa
*illegal position of 'a'*
*suggestion: 'if' OK?*
begin real array

Efficient perhaps, but definitely not friendly – more like a stern old school marm ready to pounce on a trembling child.

One step up would be a computer-aided program development system. Suppose the user has a light pen in addition to the CRT keyboard. On bringing up the editor, he/she sees:
*program*
He types 'fourier', giving the program a name
Then appears:
'begin' <declarations> <statements> 'end'
The user points with the light pen to 'declarations'. The machine responds:
<constant> <variable> <type>
The user selects 'variable and the machine awaits a variable identifier. The dialogue might continue:
array [charlie] of <simple type> < set of> <record> <file of>

Here we have an "intelligent" editor for Pascal which checks the programmer's syntax and helps him by filling in expected words. This is a little friendlier.

The authors devote only a few paragraphs to examining the real problems involved in getting logical machines to understand illogical people. If the computer responds too quickly, the user is "stressed"; if it responds too slowly, the user is bored. Can a machine be made to understand "The man, who struck the dog, which bit the cat, with the stick, fell down"?

The main portion of the book is taken up in describing a number of "desk calculator" type languages which are used interactively, most of them obscure – JOSS, LCC, PPL, CPS and so on. There are outlines of AMTRAN, DIALOG and VENUS. Most of these offbeat languages were developed for very specific tasks such as experimental numerical mathematics or simulation.

There are short outlines of BASIC and APL, the two most widely-known interactive languages.

The final chapters are comprehensible only to those who have studied the theory of formal languages. To the uninitiated, it's all page after page of chicken scratches of meta-language sets, lattices of equivalent dialogue systems and other arcane symbols.

Most copies of this book will likely end up on the shelves of those dusty, cramped little offices of graduate students in computing science. The book is definitely not for the novice. However, if you're a serious student of programming language design, by all means give it a read.

■■

If you want to present your ground-breaking work to some of the most innovative and influential people in the microcomputing industry, *DDJ* can help. Contact us at:

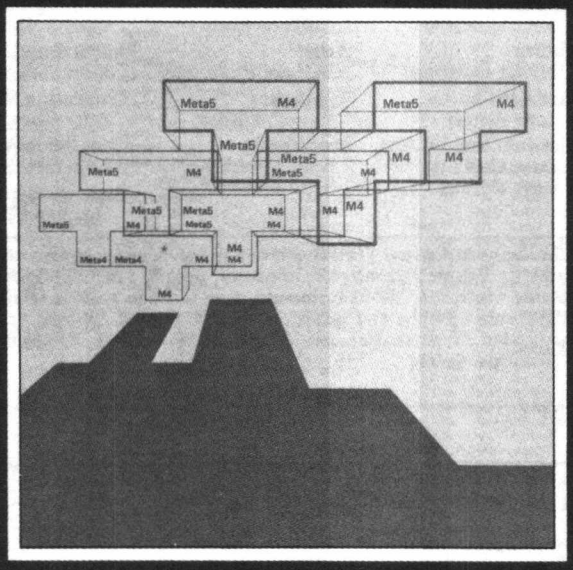
**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

$2.50
($3.00 in Canada)

# DR. DOBB'S JOURNAL

For Users of Small Computer Systems

Number 58 August 1981 Volume 6, Issue 8




**A Tool for Writing Compilers**

**About C**

**Card Playing Permutations**

**Write Your Own Compiler With Meta4**
*William A. Gale*

**Some Useful C Time Functions**
*Bill Bolton*

**The Doctor's Clinic**
*D. E. Cortesi*

**... and much more!**

DR. DOBB'S JOURNAL
For Users of Small Computer Systems

August 1981 Volume 6, Issue 8, Number 58

# Contents

Letters. . . . . 4
**Write Your Own Compiler With Meta4** . . . . . 6
**Listing**. . . . . 22
*by William A. Gale*
**Some Useful C Time Functions**. . . . . 16
**Listing**. . . . . 18
*by Bill Bolton*
Dr. Dobb's Clinic. . . . . 40
Of Interest. . . . . 43

© 1981 by People's Computer Company unless otherwise noted on specific articles. All rights reserved.

**Publisher** Michael Madaj
**Editor** Marlin Ouverson
**Assistant Editor** Julie Anton
**Contributing Editors** Dave Caulkins, Dave Cortesi
**Art Director** Clifford West
**Artist** Barbara Ruzgerian
**Typesetter** Renny Wiggins
**Proofreader** Nancy Heubach
**Marketing Director** Craig S. Harper
**Advertising Sales** Kent Safford
**Circulation Manager** Peter Clark
**Circulation Assistant** Leah Dansby
**Business Manager** Maureen Christine

*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025.

# Editorial

Planning each issue of *DDJ* is like playing a game of editorial bingo. We sit down at a long table with our game card and pieces of corn to lay on the squares. Across the top of the card are the column labels – 8080, 6502, 1802, 6800, etc. The horizontal rows are categorized into algorithm, languages, hardware, programming problems, and more. Now and then our pieces form a straight line and we all yell, "Bingo!"

That kind of thing happened with last year's all-CP/M issue, and with *DDJ* #56, which featured Jonathan Mills' cross-assembler for the 8080/Z80. We now have plans to publish the complete listing of the cross-assembler in forthcoming issues. Also coming up is an all-FORTH issue, for which articles and too-rare utilities have been trickling in.

It is often true that along with articles of major import come listings of considerable length. *Dr. Dobb's Journal* traditionally offers useful listings in their entirety, rather than just short excerpts as illustrations for an article. Sometimes that means we do not have enough space to present the usual variety of articles, because the listings in a particular issue are so voluminous. But we are banking on the fact that the usefulness of the listings will more than compensate for those occasional times when the number of articles is fewer than normal.

As always with *DDJ*, the success of the journal depends on the responsive readership. So contribute an article, an idea or a subscription! Do you think we are missing the boat? Is there a subject to which you would like an entire issue devoted? Have you a piece of useful software you'd like to make available to your fellow readers? If you just have an opinion, a well-written letter to the editor is always welcome.

There are other ways to support your magazine, of course. Mention us to your local bookstore, give a gift subscription to a friend, or get a lifetime subscription for yourself. Why not? Ours is the only respected computer journal which has maintained its friendly voice over the years. Our readers are more important to us than any other resource, and we always enjoy hearing from you.



Marlin Ouverson
Editor

# Letters

## When Push Comes to Shove . . .

Dear Mr. Ouverson:

I was alarmed by Jonathan Mills' implication that *Cross-Assembler for 8080 or Z80* might not be continued.

The contents of the article and listings not only provide an entree to 8086 projects but also give a clear example of compiler techniques. Regardless of immediate reader response, I urge you to continue this series. This is likely to become one of the most widely used and referenced articles in microcomputer literature.

I hope that Mr. Mills will include a (MAKETABLE) listing in a future article and identify the assembler he used.

Sincerely,
Don Vickers
P. O. Box 51, Parnassus Sta.
New Kensington, PA 15068

Dear Marlin:

It has been several months of wading through issues of *Byte*, *Interface*, *Kilobaud*, *Electronics*, *Electronic Design* et al. before I accidentally discovered Jonathan Mills' "A Cross-Assembler for the 8080/Z80" in your June 1981 issue. This series of upcoming articles on a 8080/8086 cross-assembler looks like it could be one of the best ever in your magazine.

Only recently I began to question whether I should have renewed my subscription. I'm glad I did because otherwise I would have missed this one. One of the most unfortunate aspects about the title is that it appears to be another (ho-hum) 8080/Z80 cross-assembler. From what I can tell it looks like the best thing to hit the microcomputer software market in a long time.

Bravo, bravo on your magazine which has been able to attract such a talented professional as Mr. Mills. I'm sure his contribution to the world of public domain software will not go unrewarded.

Anxiously awaiting my next issue of *DDJ*,

Yours,
Tom Sullivan
6543 Girvin Dr.
Oakland, CA 94611

Dear Mr. Ouverson:

It was with great delight that I read the article on the Cross-Assembler for the 8080 to 8086/8088. I do hope that you will publish the entire listing.

The Z80/8088 CPU outlined in the article sounds exciting and is more in line with the requirements that I had outlined for an entry into the 8088 world than any of the other CPU's offered to date. This includes single processor (8088 or 8086) or the dual processor boards. I had almost given up hope of finding a board that satisfied my needs.

Sincerely,
Larry Hoffman
10152 Oso Ave.
Chatsworth, CA 91311

*Dr. Dobb's Journal*:

I have just finished reading the first installment of the Cross-Assembler article by Jonathan Mills. Apparently, reader response will determine whether or not you print the remainder.

This is a response. Please print it.

Regards,
R. A. Morton
12603 Mistletoe Trail
Manchaca, TX

---

*Editor's Note: We had hoped for this kind of response to Mr. Mills' work. The above letters are only a few of those we received asking for more from the author. So, stay tuned – the next installment is on its way in an upcoming issue. The size of the listing prevents us putting it all into one issue, but if your letters keep coming in like this, we'll publish it all!*

---

## Beat the Odds

Dear People,

Please enter a new subscription to *Dr. Dobb's Journal* for me at the address given. By my calculations, *Dr. Dobb's Journal*'s last 3 issues have contained more useful information than the last 3 years of *Byte* magazine! *Byte* seems to be ½% useful information, 4½% general news and 95% advertising – 94% of which is useless to overseas readers. I feel guilty (and light of pocket) consuming so much American paper for so little benefit! I fully endorse and heartily commend *DDJ*'s advertisement policies.

Best regards,
C. I. Yandell
3/8 Spray St.
Elwood, 3184 AUSTRALIA

## CP/M's ED

Gents:

Users of the CP/M standard editor, "ED" have probably noticed that ED interprets an ASCII form feed <OC> as an imbedded carriage return. This is an amazing irritation to those of us with normal, form-feed-recognizing line printers.

ED may be altered to treat form feeds normally provided some other control character is substituted for control-L <OC>. A few tedious minutes with DDT revealed the following locations must be changed (addresses in hexadecimal):

528 162E 1A11.

I have named the result "RALPH." RALPH interprets a control-N <OC> as an imbedded carriage return.

Sincerely,
J. Elliott Smith
6 Leslie Road
Belmont, MA 02178

## A DAL Blurb

Dear Marlin:

I would appreciate it if you would pass the following information on to *DDJ* readers:

The source code and machine-readable documentation for my small language DAL (Data Acquisition Language) is now available on an 8" CP/M or ISIS diskette for a $10.00 copying charge. It was previously only available on 9-track tape. Complete assembly listings are still available for $10.00, the cost of printing and mailing. The user manual was printed in *DDJ* Vol. 4, No. 10 (Nov/Dec 1979). DAL is written in 6502 assembly language and contains many interesting features, including a variable precision (to 64 bytes) arithmetic package.

Thank you.

Sincerely,
Gregg E. Marshall
P. O. Box 3282
Walnut Creek, CA 94598

## Ergo, I Think!

Dear *DDJ*:

This letter is in response to a number of letters from readers inquiring about the particular BASIC used in the ERGO thinking program (*DDJ* #53, Mar. 1981).

The program was written in Tymshare SUPER-BASIC in 1969. Some readers may be able to get a copy of a Tymshare manual for SUPER-BASIC.

There are also similar versions of SUPER-BASIC offered by NCSS and UCS on their systems.

ERGO did not use many of the special features of SUPER-BASIC. I will try to describe those few that it does use.

All the variables used in the program are declared at the beginning. This is not required, but it is helpful. The LOGICAL declaration is for Boolean variables, which can only have values of 0 (false) or 1 (true). These can be used in conditions and in any arithmetic expressions.

An unusual feature is the "modifier" form of statement, in which any statement (or group of assignment statements), can be followed by IF, UNLESS, FOR, WHILE, or UNTIL. IF at the end of a statement means the same as at the beginning, but doesn't need the THEN keyword. UNLESS is the opposite of IF. FOR does a normal FOR loop without needing a NEXT (same as the FORTRAN implied DO-loop). WHILE and UNTIL are loop control modifiers; the loop repeats until the condition is false (WHILE), or true (UNTIL).

Assignment statements, separated by commas, can be executed as a one-line compound block.

The functions are all pretty standard. INDEX is a string function that returns the position of the second string argument within the first string argument. Later versions of SUPER-BASIC provided a 3-argument INDEX, so it can be given a starting position for the scan.

The data file used by ERGO is a sequential file with very simple structure. The "IS" and the "HAS" relations are described by string arrays, with two nouns for each entry. The array dimensions of each category are at the beginning of the file, and the arrays for the relations are then dimensioned 5 times bigger to make room for more relations deduced by the program, or told to it by the person running it.

Scanning these relation arrays in memory is done sequentially. This is inefficient, and an indexing or hash-code arrangement would be better if much larger amounts of data were to be handled.

Some readers objected to the lack of detailed documentation in ERGO. The purpose of the listing was not to provide a detailed design to be copied, but to encourage readers to devise their own approaches. The real value of this program was to demonstrate that it doesn't take a lot of code to do simple thinking.

Sincerely,
Sylvan Rubin
Algorithmic Engineering
P. O. Box 3074
Santa Clara, CA 95055

*Editor's Note: Any readers who have artificially intelligent programs which occupy a reasonably conservative amount of memory (i.e., accessible to most of us homebodies) are hereby officially invited to contribute their listings and articles to* DDJ *for publication.*

If you think you have an idea of interest to *DDJ* readers, we would like to discuss it with you. Contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# Write Your Own Compiler

by William A. Gale

Users of small computer systems have many projects for which standard computer languages are not well adapted. Yet writing a compiler for a new language has usually been a time-consuming task, even using a large computer. This paper describes a tool, Meta4, for writing compilers for small computers. It consists of a specialized interpreter and a specialized language for writing compilers. A language is completely specified, both syntactically and semantically, by a program in the Meta4 language.

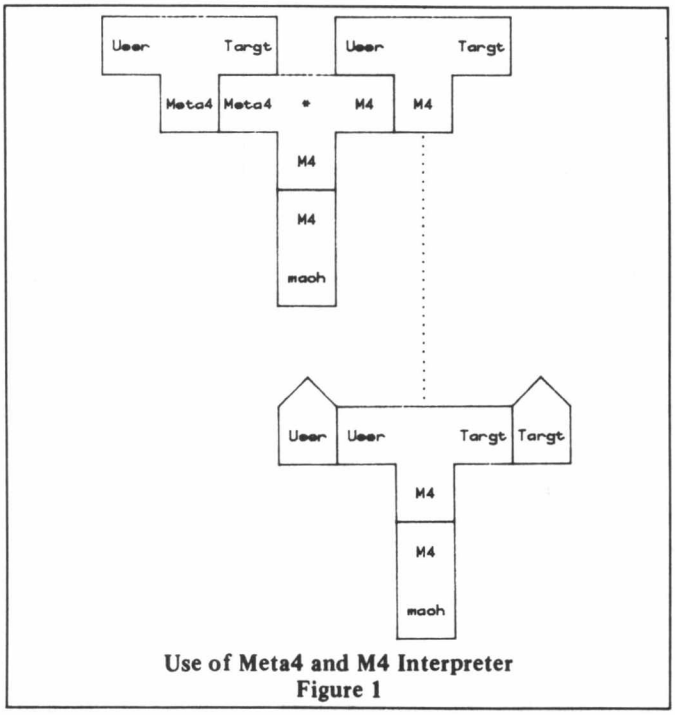
Meta4 is an outgrowth of Meta II, first described in Schorre (Proceedings of 19th National Conference of the ACM, 1964), reprinted in this magazine in April 1980. As I looked for a name, I found articles describing a Meta 3 (Schneider and Johnson, Proceedings of 19th National Conference of the ACM, 1964), and a Meta 5 (Oppenheim and Haggerty, Proceedings of the 21st National Conference of ACM, pp. 465-468). Probably there was a Meta4 also, but I didn't find a description, and it's hard to pass up such a good word play.

The reason that a 1964 compiler writing tool is interesting today is that today's microcomputers are comparable in size and speed to many of the larger 1964 computers. There has been progress in compiler writing tools, of course, so that better tools now exist. But the Meta's are both simple enough for a hobbyist to understand and powerful enough to be useful. An important point for me was that Meta4 was simple enough to use Pidgin to implement (see *DDJ* #57). If you want to pursue recent work, a good place to start is a four-article review contained in the August 1980 issue of the IEEE's *Computer*. It thus becomes part of a full boot strap operating system.

You won't need to know about Meta II to understand this article. But if you decide to implement Meta4, an understanding of Meta II will be useful. Meta4 keeps the ways that Meta II deals with lexical analysis, syntax analysis, and output. It provides better error handling, a symbolic memory for defining variables, arithmetic capabilities, memory stacks so code can be generated for machines without stacks, and facilities for generating machine code as extensions beyond Meta II.

The interpreter for Meta4 is written in Pidgin. The reason in a nutshell is portability. Pidgin, described in last month's issue, is a tiny language fairly easy to compile to machine code. In using Pidgin, I was most interested in cross-time portability for myself onto my next machine. But I also think that it is a fine means for describing to 8080, Z80, or 6800 processors the results of my work on a portable operating system.

An overview of how Meta4 works and what you will need to do in order to implement it will also serve to name the several languages and programs involved. A compiler involves three languages: the source language (what it translates from), the target language (what it translates to), and the coding language (what it itself is written in). To use Meta4, you need to write a compiler from the Userlang source language to the Targetlang target language, in the Meta4 coding language. This program is shown by a "T" in the upper left of Figure 1, the diagram coming from a paper by Earley and Sturgis, "A Formalism for Translator Interactions," CACM, p. 607, 1970. A translator is shown with the source language in the left arm, the target language in the right arm, and the coding language in the base. An interpreter is shown with a coding language on the bottom of a rectangle and the interpreted language on the top. In the figures, programs provided here by listing have an asterisk in the center of their figure.




**Use of Meta4 and M4 Interpreter**
**Figure 1**

Now your compiler program is the input to a compiler with source language Meta4, a target language called M4, and which is coded in M4. This compiler program is provided by a listing accompaying this article, so you have that part. M4 is executable by an interpreter, for which the listing in Pidgin is provided. For now, suppose you have the interpreter in machine language. Then M4 is executable, and you can run the compiler from Meta4 to M4. When you input your compiler from Userlang to Targetlang written in Meta4, the output is a compiler from Userlang to Targetlang written in M4. But M4 is executable so you can now (as shown in the lower right) input your program, written in Userlang and get as output your program, written in Targetlang.

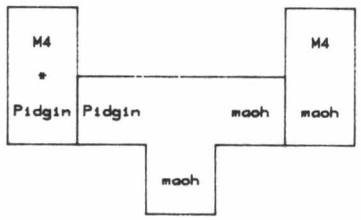
You can see that the M4 interpreter plays a key role in this process, because it is used once when you translate your compiler from Meta4 to M4, and again when you translate your program from Userlang to Targetlang. What you need to do to get Meta4 working is shown in Figure 2 – the listing of M4 interpreter in Pidgin is provided. You need a translator from Pidgin to your machine code that is executable on your machine, and that's what you got in last month's article.

Since the key program in Meta4 is an interpreter, it is best termed a compiler generator interpreter. That is, it is an

© *1981 by William A. Gale. All Rights Reserved.*

*The author, who works in statistics and economics at Bell Telephone Laboratories, can be reached at 439 S. Orange Ave., South Orange, NJ 07079.*

**Producing an M4 Interpreter**
**Figure 2**




interpreter intended to both generate and run compilers.

To use Meta4 you have to understand the Meta4 language. So the next section gives you a start towards understanding it by going over the compiler from Meta4 to M4 written in Meta4. This is the first listing following the article (see page 22). It defines Meta4 exactly in terms of M4. The following section has some thoughts on *how* to use Meta4, for it can be used many ways. To implement Meta4, or to understand completely the definition of Meta4, you will need to understand M4, which is described in detail next. The article concludes with some help for the implementor.

## Meta4 Language

The listing of the compiler from Meta4 to M4 in Meta4 shows exactly what syntax constructions will be recognized, and how they will be compiled to M4. What I say here should be taken as a guide to that listing. It may sometimes be useful to compare parts of this listing with its translation (which is the listing of the Meta4 to M4 compiler in M4). The top line says to start processing with the subroutine called "COMPILE", which is at the bottom of the listing. COMPILE says to look first for the exact characters '.SYN', and then for an identifier (.ID, a letter followed by any number of letters or digits), then to generate a mark command (.OUT ('XM;')), a subroutine call to the identifier found (.OUT ('G'*)), and a stop command (.OUT ('E')) so the translation of this top line is

```
XM;
GCOMPILE
E,
```

as can be seen in the Meta4 to M4 in M4 listing. The commands being output are M4 commands, explained in detail later. The syntax description then consists of zero or more "DECLARE's" ($DECLARE) followed by zero or more "STATEMENT's" ($STATEMENT). Let's check out the STATEMENTs, first, because they came from Meta II, and are the core of the parser.

Finding the subroutine STATEMENT just above COMPILE, you see that a STATEMENT is

(1) An identifier, after which a label (.LAB *) with the symbolic name of the identifier will be generated;
(2) The exact character =
(3) A PHRASE, and
(4) The exact character ';' after which a return ('R') will be generated.

A PHRASE, in turn, has one of three alternative structures, which are written separated by the character '/'. It can be the exact characters '.PREP' followed by zero or more NOTSYN's, followed by an ALTS; or it can be just an ALTS; or it can be one NOTSYN followed by zero or more NOTSYN's. The first alternative that is satisfied will be accepted and others will not be checked. This is accomplished by generating a jump on true before each alternative as shown in ALTS. An ALTS is a SEQ followed by zero or more alternatives, each of which starts with the exact character '/', at which point we generate a jump on true to the label *1, followed by another SEQ. After all the alternatives we define the label *1 to which we generated jumps.

A SEQ, now, consists of a SYN followed immediately by zero or more SYN's or NOTSYN's. If the first SYN is not matched, we will just jump to the end of the SEQ (.OUT ('F/' *1)) but if any following SYN returns without matching, we have an error (.OUT ('X')). For following NOTSYN's we do not generate an error test.

A SYN is a basic syntactical item, for which it can be stated that one does or one does not occur next in the input. It is (1) a *new* identifier, whch is compiled as a subroutine call, *or* (2) a single quoted string, which is compiled as a string to match exactly, *or* (3) the keyword '.ID' which is compiled as a lexical search for an identifier *or* (4) the keyword '.NUM' which is compiled as a lexical search for a number *or* (5) the keyword '.STR' followed by either a singly or doubly quoted string which is compiled as a lexical search for a string quoted by the first character of the string given, *or* (6) the keyword '.EMP' which compiles as a command to set the flag, *or* (7) a '$' followed by a SYN, which is compiled as a request for zero or more of the specified SYN, *or* (8) the exact character '(' followed by a PHRASE followed by the exact character ')'. Any of these may be followed by an optional error message.

These subroutines define the parsing that Meta4 allows, and a few comments on their use may be worthwhile here. This is called recursive descent parsing, and it requires being able to tell which construct is to be chosen (at ALTS) by looking just at the next input symbol.

It works like this. A SYN always returns true or false, along with its specification. If the first SYN in a SEQ is false, we jump over the rest of the SEQ, still with false. If it is true, we go into the SEQ. If a SEQ returns false, the ALTS specifies trying the next alternative. If all alternatives return false, then an ALTS returns false. But if one returned true, then it returns true.

The order in which alternatives are listed is sometimes important. This is not good, but comes from choosing simple lexical analysis options. Notice that .EMP is always satisfied, so it must be last. Any alternatives beyond it will never be considered. If one keyword is contained in the left of another keyword (suppose you wanted to use REAL and REALTWO), then the longer one must be tested first. You will notice that all keywords in Meta4 start with '.'; this means they cannot be confused with identifiers so we don't have to put all keywords first. It is a good idea to put the most common alternatives first, and this gives the flexibility to do so.

A NOTSYN is an action not involving the input. In Meta II it consisted only of commands to generate commands

('.OUT') and of commands to generate labels ('.LAB'). A command to generate commands consists of the keyword '.OUT', followed by a '(' followed by zero or more AOUT's followed by a ')'. An AOUT is exactly the characters '*1' or '*2' in which case the first or second label numbers for the current subroutine will be concatenated to the output buffer (and generated if need be – see the discussion later), *or* the character '*' in which case the input buffer will be concatenated to the output buffer, *or* a singly quoted string which will be concatenated to the output buffer or a RVALUE which will be concatenated to the output as a decimal number, *or* the character '.' for which the current input line number will be concatenated to the output. After the ')' is found, the output buffer will be dumped to the output file. A label is generated by the keyword '.LAB' followed by exactly one AOUT.

The most important NOTSYN that I have added is '.ACT'. This generates instructions to the compiler generated to do something. The instructions need to be in the M4 language. The instructions in an ACT are for the M4 interpreter. The instructions in an OUT are for the target machine. In this example the target is M4, but usually it is easier to tell them apart. Since Meta II didn't have anything to do except output, it didn't need this, but Meta4 needs to check symbol tables, assign storage, decide between alternatives, etc. An ACT is the keyword '.ACT', the symbol '(', and zero or more of {[a singly or doubly quoted string (the instruction code) followed by '*1' or '*2' or nothing] or [a comment which is a string quoted by '!'] }. ACT essentially allows you to write compiler instructions in M4 machine code.

I've added two other kinds of output. '.CAT' is simply an OUT that does not dump the output buffer. It is useful for outputting part of a command, doing some more work, then finishing it. '.ERMS' is designed to allow putting error messages into a compiler so the users can know why they had an error. The basic error message that the M4 interpreter provides identifies the line and symbol number where it detected an error. In the context of Meta4, this is the line and symbol number of the first symbol in a statement that could not possibly be legal given the preceding symbols. This is quite useful, but any other help you can give yourself will be repaid.

The most frequent ACT is to transfer a number from one location to another. Meta4 makes these programmable in symbolic terms rather than having to use the machine instructions. The syntax for the transfers reflects the structure of M4 transfer commands with a requirement to first specify the value to transfer (RVALUE) as either a NAME identifier (previously defined by a declaration), or a CONSTSIMP or an EXPRESSION. If a NAME identifier is specified, it may also have appended to it ':' and a FIELD identifier. The value in the field of the symbolic memory indexed by the place named will be taken as the value. The transfer continues with the exact characters '=:' and the NAME of a place optionally followed by ':' and a field identifier, which will be used to store the fetched value.

EXPRESSION, CONSTEXP, and CEXP all use a layering kind of structure that is most easily understood by studying CONSTEXP. The four subroutines CONSTEXP, CONSTTERM, CONSTFAC, and CONSTSIMP should be examined together. Start at the top line of CONSTSIMP, which says that a number is a constant factor, and to put it on the stack. Now pass up to CONSTTERM. It says to look for a CONSTFAC (which can be satisfied by a CONSTSIMP), and then to look for an asterisk followed by another CONSTFAC. At this point two numbers are on the stack. The action is to pop (!) one of them and multiply (*) that times what is on the stack. This will be done as many times as there are asterisks in the input. Now look at CONSTEXP. That says to look for a CONSTTERM (which has any number of multiplications of CONSTFACTORS) and then to look for a plus or a minus. If either is found, another CONSTTERM will be sought and then added to or subtracted from the first. The result of this layering is that '*' binds more tightly than '+' or '–', so

a*b+c means (a*b)+c
not a*(b+c).

And what if you want the second form? Well, look again at CONSTFAC. Besides CONSTSIMPS, which can be previously defined constant identifier (stack its value), a single quoted string (stack its first character), a minus sign and any other CONSTFAC (take the negative of the stack top), CONSTFAC can be an open parenthesis and any CONSTEXP then a close parenthesis. The last alternative lets you put in nested parentheses to get your meaning.

One point about EXPRESSION is worth discussing. Notice that it must be introduced by a '+' sign. This is a little ugly, but necessary to get the generality of expression wanted. The problem is that '(' has already been used in SYN to indicate a new PHRASE. Without the introductory '+', if you tried to start an expression with (a+b), then you would start compiling as a SYN rather than as an EXPRESSION. (Or, perhaps it would clobber your SYN's, depending on the order.) This was not obvious to me until it happened, and it represents a distinct limitation on the kinds of languages Meta4 can compile.

Another main NOTSYN is an IF statement which allows actions built of NOTSYN's to be done based on a test. In order not to foul up the recursive descent parsing, only NOTSYN's are allowed in the selected portions. Also, to keep the flag in shape for the parsing, it is set true by an IF after using it to make the branch. The IF statement also refers to a CEXP, a conditional expression. CEXP, CTERM, and CFAC form another layered definition, this time of the condition expression. CFAC shows that the low level entities are two RVALUE's connected by one of six relational operators. These can be qualified by '.NOT', joined by '.ANDIF' and '.ORIF', and grouped by parentheses.

NOTSYN's useful together with an IF statement are '.ERROR', which deliberately flags an error, and '.RETURN', which returns from a subroutine before the end. A widely useful NOTSYN is to define a string quoted by '!' as causing no action, so it becomes a comment. The most common transfer is to the stack, for which I introduced a special symbol ']'. Finally, the symbol '&' followed by a new identifier will be compiled as a subroutine call. This is the same action as the first SYN, but does not generate a test. It should be used if and only if the subroutine consists entirely of NOTSYN, but there is no test for this.

Finally, let's go back to the DECLARE's. These allow the user to define an identifier as a FIELD, a NAME, or a CONSTANT. FIELD's are only 0-3 for Meta4, but can be increased if needed in other compilers. NAME's are limited to ten registers, and the stacks. CONSTANT's can be any constant expression. The identifiers for these can be any length up

to the buffer sizes used in the M4 interpreter (80 bytes).

## Using Meta4

Schorre apparently conceived of using Meta II to generate code for an interpreter, not for direct machine execution. At first I thought this was a lousy idea, because interpreters are slower than machine code. So the M4 H, L, and OX instructions let it put out machine code. But after experience with the M4 interpreter, and with other interpreters derived from it, I see advantages to interpreters.

The big advantage is ease of debugging. Even though it is not interactive, the M4 code can be read easily with an edit program. If not understood, output commands of the interpreter can be inserted, and the bug chased down. This debugging ease comes in part from using only readable and partly meaningful symbols for M4. Somewhat more efficiency in storage and speed could be gained by sacrificing it, but don't be tempted. A microcomputer has little enough help for debugging, on which most of your time is spent.

A second advantage is that it is usually easier to write a compiler for an interpreter, because the interpreter does more work per instruction. It is also quite reasonable to build an interpreter with more advanced features, such as stack arithmetic and integers to make compiler writing easier.

Finally, (do I protest too much?) an interpreter is a particularly easy form of program to debug once the basic program loading and command fetching are done. This is because the interpreter exercises a command if and only if you tell it to. This narrows the scope for bugs, and leads to finding them faster.

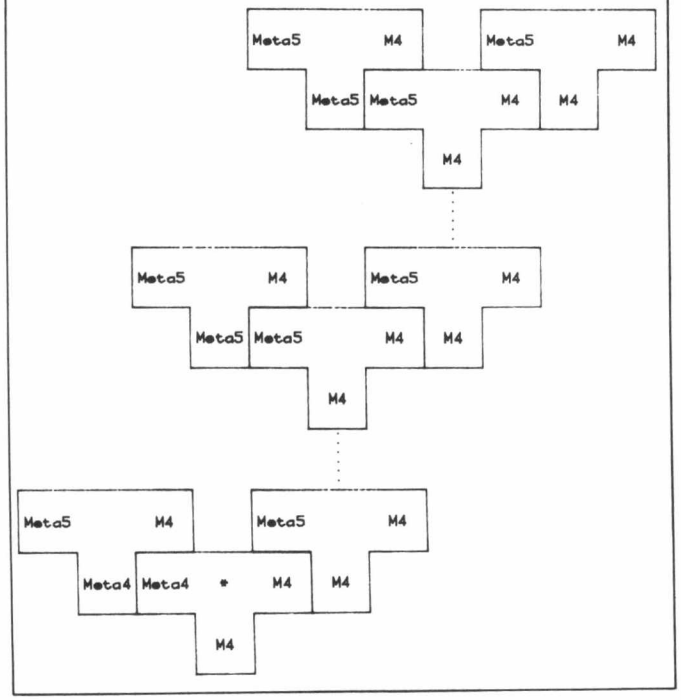
If you go the interpreter route, a second advantage to getting the M4 interpreter running is that it becomes a base you can modify to make other interpreters pointed toward your own needs.

Meta4 makes a powerful system for defining a language. One of the useful features is that a Meta4 program defines not only the syntax of a language (the rules for what are legal statements), but also its semantics (what these statements mean). Admittedly the syntax is laid out a lot more clearly than the kinky, as it grew, means for dealing with semantics. Also, it is quite possible, when designing a compiler for a given language to add new Meta rules by augmenting Meta4. For example, a sequence of SYN's separated by given punctuation could be defined. Two-level means of defining languages have been receiving theoretical attention recently (Cleveland and Uzgalis, *Grammars for Programming Languages*, is a less abstract introduction). They are, needless to say, conceptually much cleaner, but operationally less clear.

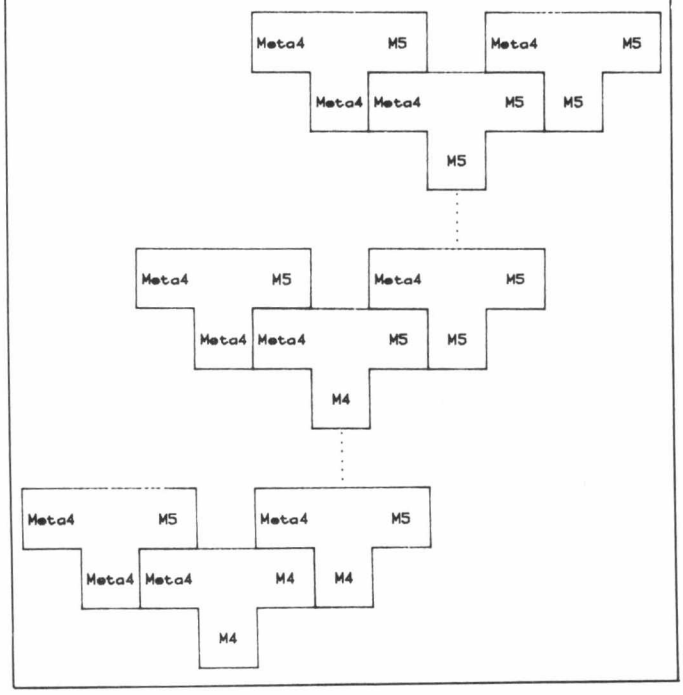
Using Meta4 in a two-level manner, it becomes a fairly common operation to bootstrap a larger compiler. Figure 3 shows the operation. It can be done in two steps, but it's wise to make a third to be sure that the compiler is self-compiling. Start reading the figure at the lower left. You write a Meta 5 to M4 compiler in the Meta4 language. You compile this as usual to a Meta 5 to M4 compiler written in M4. Using this compiler you can translate another program you write (but usually similar to the first, and sometimes the same) which is the Meta 5 to M4 compiler written in Meta 5. You now generate a different Meta 5 to M4 compiler written in M4. You can use this with the same input as the second step and you should get the same output. Since that doesn't usually happen the first time, it's wise to check it.

Figure 4 shows a similar bootstrapping process if you want to improve the M4 language to an M5 language. The same

**Bootstrap to another Meta Language**
**Figure 3**




**Bootstrap to Another Interpreter**
**Figure 4**



Meta4 to M5 compiler in Meta4 is input at all three stages, first producing a Meta4 to M5 compiler in M4, then a Meta4 to M5 compiler in M5. In the third step the coding language of the compiler running differs, but the source language and target language are the same, and the input is the same. Therefore the output should be the same. Check it.

Writing and initial debugging for a compiler written in Meta4 take about as long as other programs of comparable size in characters. But debugging the language – making sure that what you want to mean by a construct is what gets done – takes longer, because the compiler can only be tested by compiling and testing programs. Bugs in the compiled program may be due either to a bug in the program or in the compiler. I find that systematically exercising each feature in turn, by artificially short programs, flushes most of the bugs in a compiler, but a few will be left for your first programs to point out.

## The M4 Language

The three basic functions of lexical analysis, parsing, and output are handled very nearly the same in Meta4 as they are in Meta II. Table 1 gives the commands used for lexical analysis. The first five are very similar to Meta II, and each returns either "true" or "false" by setting or resetting a logical flag.

The lexical analysis commands are both powerful and restrictive: they each do a lot for one command, but only five kinds of strings are recognized. Schorre thought out the kinds very well, however, and a wide class of languages can be recognized. I added only the recognition of a hexadecimal number. One dead end I followed was to make commands for simpler primitives, like an alphabetic symbol. Using them slowed the interpreter down a lot for little gain in flexibility. The use of whitespace is particularly likely to mismatch a pre-specified language definition, but is easy to change by changing the interpreter. Newlines always terminate strings, while blanks or tabs will also terminate identifiers or numbers. Otherwise these three whitespace characters are ignored, and can and should be used to improve readability of a Meta4 program.

The purpose of the fifth command is to allow a slight amount of backing up. After giving an LI command, especially, if an identifier has been found, the symbol table can be consulted. If the identifier found is not of a specified type (such as that representing a constant), it can be rejected at that point and tested again at a later point. It makes writing a syntax easier. The Meta4 to M4 compiler listed below has some examples (see IDNEW).

The commands used to do the recursive descent parsing explained before are just the flow of control statements given in Table 2.

The third core function, largely taken from Meta II, is output. The M4 interpreter has an output buffer to which


# YOU CAN HELP YOUR COMMUNITY LEARN ABOUT COMPUTERS

**SUPPORT**

ComputerTown™

*ComputerTown*, a computer literacy project sponsored by People's Computer Company and later funded by the National Science Foundation, began operations in 1979. The project's ongoing goal is to design, publish and disseminate materials for the development of community-based computer literacy projects around the world.

**You too can help!** Your tax-deductible donations of new and used computer hardware and software will enable us to bring "hands-on" experience to people everywhere. These items can mean the difference between a person just hearing about computer technology and actually being able to sit down and use it. A *ComputerTown* can be set up in a public library, museum, school, business or a number of other places. What's needed, along with the information, resources and experience we provide, is just a few machines and some software to get started.

Already there are more than eighty *ComputerTowns* in the U.S., Canada, England and other countries. Your generous donations can help this number grow.

For more information about *ComputerTown*, and for details on how to make contributions, contact:

Fritzi Lareau, c/o *ComputerTown*
1263 El Camino Real, P.O. Box E, Menlo Park, CA 94025
**(415) 323-3111**

various strings are concatenated, and which is dumped to the output file on command. Table 3 describes the output options. Among them is the option to generate and use one or two labels per subroutine. The two are required, for instance, to build a loop construct, with one label at the top to repeat from, and one label at the bottom to exit to. It has turned out that only two labels has not been an impediment because it is two labels per subroutine call. Two labels are a marvelous encouragement to keep your subroutines small and intelligible, and to segregate concepts by level.

**Table 1**
**Lexical Commands**

| | |
|---|---|
| **LI** | An *Identifier* is a string starting with an alpha symbol and continuing with alpha or digits. The command returns false and does not change the input buffer if no identifier is in the input stream. Otherwise it returns true and moves the identifier to the input buffer. If the identifier is not in symbolic memory, it is entered. Register zero points to the memory cell. |
| **LN** | A *Number* is a sequence of digits. The command returns false if no digit is in the input stream. Otherwise it returns true and places the value of the number in register zero. |
| **LH** | A *Hex number* is a sequence of hexadecimal digits. The command returns false if no hexit is in the input stream. Otherwise it returns true and places the value of the number in register zero. |
| **LQchar** | A *Quoted string* is composed of a quoting character, any string of characters except newline or the quoting character, and a closing quoting character or newline. The command returns false if the input stream does not contain a string quoted by the character given. Otherwise it returns true and places the string, without its quotes, in the input buffer. |
| **LMchar . . . char** | A *Literal string* is any string of characters except a newline. The command returns true if and only if the input stream matches the literal string. The input buffer is not changed. |
| **L** | If the flag is set, the symbol found in the input stream is accepted. Otherwise, it is rejected and becomes the next string of characters in the input stream again. This modest 1 symbol backup makes different types of identifiers feasible. |

**Table 2**
**Control Commands**

| | |
|---|---|
| **S** | Set the flag true. |
| **SF** | Set the flag false. |
| **SC** | Set the flag changed (true if it was false, and vice versa). |
| **.number**<br>**.identifier** | Identify the next operational command as the place associated with this number or identifier. |
| **J/number** | Jump: Transfer control to the place associated with the number. |
| **F/number** | False jump: If the flag is false, transfer control to the place associated with the number, otherwise continue to the next command. |
| **T/number** | True jump: If the flag is true, transfer control to the place associated with the number, otherwise, continue to the next command. |
| **Gidentifier** | Go to a subroutine. Stack the next command location, and two zero label numbers, then transfer control to the place associated with the identifier. |
| **R** | Return from subroutine. Unstack two label numbers and a command location. Transfer control to the command location unstacked. |
| **E** | End. Stop processing. |

M4 has several ways to store numbers. The simplest are ten registers, each identified by a digit. The 'Y' stack allows stacking a number, reading its top entry, or unstacking a number. The 'Z' stack allows stacking or unstacking a number. Memory 'cells', described soon, can also store numbers. Table 4 shows the direct Fetches that are possible in M4.

Why a stack? Meta II can only compile code for a stack machine. By giving M4 a stack, operations can be stacked by the compiler, then output in the required order for a register machine. The second stack allows putting loop labels and if labels on different stacks, so that 'break' and 'continue' kinds of statements can be compiled. If the operations get hairy, with types for each variable, go to the stack of memory cells (Table 5).

The symbolic memory stores identifiers together with a memory *cell*. The cell consists of N=1 to 5 integers (known as 0 to N-1) and N byte (known as N to 2N-1) storage locations.

The symbolic memory is layered with all entries made in the top layer. The 'LI' command automatically searches the symbolic memory from the top layer down. If identifiers with the same name are in different levels, the top one will be found. This structure will allow defining variables with a local scope. Table 5 gives the commands for manipulating memory.

The last two commands use the space at the top of the symbolic memory vectors to contain a stack of memory cells.

The contents of a memory cell are fetched by an indirect Fetch. Indirect Fetches also allow getting the contents of the input buffer, and the registers. They are summarized in Table 6.

The direct and indirect fetches can be used for output with 'C' and 'L', and 'H' as shown in Table 3. There are several other disposal options shown in Table 7. The general form of a number manipulation is

[Fetch] [dispose]

while a fetch can be either

[directFetch]

or

[directFetch] [directFetch] [indirectFetch]

A final group of commands are useful for doing something about errors. Table 8 shows them. The 'XM' command essentially defines a place in the syntax to return to after finding an error. The 'XO' command is of some use for remarking which error was made, but of much greater use, I find, for debugging.

**Table 3**
**Normal Output Options**

| | |
|---|---|
| **O** | Out: Dump the output buffer to the output file with a newline, and reset it to the null string. |
| **OX** | Same as Out except no newline written. |
| **P specified string** | Concatenate the specified string to the output buffer. |
| **C** | Concatenate the input buffer to the output buffer. |
| **Fetch C** | (See Tables 4 and 6 for Fetches.) Convert the number fetched to a decimal string and concatenate to the output buffer. |
| **Fetch L** | Concatenate the low byte of the fetched to the output buffer. |
| **Fetch H** | Concatenate the low byte of the fetched to the output buffer as two hexadecimal digits. |
| **U** | If the first label number stacked is zero, substitute the symbol generator and then increment the symbol generator. Convert the first label number stacked to a digital string and concatenate to the output buffer. Also store the number in the zero register. |
| **V** | If the second label number stacked is zero, substitute the symbol generator and then increment the symbol generator. Convert the second label number stacked to a digital string and concatenate to the output buffer. Also store the number in the zero register. |

It is easy to use an editor to insert output commands and an 'XO' command to trace the operation of a troublesome program so as to see what is going wrong. These are pretty crude aids by modern standards, but they let you skip over a bug you don't understand and work on others, because it will find a bug per statement.

## Implementing Meta4

The listing for the M4 interpreter in Pidgin gives detailed

**Table 4**
**Direct Fetches**

| | |
|---|---|
| **Y** | Fetch the number stacked on the Y stack, without popping. |
| **!** | Fetch the number stacked on the Y stack, and pop. |
| **Z** | Fetch the number stacked on the Z stack, and pop. |
| **H** | Fetch the high byte of the Y stack, without popping. |
| **Digit** | Fetch the contents of the register indexed. |
| **N/number** | Fetch the number specified. |
| **B** | Fetch the breadth of the input string. |
| **U** | Fetch the first number label without altering. |
| **V** | Fetch the second number label without altering. |

**Table 5**
**Memory Commands**

| | |
|---|---|
| **MI digit** | Initialize memory, and set cell size to the digit (1-5); default size is 2. |
| **ME** | Enter the input buffer into the top layer of memory. Return the index pointing to the memory cell to register zero, and zero the memory cell. |
| **MS** | Stack memory – seal off the previous top layer and open a new layer. |
| **MP** | Pop memory – delete all entries in current top layer and reopen the next lower layer. |
| **MC** | Create a memory cell in the cell stack. Return the index to register zero, and zero the cell. |
| **MD** | Destroy the top memory cell on the cell stack. That is, pop this stack. |
| **MQ** | Query the entire memory using the input buffer. Return an index to the memory cell found (or zero) to register to zero. |

### Table 6
### Indirect Fetches

| | |
|---|---|
| **M digit** | If digit is not in the range of memory cell indexes (0 to 2 N-1), use zero. Fetch the contents of the digit'th storage location of the cell indexed by the direct Fetch. |
| **S** | If the direct Fetch is less than the input string length, fetch the direct'th character from the input buffer. Default to the zeroth character. |

### Table 7
### Non-Output Dispose Options

| | |
|---|---|
| **Y** | Stack the Fetched on the Y stack. |
| **Z** | Stack the Fetched on the Z stack. |
| **+** | Pop the Y stack, add the result to the fetched, and stack the result on Y. |
| **-** | Pop the Y stack, subtract the fetched from the popped, and stack the result on Y. |
| ***** | Pop the Y stack, multiply the result times the fetched, and stack the result on Y. |
| **>** | Pop the Y stack, set the flag if the fetched was greater than the popped, else reset the flag. |
| **<** | Pop the Y stack, set the flag if the fetched was less than the popped, else reset the flag. |
| **=** | Pop the Y stack, set the flag if the fetched was equal to the popped, else reset the flag. |
| **digit** | Store the fetched in the digit'th register. |
| **I[loc][direct]** | Store the fetched indirect to memory, in the loc'th location of the memory cell indexed by the result of this direct fetch. |
| **A** | Append low byte of fetched to input register. |
| **B** | Set length of string in input register (especially shorter) from the fetched. |
| **G** | Set the symbol generator from the fetched. |
| **U** | Store the fetched as the first label number. |
| **V** | Store the fetched as the second label number. |
| **D** | Dump the fetched – do nothing. |

### Table 8
### Error Handling Commands

| | |
|---|---|
| **XM character** | Save the current code position, stack depth and the mask character. |
| **X** | If flag is true, do nothing. If false print the line and symbol number; discard characters from the input until find the mask character or end of file. Reset the code pointer and stack depth to saved values. |
| **XN** | Copy the line number to the output buffer. |
| **XO** | Dump the output buffer to the terminal. |


**You Read Dr. Dobb's Journal And You Don't Subscribe?!**

**Save $13 off newsstand prices for 2 yrs. Save $5 for 1 yr.**

Can you afford to miss an issue with information vital to your interests? As a subscriber you can look forward to articles on Small-C, FORTH, CP/M, S-100, Compiler optimization, Concurrent Programming and more, delivered right to your door. And you'll never miss the issue that covers your project.

**DR. DOBB'S JOURNAL**
**P.O. Box E**
**Menlo Park, CA 94025**

**Yes!** Sign me up for ___2 yrs. $47 ___1 yr. $25

___ I enclose a check/money order
___ Charge my Visa, MasterCard, American Express
___ Please bill me later

Name _____
Address _____
_____
Credit Card _____ Exp. date _____
Acct No. _____
Signature _____

algorithms for implementing an M4 interpreter. You also need the Meta4 to M4 compiler in M4 to get started, once the M4 interpreter is debugged. A good test of your M4 interpreter is to recompile it from the Meta4 listing.

The M4 interpreter compiles into 9056 program bytes of 6502 code using the compiler described last month. It also uses memory for label index table, interpretive code, subroutine stack, and symbolic memory. I have set aside 2K, 10K, 1K, and 9K bytes respecitvely for these. To compile the Meta4 syntax only takes 1/2K, 3K, 1/10K, and 1/2K of this working space. It takes my Apple II about 60 seconds for this compilation.

The following brief description of the algorithms and methods of the M4 interpreter may also help in implementing. And don't overlook the comments within the program.

The first phase of the M4 interpreter is to initialize constants, etc., then to load and compact its program. The symbolic memory is used in this phase to compile the symbolic subroutine calls to show the location in the interpretive code referenced. Numbers preceded by a '/' are compiled into two-byte binary representations from a decimal representation. Number labels are entered in the numeric label table (they could also be compiled at this time and save the table space at a cost of more program to compile). In the interpretive code space each instruction is represented by a byte showing how many other bytes follow, then by the bytes of command, as compacted. This phase is accomplished about as fast as Apple can read the disk. At the end of this phase, the M4 interpreter notifies the user of how many instruction bytes were used, how many labels, and how many subroutines. For each subroutine call made to a subroutine never defined, a message is output. The name is not easily reconstructed at this point, but the last of the characters printed are the last characters of the name of subroutine called. Since the error is not necessarily fatal, the program continues. But it's usually a bad sign.

The main body of the M4 interpreter is a loop that fetches the next instruction, then chooses a sequence of actions as described before from the command characters. They are ordered so the most frequently used in the Meta4 to M4 compiler are first. When it is through it prints the maximum amount of symbolic memory that was used.

The registers and stack are integer vectors. Each stack operation is checked for overflowing or underflowing the space allocated. If you change a dimension statement, be sure to change the corresponding initialization statement.

The symbolic memory is constructed for digital searching (Knuth, *Art of Computer Programming*, Vol. III sorting and search, section 6.3). Knuth shows that this requires on average only $\log_2 N$ search steps if the input data is random, so it is usually an efficient system. It is also easy to program. Two vectors are used to make a binary tree holding a level of the symbol table. One vector holds characters, the other integers which index an alternate place in the tree. In searching, the first character of memory is compared to the first character of the symbol. When a match is found, if it is a match on the end of string character (zero), the memory cell for the symbol begins at the next index. If a pair of characters matches, but is not the end of string character, the next character in memory will be compared to the next character in the symbol. Whenever the character in memory does not match, but there is another alternative (the alternative index is not zero), the current character of the symbol will be compared to the alternative character in memory. If there is no alternative, then the search has failed – the symbol has not been entered into that level of memory.

Table 9 shows the representation from entering HAD, HAVE, HE, HER, in that order, and reserving memory cells of size 2.

## Availability in Machine Readable Form

With the listings of the M4 interpreter and Meta4 provided here, you can get Meta4 working. If you want to save yourself the typing, and can transfer files from an Apple near you, you may be interested in the following offer. A version of the M4 interpreter in an experimental operating system is available from the author. It includes the listings here, and machine language versions of the M4 interpreter, the Tincmp compiler, and an editor. It will run on an Apple, configured for 1 to 4 floppy disk drives. It requires a 16K memory card or an Apple II (not plus). Two disks and documentation are available for $20. A third disk with Pidgin listings for the operating system programs is available for $10. If you are reading this after 1981, better write to ask about availability. DDJ

**Table 9**
**Representation in Symbolic Memory**

| Index | Choice | Alt. | Notes |
|---|---|---|---|
| 1 | H | 0 | No alt to H |
| 2 | A | 12 | Alt to 'A' is 'E' starting at 12 |
| 3 | D | 7 | Alt to 'D' is 'V' starting at 7 |
| 4 | 0 | 0 | end of word HAD |
| 5 | M2 | M0 | Memory cell for HAD |
| 6 | M3 | M1 | |
| 7 | V | 0 | |
| 8 | E | 0 | |
| 9 | 0 | 0 | |
| 10 | M2 | M0 | Memory cell for HAVE |
| 11 | M3 | M1 | |
| 12 | E | 0 | No alt to 'HE' |
| 13 | 0 | 16 | But there is an alt to 'HE[end]' |
| 14 | M2 | M0 | Memory cell for HE |
| 15 | M3 | M1 | |
| 16 | R | 0 | |
| 17 | 0 | 0 | |
| 18 | M2 | M0 | Memory cell for HER |
| 19 | M3 | M1 | |
| 20 | Free memory starts | | |

# Some Useful C Time Functions

by Bill Bolton

*Bill Bolton, Software Tools, P.O. Box 80, Newport Beach, NSW, 2106, Australia.*

Rather than show why you *should* use a programming language (in this case C), this article endeavours to show why I *have* used it. An example program demonstrates some aspects of the language and is directly useful to owners of the Mountain Hardware 100,000 day clock board, as well as being generally useful to others with real time clock facilities in their systems. A basic knowledge of C will be required to understand the example program; it is structured and commented, and brief extra descriptions of the functions are given.

## Why C?

While I have admired the C language from afar for a long time, only relatively recently have I discovered its true delights. Because Pascal was supposed to be *the* language for structured programming and there was a need for more structure in my programming efforts, I had given Pascal a good try. The system utilities which I was interested in writing just would not come together properly under Pascal; I had the distinct feeling the language was fighting me rather than helping my programming efforts. My problems with Pascal were not limited to one implementation. Pascal/M, Pascal MT+, Pascal/Z and UCSD Pascal all presented similar generic problems. Before the Pascal phreaks rise up again, let me stress that there are a lot of things about Pascal which seem very worthwhile, but I couldn't get it to do the things that I need done, in a way I could grasp.

So for a time I retreated to assembler and FORTRAN for the serious stuff and to BASIC for the quickies. I found it was possible to write reasonably structured programs in BASIC-80[1] and FORTRAN 80 if a moderate amount of care was taken, but assembler was still the mainstay. A brief flirtation with FORTH followed. FORTH showed great promise but the temptation to produce code which even I couldn't understand a week later was too great. Still, the idea of an extensible language is fascinating!

In the meantime, I had been reading "The C Programming Language"[2] and it was clear that C was a language which was structured but not too verbose or inflexible (my major problems with Pascal). It also appeared to have an extensible aspect in that the overhead in writing, maintaining and using a library of functions was quite small.

At about this time, a major software project of the AWA Data Group was started in C. I was able to look over the programmer's shoulder as the project progressed and I liked what I saw. What's more, when the project finished the programmers involved still seemed sane. I had a chance to see a working C compiler (Whitesmith's for the PDP-11) in action and to study the documentation. While the Whitesmith compiler was out of my financial range for starters, Leor Zolman had his BDS C compiler available at a very reasonable price (on reflection I'd say it's an absolute steal), so that was my choice.

As soon as the compiler arrived, I was thrown into the situation that forced me to jump in at the deep end and use it for a commercial project. I had quoted a client four hours to write a file conversion program in assembler. It soon became obvious that I had made a major error with my time prediction, so I decided to try and salvage the situation by writing the program in C. This first programming effort in C went so smoothly that the project was finished within the budgeted time and the client got a product with error handling an order of magnitude better than he would have received from an assembler program. Since then I have abandoned BASIC almost entirely, only use FORTRAN for maintenance, and have greatly reduced the amount of assembler programming. The description of C as "a Pascal that's not afraid to get its hands dirty"[3] is very, very apt.

If there is one single thing that delayed my entry in the world of C it is probably the sparcity of references to C in the microcomputer journals. There have been only a handful of programs published and often they are examples that don't really do anything useful (Small C in an obvious exception). Following is one small effort to correct this situation.

## The Program

DATEDEMO is a collection of C functions which will format a string with the current date and current time fetched from a Mountain Hardware 100,000 day clock S-100 board. Several date and time formats are available by specification as one of the arguments to the primary functions.

The date formats are those used by the nearly all "Western" countries outside North America (this is an appropriate point to complain loudly about software packages that lock the user into MM/DD/YY date formats!!!) but are coded in a very general manner so that nearly any imaginable date format could be easily generated and extra formats are easily added. Time formats are somewhat arbitrary and only support 24-hour types (perhaps if someone is keen they will do an AM/PM 12-hour type and contribute it to *DDJ*).

In practice I use the functions to put date and time in the page headings of hardcopy output and in some business packages. The functions live in my standard C function library and are linked in at compile time. Normally the "#define" statements are loaded from a header file CLOCK.H but they are listed in full here. While the program is written for compilation with the BDS C compiler (Version 1.4X), it should also compile with the other C compilers. The only non-standard library function is 'initw' which is the BDS kludge to allow array initialization, as BDS C does not support initializers. As far as I know, all other library functions are supported by other C compilers (maybe not Small C).

## The Functions

**'main'** Is simply an executable demonstration program to tie the actual working functions together, all the presently supported formats are displayed and a typical use of the formatted strings is shown on the last line displayed.

**'date'** Attempts to get basic data to format a data string form the clock; if this fails an error message is displayed, else the month and week name are determined and the string formatted.

**'get_date'** Tests if the clock board is in the system. Then col-

lects digits from the clock to get the number of days since 31/Dec/1977 (N.B. this day was a Saturday, so a simple MOD 7 on the total number of days since then makes Sunday day 1). This value is then turned into the year, month and day of the month.

**'month_day'** Uses an initialized array to determine month and day of the month from year and day of the year.

**'ndays'** Returns number of days in the specified year.

**'leap'** Thorough test for leap years.

**'name_month'** Fills a string with a month name given the number of the month in range 1 to 12.

**'name_week'** Fills a string with day of the week name given the number of the day in range 0 to 6.

**'time'** As for 'date' except that time needs much less processing.

**'read_clock'** Tests if clock is in system (this test must be duplicated from 'get_date' as the date and time functions may be used separately), then fetches digits from the clock and puts them straight into an integer array.

## The Hardware

The MH board occupies a block of 16 I/O ports. The values returned from each port are shown at the head of the listing. Each value is in the form of a BCD number and for the purposes of this program the high nybble of the port can be ignored. A simple test for presence of the board in the bus can be made by looking at the base port. An FF hex will indicate that the board is missing. An onboard battery keeps the clock running all the time, so I only need to reset time for Daylight saving and usually do it using DDT's inline assembler. The board has been super reliable and functions well in a system set up to IEEE 696 standards.

The rest of the system is a North Star Horizon mainframe with a Morrow Designs "Wunderbuss" 12 slot motherboard (Yes it *is* possible to get another motherboard into an Horizon, and attacking your computer with a hammer can be a very therapeutic experience, but that is a story for another time), Godbout Z-80 CPU, Morrow SuperRam memory, Morrow Disk Jockey 2D controller and Godbout Interfacer I Serial I/O.

## Program Availability

This program will be available from the BDS C User's Group, 409 E. Kansas, Yates Centre, KS 66783. Membership in the group is $10.00 a year (US$20.00 a year foreign) and numerous disks of C software are available from the group for nominal copying charges. Among the software available is Ron Cain's Small-C Compiler and Run Time Library previously published in *DDJ*, and W. C. Colley's 6800 & 1802 cross-assemblers previously described in *DDJ*. DDJ

References:


[1] See CPMLABLE.BAS on CP/M User's Group Disk 46, one of my efforts at structure in BASIC 80.

[2] "The C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie, published by Prentice-Hall 1978, ISBN 0-13-110163-3. This is the "bible" of C and essential to any study of the language.

[3] From "C Notes" by C. T. Zahn, published by Yourdon Press 1979, ISBN 0-917072-13-8. The only other readily available work on C; interesting as it was written by someone outside Bell Labs and who has a slightly different perspective on the language.


## The Author


*Bill Bolton is employed as Senior Technical Officer at Multicultural Television Channel 0/28, involved in the maintenance of both mini and microcomputer systems as well as video systems. BASIC and FORTRAN were learned as part of a college course in 1975 and that was where he became interested in computers. Present activities include consulting work for CP/M system users, coordinating 80AT (the Australian 8080/Z80 User's Group) and writing a regular CP/M column in a local microcomputer journal "Your Computer."*


### C Functions Demo

```
A>DATEDEMO

Demonstration of 'C' functions for the Mountain Hardware 100,000 day clock.


        By Bill Bolton, Software Tools
        P.O. Box 80,
        Newport Beach,
        NSW, 2106, AUSTRALIA


Date format 0 is May 20, 1981

Date format 1 is Wednesday, May 20, 1981

Date format 2 is 20/5/1981

Date format 3 is Wednesday, 20/5/1981

Date format argument ERROR !

Time of day format 0 = 0940:22

Time of day format 1 = 09:40:22

Time of day format 2 = 09:40

Time of day format argument ERROR !

Printed at 0940:23 Hours on Wednesday, May 20, 1981 for Dr.Dobbs Journal.

A>
```

# C Time Functions (Text on pages 16-17)

```
/*
#include        CLOCK.H         /* MH clock constants */
*/


 #define CLOCK    32             /* Base port for Mountain Hardware clock */
 #define USEC100 CLOCK           /* Hundreds of microseconds digit port */
 #define MSEC1    CLOCK+1        /* Unit milliseconds digit port */
 #define MSEC10   CLOCK+2        /* Tens of milliseconds digit port */
 #define MSEC100 CLOCK+3         /* Hundreds of milliseconds digit port */
 #define SEC1     CLOCK+4        /* Unit seconds digit port */
 #define SEC10    CLOCK+5        /* Tens of seconds digit port */
 #define MIN1     CLOCK+6        /* Unit miniutes digit port */
 #define MIN10    CLOCK+7        /* Tens of minutes digit port */
 #define HOUR1    CLOCK+8        /* Unit hours digit port */
 #define HOUR10   CLOCK+9        /* Tens of hours digit port */
 #define DAY1     CLOCK+10       /* Unit days digit port */
 #define DAY10    CLOCK+11       /* Tens of days digit port */
 #define DAY100   CLOCK+12       /* Hundreds of days digit port */
 #define DAY1K    CLOCK+13       /* Thousands of days digit port */
 #define DAY10K   CLOCK+14       /* Ten of thousands of days digit port */
 #define CINTER   CLOCK+15       /* Interupt set port */
 #define MASK     0X000F         /* Digit bit mask, lower 4 bits BCD digit */


#define BASE_YEAR       1978


/* Example program for Mountain Hardware "100,000 day clock" S-100 board.
   This main program simply calls the date function and time function,
   displaying the returned strings. */

main()


{
        char    date_str[35];   /* string (character array) for date */
        char    time_str[8];    /* string for time */



       if (date(date_str,0)){
               printf("Exiting to CP/M\07\n");
               exit();



       }
       printf("Demonstration of 'C' functions for the ");
       printf("Mountain Hardware 100,000 day clock.\n\n");
       printf("\tBy Bill Bolton, Software Tools\n");
       printf("\tP.O. Box 80,\n\tNewport Beach,\n");
       printf("\tNSW, 2106, AUSTRALIA\n\n");
       printf("\Date format 0 is %s\n\n",date_str);
       date(date_str,1);
       printf("Date format 1 is %s\n\n",date_str);
       date(date_str,2);
       printf("Date format 2 is %s\n\n",date_str);
       date(date_str,3);
       printf("Date format 3 is %s\n\n",date_str);
        name_week(wname,wday);
        switch(format){
        case 0:


                sprintf(str,"%s %d, %d",mname,*mday,*year);
                return(0);


        case 1:


                sprintf(str,"%s, %s %d, %d",wname,mname,*mday,*year);
                return(0);


        case 2:


                sprintf(str,"%d/%d/%d",*mday,*month,*year);
                return(0);


        case 3:


                sprintf(str,"%s, %d/%d/%d",wname,*mday,*month,*year);
                return(0);


        default:


                printf("Date format argument ERROR !\07\n\n");
                return(-1);
        }
}



/* get_date(year,month,mday,wday) provides the basic data for formatting
   a date string, fetched from the clock board and converted to a useable
   set of values


*/

get_date(year,month,mday,wday)


     int     *year;          /* pointer to current year */
     int     *month;         /* pointer to current month */
     int     *mday;          /* pointer to day of the month */
     int     *wday;          /* pointer to day of the week */

     {
```

```
        date(date_str,4);
        time(time_str,0);
        printf("Time of day format 0 = %s\n\n",time_str);
        time(time_str,1);
        printf("Time of day format 1 = %s\n\n",time_str);
        time(time_str,2);
        printf("Time of day format 2 = %s\n\n",time_str);
        time(time_str,3);
        date(date_str,1);
        time(time_str,0);
        printf("Printed at %s Hours on %s for Dr.Dobbs Journal.\n\n",
                        time_str,date_str);


 }

/* date(srt,format) will fill a string with date formatted as follows:



        format = 0      "May 11, 1981"
        format = 1      "Monday, May 11, 1981"
        format = 2      "11/5/1981"
        format = 3      "Monday, 11/5/1981"
*/


date(str,format)

char    *str;           /* pointer to date string */
int     format;         /* format identifier */



{
        char    wname[10];      /* string for week day proper name */

        char    mname[12];      /* string for month proper name */

        int     year[1];        /* year, range 1978 to ????? */

        int     month[1];       /* month of the year, range 1 to 12 */

        int     mday[1];        /* day of the current month, range 0 to 31 */

        int     wday[1];        /* day of the week, range 0 to 6 */




        if (get_date(year,month,mday,wday)){
                printf("No clock board present in system\07\n");
                return(-1);



        }
        name_month(mname,month);
```

*(Continued on top of page 18)*

```
        int     tdays;

        int     port;   /* input port number */

        int     pos;    /* digit decimal position */

        int     tyear;


        tdays = 0;



          if (inp(CLOCK) == 0XFF )        /* no clock board present */
                return(-1);


          for (port = DAY1, pos = 1; port <= DAY10K; port++, pos *= 10){
                tdays += (inp(port) & MASK) * pos;
          }


          *wday = tdays%7;
          for (*year = BASE_YEAR; (tyear = tdays - ndays(year)) > 0;(*year)++){
                tdays = tyear;
          }



          month_day(year,tdays,month,mday);
          return (0);
}

/* month_day(year,yday,month,mday) is straight out of Kernighan and Ritchie
   page 104, except that BDS C doesn't support intialisers so the special
   BDS C "intw" kludge is used to initialise the day_tab array. Also the
   leap year cal culation has been split off into a separate function as
   it is generally useful.
*/



month_day(year,yday,month,mday)


int     *year;          /* pointer to current year */

int     yday;           /* day of the year, range 1 to 366 */

int     *month;         /* pointer to month of the year */

int     *mday;          /* pointer to day of the month */
{


        int     i;
        int     lyear;
        int     day_tab[2][13];
```

*(Continued on next page)*

# C Time Functions (Listing continued, text on pages 16-17)

```
        initw(day_tab[0],"0,31,28,31,30,31,30,31,31,30,31,30,31");

        initw(day_tab[1],"0,31,29,31,30,31,30,31,31,30,31,30,31");

        lyear = leap(*year);


        for (i = 1; yday > day_tab[lyear][i]; i++){
                yday -= day_tab[lyear][i];
        }


        *month = i;


        *mday = yday;
}



/* ndays(year) returns the number of days in the current year.
*/

ndays(year)

int     *year;          /* pointer to current year */



{
        return(leap(*year) ?  366 : 365);
}


/* leap(year) returns a flag to indicate if current year is a leap year.
*/


leap(year)

int     year;           /* current year */

{
        return (year%4 == 0 && year%100 != 0 || year%400 == 0);
}

/* name_month(mname,month) fills a string with the name of the the
   current month.
*/

name_month(mname,month)

char    *mname;         /* pointer to month name string */
int     *month;         /* pointer to current month */


{
        switch(*month){
        case 1:
                strcpy(mname,"January");
                return(0);
        case 2:
                strcpy(mname,"February");
                return(0);
        case 3:
                strcpy(mname,"March");
                return(0);
        case 4:
                strcpy(mname,"April");
                return(0);
        case 5:
                strcpy(mname,"May");
                return(0);
        case 6:
                strcpy(mname,"June");
                return(0);
        case 7:
                strcpy(mname,"July");
                return(0);
        case 8:
                strcpy(mname,"August");
                return(0);
        case 9:
                strcpy(mname,"September");
                return(0);
        case 10:
                strcpy(mname,"October");
                return(0);
        case 11:
                strcpy(mname,"November");
                return(0);
        case 12:
                strcpy(mname,"December");
                return(0);
        default:
                printf("Month name ERROR !\n");
                return(-1);
        }
}

/* name_week(wname,wday) fills a string with the name of the current
   week day.
*/

name_week(wname,wday)

char    *wname;         /* pointer to week name string */
int     *wday;          /* pointer to current week day */

{
        switch(*wday){
        case 1:
                strcpy(wname,"Sunday");
                return(0);
        case 2:
                strcpy(wname,"Monday");
                return(0);
        case 3:
                strcpy(wname,"Tuesday");
                return(0);
        case 4:
                strcpy(wname,"Wednesday");
                return(0);
        case 5:
                strcpy(wname,"Thursday");
                return(0);
        case 6:
                strcpy(wname,"Friday");

                return(0);
        case 0:
                strcpy(wname,"Saturday");
                return(0);
        default:
                printf("Weekday name ERROR !\n");
                return(-1);
        }
}

/* time(str,format) fills a string with the time of day in the
   following formats :

        format 0        1800:15
        format 1        18:00:15
        format 2        18:00
*/

time(str,format)

char    *str;           /* string to fill with time */
int     format;         /* flag for format of string */

{
        int     t[6];

        if (read_clock(t)){
                printf("No clock board present in system !\07\n");
                return(-1);
```

```
        strcpy(mname,"January");
        return(0);
case 2:

        strcpy(mname,"February");
        return(0);

case 3:

        strcpy(mname,"March");
        return(0);

case 4:

        strcpy(mname,"April");
        return(0);

case 5:

        strcpy(mname,"May");
        return(0);

case 6:

        strcpy(mname,"June");
        return(0);

case 7:

        strcpy(mname,"July");
        return(0);

case 8:

        strcpy(mname,"August");
        return(0);

case 9:

        strcpy(mname,"September");
        return(0);

case 10:

        strcpy(mname,"October");
        return(0);

case 11:

        strcpy(mname,"November");
        return(0);
```

*(Continued on top of page 20)*

```
        }
        switch(format){
        case 0:
                sprintf(str,"%d%d%d%d:%d%d",t[0],t[1],t[2],t[3],t[4],t[5]);
                return(0);
        case 1:
                sprintf(str,"%d%d:%d%d:%d%d",t[0],t[1],t[2],t[3],t[4],t[5]);
                return(0);
        case 2:
                sprintf(str,"%d%d:%d%d",t[0],t[1],t[2],t[3]);
                return(0);
        default:
                printf("Time of day format argument ERROR !\07\n\n");
                return(-1);
        }
}

/* read_clock(t) fills an array with the time of day digits read from
   the clock board
*/

read_clock(t)

int     *t;             /* array to store clock digits */

{
        int     port;   /* clock port to read */
        int     ptr;    /* pointer into digit array */

        if (inp(CLOCK) == 0XFF )        /* no clock board present */
                return(-1);
        for (port = HOUR10, ptr = 0; port >= SEC1; port--, ptr++)
                t[ptr] = inp(port) & MASK;
        return(0);
}

END OF SOURCE LISTING


A>CC1 DATEDEMO.C

BD Software C Compiler v1.43a (part I)
  28K elbowroom
BD Software C Compiler v1.43 (part II)
  25K to spare

A>CLINK DATEDEMO -s

BD Software C Linker   v1.43

1ABD ATOI       0A87 DATE       1BAC EXIT       18DF GETVAL
0DC7 GET_DATE   14F4 INITW      1BAF INP        19F3 ISDIGIT
1B7D ISLOWER    141D LEAP       079E MAIN       12FF MONTH_DA
0EE7 NAME_MON   10F0 NAME_WEE   12D2 NDAYS      145F PRINTF
1BBC PUTCHAR    18AD PUTS       1243 READ_CLO   148D SPRINTF
14AC STRCPY     0C4A TIME       1A85 TOUPPER    1A22 _GV2
153E _SPR       1949 _USPR

Last code address: 1BF1
Externals start at 1BF2, occupy 0000 bytes, last byte at 1BF2
Top of memory: D1FF
Stack space: B60E
Linkage complete
  38K left over
```

**End Listing**

# Write Your Own Compiler (Text on pages 6-14)

## Listing I

```
.SYN COMPILE
.FIELD VALUE Ø TYPE 3 ;
.CONSTANT NAME 1 FIELD 3 CONSTANT 2 UNDEFINED Ø
   AVAILFIELDS 5
   ;
.NAME NUMBER 'Ø' ID 'Ø' SAVID '2' TEMP 'Z'
   STACK 'Y' UNSTACK '!'
   ;
RVALUE =
    IDNAME &CATNAME (':' IDFIELD .CAT ('M' ID:VALUE)
          / .EMP )
   /CONSTSIMP
          .CAT('N/' UNSTACK ' ')
   / '+' EXPRESSION .CAT('!')
   ;
EXPRESSION = TERM $( '+'TERM .OUT('!+')
      /'-' TERM .OUT('!-')
      )
   ;
TERM = FACTOR
   $('*' FACTOR .OUT('!*')
   /'/''256' .OUT('HZ')
      .OUT('!D')
      .OUT('ZY')
   /'%256'
      .OUT('HY')
      .OUT('N/''256*')
      .OUT('!-')
   )
   ;
FACTOR='(' EXPRESSION ')'
   /CONSTSIMP .OUT('N/' UNSTACK 'Y')
   /IDNAME &CATNAME (':' IDFIELD
            .CAT('M'ID:VALUE)
            /.EMP) .OUT('Y')
   /'-'FACTOR .OUT('!Z')
         .OUT('N/''ØY')
         .OUT('Z-')
   ;
CATNAME = ID:VALUE=:STACK
   .ACT('!L')    !ACTS AS A ONE BYTE MACRO
   ;
TOPLACE = IDNAME ID=:SAVID
      (':' IDFIELD .CAT( 'I'  ID:VALUE)
      /.EMP )
         SAVID=:ID &CATNAME
   ;
DECLARE = '.FIELD' $ ( IDNEW ID=:SAVID
         CONSTEXP  &CHECKFRANGE NUMBER=:SAVID:VALUE
               FIELD=:SAVID:TYPE
         )          ';'
   / '.NAME' $ ( IDNEW ID=:SAVID
            CONSTEXP
            UNSTACK=:SAVID:VALUE
            NAME=:SAVID:TYPE
         )       ';'
   .OUT ( 'O' )
   / '.ACT' '(' $(
       ( .STR "'" / .STR '"' ) .ACT ('C')
         ('*1' .ACT('U')
         /'*2' .ACT ('V')
         /.EMP)
         .ACT('O')
      / .STR '!'
      ) ')'
   /']' RVALUE .OUT('Y')
   / RVALUE '=:' TOPLACE .OUT()
   /'.IF' '(' CEXP ')' .OUT('F/' *1) $ NOTSYN
      ('.ELSE' .OUT('J/'*2)  .LAB*1 .OUT('S')   $ NOTSYN  '.END' .LAB*2
      /'.END'   .LAB*1 .OUT('S')  )
   /'.CAT'
      '(' $AOUT ')'
   /'.CONDLAB'
      ('*1' .OUT('UY')
         .OUT('Ø=')
         .OUT('T/' *1)
         .OUT('P... ')
         .OUT('U')
      /'*2' .OUT('VY')
         .OUT('Ø=')
         .OUT('T/' *1)
         .OUT('P... ')
         .OUT('V')
      )
      .OUT('O')
      .OUT('S') !!!NOTE SIDE EFFECT!!!
      .LAB *1
   /ERRORMESSAGE
   /'.ERROR' .OUT('SF')
   /'.SUCCEED' .OUT('S')
   /'.FAIL' .OUT('SF')
   / '&' IDNEW .OUT ( 'G' * )
   / '.RETURN' .OUT ( 'R' )
   /.STR '!'
   ;
CRIGHT=
   '==' .OUT('Y')
    RVALUE .OUT('=')
   /'!=' .OUT('Y')
    RVALUE .OUT('=') .OUT('SC')
   /'<=' .OUT('Y')
    RVALUE .OUT('<') .OUT('SC')
   /'>=' .OUT('Y')
    RVALUE .OUT('>') .OUT('SC')
   /'<' .OUT ('Y')
    RVALUE .OUT ('>')
   /'>' .OUT ('Y')
   RVALUE .OUT ('<')
   ;
CTERM = CFAC $( '.ANDIF' .OUT('F/' *2) CFAC)
   .CONDLAB *2
   ;
```

```
   / '.CONSTANT' $ (IDNEW ID=:SAVID
            CONSTEXP
            CONSTANT=:SAVID:TYPE
            UNSTACK=:SAVID:VALUE
          )      ';'
   ;
CONSTEXP = CONSTTERM $( '+' CONSTTERM .ACT('!+')
         /'-' CONSTTERM .ACT('!-')
         )
   ;
CONSTTERM = CONSTFAC $( '*' CONSTFAC .ACT('!*')
         )
   ;
CONSTSIMP = .NUM NUMBER=:STACK
   /IDCONS ID:VALUE=:STACK
   /.STR "'" .ACT('N/ØSY')   !VALUE IS FIRST CHARACTER
   /'.X' .HEXNUM ]NUMBER
   ;
CONSTFAC =
   CONSTSIMP
   /'-' CONSTFAC .ACT('!Z' 'N/ØY' 'Z-')
   /'(' CONSTEXP ')'
   ;
IDTYPE = .ACT ('LI'       !IS IT AN ID?
      'F/' *1    !NO,RETURN
      'ME'       !DEFINE OR FIND
      'ØM3Y' 'Z='    !ID.TYPE==TEMP
      'L' 'R'    !TAKE AND RETURN
      '.' *1       !NOW ACCEPT OR REWIND INPUT
      'L'  'Z9' )
   ;
IDNAME = .PREP NAME=:TEMP IDTYPE
   ;
IDFIELD = .PREP FIELD=:TEMP IDTYPE
   ;
IDCONS = .PREP CONSTANT=:TEMP IDTYPE
   ;
IDNEW = .PREP UNDEFINED=:TEMP IDTYPE
   ;
CHECKFRANGE = . IF ( NUMBER<=AVAILFIELDS)
      .RETURN
   .ELSE
   .ERMS ( NUMBER ' IS TOO BIG FOR A FIELD')
   .END

   ;
AOUT = '*1' .OUT ( 'U' )
   / '*2' .OUT ( 'V' )
   / '*' .OUT ( 'C' )
   /( .STR "'" /.STR'"') .OUT ( 'P' * )
   /RVALUE .OUT('C')
   /'.H' RVALUE .OUT('H')
   /'.' .OUT('XN')
   ;
NOTSYN = ( '.OUT' '(' $ AOUT ')'
   / '.LAB' .OUT ( 'P... ' ) AOUT
   )
```

*(Continued on top of page 22)*

```
CFAC = RVALUE CRIGHT
   /'.NOT' CFAC .OUT('SC')
   /'(' CEXP ')'
   /'&' IDNEW .OUT('G' *)
   ;
CEXP = CTERM $( '.ORIF' .OUT('T/' *1) CTERM)
   .CONDLAB *1
   ;
ERRORMESSAGE = '.ERMS' '(' .OUT('T/'*1) $AOUT
         ')' .OUT('XO' ) .LAB *1



   ;
SYN = (   IDNEW  .OUT ( 'G' * )
   / .STR "'"    .OUT ( 'LM' * ) .OUT ( 'L' )
   / '.ID'  .OUT ( 'LI' )
       .OUT( 'F/' *1 ) .OUT ( 'L' ) .OUT ( 'ME' ) .LAB *1
   / '.NUM'  .OUT ( 'LN' ) .OUT ( 'L' )
   / '.STR' (.STR "'" / .STR '"' ) .OUT ( 'LQ' * ) .OUT ( 'L' )
   /'.EMP' .OUT('S')
   /'$' .LAB *1 SYN .OUT('T/' *1 ) .OUT('S')
   /'(' PHRASE ')'
   /'.HEXNUM' .OUT('LH') .OUT('L')
   )
   (ERRORMESSAGE
   /.EMP)



   ;
SEQ =  SYN .OUT ( 'F/' *1 )
   $ ( SYN .OUT ( 'X' ) / NOTSYN )
   .LAB *1


   ;
ALTS = SEQ $('/' .OUT('T/'*1) SEQ)
   .CONDLAB *1


   ;
PHRASE = '.PREP' $NOTSYN ALTS
   /ALTS
   /NOTSYN $NOTSYN


   ;
STATEMENT =  .ID .LAB * '=' PHRASE ';' .OUT ( 'R' )
   (.STR '!' /.EMP)


   ;
COMPILE = '.SYN'  .ID .OUT('S'
       .OUT ( 'XM;' ) .OUT ( 'G' * ) .OUT ( 'E' )
   .ACT('XM;') $ DECLARE  $ STATEMENT
   ;
```

**(End Listing I. Listing II begins on next page.)**

# Write Your Own Compiler (Listing continued, text on pages 6-14)

## Listing II

```
S
XM;
GCOMPILE
E
... RVALUE
GIDNAME
F/1
GCATNAME
LM:
L
F/2
GIDFIELD
X
PM
ØMØC
... 2
T/3
S
F/4
... 4
... 3
X
... 1
T/5
GCONSTSIMP
F/6
PN/
!C
P
... 6
T/5
LM+
L
F/7
GEXPRESSION
X
P!
... 7
... 5
R
... EXPRESSION
GTERM
F/8
... 9
LM+
L
F/10
GTERM
X
P!+
O
... 10
T/11
LM-
L
F/12
GTERM
X
P!-
O
... 12
... 11
T/9
S
X
... 8
... 13
R
... TERM
GFACTOR
F/14
... 15
LM*
L
F/16
GFACTOR
X
P!*
O
... 16
T/17
LM/
L
F/18
LM256
L
X
PHZ
O
P!D
O
PZY
O
... 18
T/17
LM%256
L
F/19
PHY
O
PN/
P256*
O
P!-
O
... 19
... 17
T/15
S
X
... 14
... 20
R
... FACTOR
LM(
L
F/21
GEXPRESSION
X
LM)
L
X
... 21
T/22
GCONSTSIMP
F/23
PN/
!C
PY
O
... 23
T/22
GIDNAME
F/24
GCATNAME
LM:
L
F/25
GIDFIELD
X
PM
ØMØC
... 25
T/26
S
F/27
... 27
... 26
X
PY
O
... 24
T/22
LM-
L
F/28
GFACTOR
X
P!Z
O
PN/
PØY
O
PZ-
O
... 28
... 22
R
... 63
T/64
LM-
L
F/65
GCONSTFAC
X
!Z
N/ØY
Z-
... 65
T/64
LM(
L
F/66
GCONSTEXP
X
LM)
L
X
... 66
... 64
R
... IDTYPE
LI
F/67
ME
ØM3Y
Z=
L
R
.67
L
Z9
R
... IDNAME
N/1 Z
GIDTYPE
F/68
... 68
... 69
R
... IDFIELD
N/3 Z
GIDTYPE
F/70
... 70
... 71
R
... IDCONS
N/2 Z
GIDTYPE
F/72
... 72
... 73
R
... IDNEW
N/Ø Z
GIDTYPE
F/74
... 74
... 75
R
... CHECKFRANGE
ØY
N/5 <
SC
... 76
... 77
F/78
R
J/79
... 78
S
T/8Ø
ØC
P IS TOO BIG FOR A FIELD
XO
... 8Ø
... 79
R
... AOUT
LM*1
L
F/81
PU
O
... 81
T/82
LM*2
L
F/83
PV
O
... 83
T/82
LM*
L
F/84
PC
O
... 84
T/82
LQ'
L
F/85
... 85
T/86
LQ"
L
F/87
... 87
... 86
F/88
PP
C
O
... 88
T/82
GRVALUE
F/89
PC
O
... 89
T/82
LM.H
L
F/9Ø
GRVALUE
X
PH
O
... 9Ø
T/82
LM.
L
F/91
PXN
O
... 91
... 82
R
... NOTSYN
LM.OUT
L
F/92
LM(
L
X
... 93
GAOUT
T/93
S
X
LM)
L
X
... 92
T/94
LM.LAB
L
F/95
PP...
O
GAOUT
X
... 95
... 94
```

```
... CATNAME
ØMØY
!L
R
... TOPLACE
GIDNAME
F/29
Ø2
LM:
L
F/3Ø
GIDFIELD
X
PI
ØMØC
... 3Ø
T/31
S
F/32
... 32
... 31
X
2Ø
GCATNAME
... 29
... 33
R
... DECLARE
LM.FIELD
L
F/34
... 35
GIDNEW
F/36
Ø2
GCONSTEXP
X
GCHECKFRANGE
ØIØ2
N/3 I32
... 36
... 37
T/35
S
X
LM;
L
X

... 34
T/38
LM.NAME
L
F/39

... 4Ø
GIDNEW
F/41
```

```
Ø2
GCONSTEXP
X
!IØ2
N/1 I32
... 41
... 42
T/4Ø
S
X
LM;
L
X
... 39
T/38
LM.CONSTANT
L
F/43
... 44
GIDNEW
F/45
Ø2
GCONSTEXP
X
N/2 I32
!IØ2
... 45
... 46
T/44
S
X
LM;
L
X
... 43
... 38
R
... CONSTEXP
GCONSTTERM
F/47
... 48
LM+
L
F/49
GCONSTTERM
X
!+
... 49

T/5Ø
LM-
L
F/51
GCONSTTERM

X
!-
... 51
```

```
... 5Ø
T/48
S
X
... 47
... 52
R
... CONSTTERM
GCONSTFAC
F/53
... 54
LM*
L
F/55
GCONSTFAC
X
!*
... 55
... 56
T/54
S
X
... 53
... 57
R
... CONSTSIMP
LN
L
F/58
ØY
... 58
T/59
GIDCONS
F/6Ø
ØMØY
... 6Ø
T/59
LQ'
L
F/61
N/ØSY
... 61
T/59
LM.X
L
F/62
LH
L

X
ØY
... 62
... 59
R

... CONSTFAC
GCONSTSIMP
F/63
```

(Continued on top of page 24)

```
F/96
PO
O
... 96
T/97
LM.ACT
L
F/98
LM(
L
X
... 99
LQ'
L
F/1ØØ
... 1ØØ
T/1Ø1
LQ"
L
F/1Ø2
... 1Ø2
... 1Ø1
F/1Ø3
C
LM*1
L
F/1Ø4
U
... 1Ø4
T/1Ø5
LM*2
L
F/1Ø6
V
... 1Ø6
T/1Ø5
S
F/1Ø7
... 1Ø7
... 1Ø5
X
O
... 1Ø3
T/1Ø8
LQ!
L
F/1Ø9
... 1Ø9
... 1Ø8
T/99
S
X
LM)

L
X
... 98
```

```
T/97
LM]
L
F/11Ø
GRVALUE
X
PY
O
... 11Ø
T/97
GRVALUE
F/111
LM=:
L
X
GTOPLACE
X
O
... 111
T/97
LM.IF
L
F/112
LM(
L
X
GCEXP
X
LM)
L
X
PF/
U
O
... 113
GNOTSYN
T/113
S
X
LM.ELSE
L
F/114
PJ/
V
O
P...
U
O
PS
O
... 115
GNOTSYN
T/115

S
X
LM.END
```

(Continued on next page)

```
L
X
P...
V
O
... 114
T/116
LM.END
L
F/117
P...
U
O
PS
O
... 117
... 116
X
... 112
T/97
LM.CAT
L
F/118
LM(
L
X
... 119
GAOUT
T/119
S
X
LM)
L
X
... 118
T/97
LM.CONDLAB
L
F/12Ø
LM*1
L
F/121
PUY
O
PØ=
O
PT/
U
O
PP...
O
PU
O

... 121
T/122
LM*2
```

# Write Your Own Compiler (Listing II continued, text on pages 6-14)

```
L
F/123
PVY
O
PØ=
O
PT/
U
O
PP...
O
PV
O
... 123
... 122
X
PO
O
PS
O
P...
U
O
... 120
T/97
GERRORMESSAGE
F/124
... 124
T/97
LM.ERROR
L
F/125
PSF
O
... 125
T/97
LM.SUCCEED
L
F/126
PS
O
... 126
T/97
LM.FAIL
L
F/127
PSF
O
... 127
T/97
LM&
L
F/128
GIDNEW
X
PG
C
O
... 128
T/97
LM.RETURN
L
F/129
PR
O
... 129
T/97
LQ!
L
F/130
... 130
... 97
R
... CRIGHT
LM==
L
F/131
PY
O
GRVALUE
X
P=
O
... 131
T/132
LM!=
L
F/133
PY
O
GRVALUE
X
P=
O
PSC
O
... 133
T/132
LM<=
L
F/134
PY
O
GRVALUE
X
P<
O
PSC
O
... 134
T/132
LM>=
L
F/135
PY
O
GRVALUE
X
P>
O
PSC
O
... 135
T/132
LM<
L
F/136
PY
O
GRVALUE
X
P>
O
... 136
T/132
LM>
L
F/137
PY
O
GRVALUE
X
P<
O
... 137
... 132
R
... CTERM
GCFAC
F/138
... 139
LM.ANDIF
L
F/140
PF/
V
O
GCFAC
X
... 140
... 141
T/139
S
X
VY
Ø=
T/142
P...
... 167
T/159
LM$
L
F/168
P...
U
O
GSYN
X
PT/
U
O
PS
O
... 168
T/159
LM(
L
F/169
GPHRASE
X
LM)
L
X
... 169
T/159
LM.HEXNUM
L
F/170
PLH
O
PL
O
... 170
... 159
F/171
GERRORMESSAGE
F/172
... 172
T/173
S
F/174
... 174
... 173
X
... 171
... 175
R
... SEQ
GSYN
F/176
PF/
U
O
... 177
GSYN
F/178
PX
O
... 178
T/179
GNOTSYN
F/180
... 180
... 179
T/177
S
X
P...
U
O
... 176
... 181
R
... ALTS
GSEQ
F/182
... 183
LM/
L
F/184
PT/
U
O
GSEQ
X
... 184
... 185
T/183
S
X
UY
Ø=
T/186
P...
U
O
S
... 186
... 182
... 187
R
... PHRASE
LM.PREP
L
F/188
... 189
GNOTSYN
T/189
S
X
GALTS
X
... 188
T/190
GALTS
F/191
... 191
T/190
GNOTSYN
F/192
... 193
GNOTSYN
T/193
S
X
... 192
... 190
R
... STATEMENT
LI
F/194
L
ME
... 194
F/195
P...
C
O
LM=
L
X
GPHRASE
X
LM;
L
X
PR
O
LQ!
L
F/196
... 196
T/197
S
F/198
... 198
... 197
X
... 195
... 199
R
... COMPILE
LM.SYN
L
F/200
LI
```

```
V
O
S
... 142
... 138
... 143
R
... CFAC
GRVALUE
F/144
GCRIGHT
X
... 144
T/145
LM.NOT
L
F/146
GCFAC
X
PSC
O
... 146
T/145
LM(
L
F/147
GCEXP
X
LM)
L
X
... 147
T/145
LM&
L
F/148
GIDNEW
X
PG
C
O
... 148
... 145
R
... CEXP
GCTERM
F/149
... 150
LM.ORIF
L
F/151
PT/
U

O
GCTERM
X
... 151
... 152
T/150
S
X
UY
Ø=
T/153
P...
U
O
S
... 153
... 149
... 154
R
... ERRORMESSAGE
LM.ERMS
L
F/155
LM(
L
X
PT/
U
O
... 156
GAOUT
T/156
S
X
LM)
L
X
PXO
O
P...
U
O
... 155
... 157
R
... SYN
GIDNEW
F/158
PG
C
O
... 158
T/159
LQ'
L
F/160

PLM
C
O
PL
O
... 160
T/159
LM.ID
L
F/161
PLI
O
PF/
U
O
PL
O
PME
O
P...
U
O
... 161
T/159
LM.NUM
L
F/162
PLN
O
PL
O
... 162
T/159
LM.STR
L
F/163
LQ'
L
F/164
... 164
T/165
LQ"
L
F/166
... 166
... 165
X
PLQ
C
O
PL
O
... 163
T/159
LM.EMP
L

F/167
PS
O
```

*(Continued on top of page 26)*

```
F/201
L
ME
... 201
X
PS
O
PXM;
O
PG
C
O
PE
O
XM;
... 202
GDECLARE
T/202
S
X
... 203
GSTATEMENT
T/203
S
X
... 200
... 204
R
```

**End Listing II.**

# Listing III

```
TOP;METATERP
;COPYRIGHT (C) 1981 W.A.GALE
HIMEM=36864;$9000
LOMEM=02048;$800
REGISTER=086
BYTE AA;ALL LOW DOUBLE LETTERS ARE TEMP VBLS
BYTE BB
BYTE BO(080);OUTPUT STRING
BYTE C0;NUMBER ZERO
BYTE C1;NUMBER ONE
BYTE C2;NUMBER TWO
BYTE C3;NUMBER THREE
BYTE C9;NUMBER NINE
BYTE CB;BLANK
BYTE CC
BYTE CD;'.' DOT
BYTE CE;'/' ESCAPE FOR NUMBERS
BYTE CG;'>'
BYTE CL;'<'
BYTE CM;'-'
BYTE CP;'+'
BYTE CQ;SINGLE QUOTE
BYTE CS;'*'
BYTE CT;TAB
BYTE CU;'='
BYTE CV;NUMBER 25
BYTE CX;'!'
BYTE DD
BYTE DS(010);DIGIT STACK FOR WRITING NUMBERS
BYTE EE
BYTE F1(00275);INPUT BUFFER
BYTE F2(00275);OUTPUT BUFFER
BYTE FL;FLAG FOR XRUE AND FALSE JUMPS
BYTE KS(10000);PROGRAM MEMORY SPACE
BYTE LI;LENGTH OF INSTRUCTION
BYTE LL;LINE LENGTH DURING LOADING
BYTE MC(03000);SYMBOLIC MEMORY CHARACTER VECTOR
BYTE MK;MEMORY CELL SIZE
BYTE MN;DIMENSION OF NS LESS ONE
```

*(Continued on next page)*

# Write Your Own Compiler (Listing III continued, text on pages 6-14)

```
BYTE ND;NUMBER OF DIGITS FOR WRITING NUMBERS
BYTE NL;NEWLINE !!!! SYSTEM DEPENDENT
BYTE NS(080);CIRCULAR INPUTBUFFER
BYTE OS(080);INPUT STRING
BYTE PB;POINTER INTO BO
BYTE PI;INDEX INTO RI
BYTE PL;POINTER INTO NS
BYTE PM;BEGINNING IN NS OF INPUT NOT ACCEPTED
BYTE PN;COUNT OF SUBROUTINES AS LOADED
BYTE PO;POINTER INTO OS
BYTE QI;POINTER INTO RI
BYTE RC;COMMAND READ
BYTE RI(080);INSTRUCTION REGISTER
BYTE SD;STACK DIMENSIONS BOTH Y AND Z
BYTE WA;WORK IN PACK
BYTE WB;WORK IN PACK
BYTE X0;CHARACTER ZERO
BYTE X1
BYTE X2
BYTE X3
BYTE X9
BYTE XA;ALL X VARIABLES ARE CHARACTERS
BYTE XB
BYTE XC
BYTE XD
BYTE XE
BYTE XF
BYTE XG
BYTE XH
BYTE XI
BYTE XJ
BYTE XK
BYTE XL
BYTE XM
BYTE XN
BYTE XO
BYTE XP
BYTE XQ
BYTE XR
BYTE XS
BYTE XT
BYTE XU
BYTE XV
BYTE XW
BYTE XX
BYTE XY
BYTE XZ
BYTE YP;Y STACK POINTER
BYTE ZP;Z STACK POINTER
BYTE ZX;STORAGE FOR ERROR RECOVERY SYMBOL
INT I00;NUMBER ZERO
INT I01;NUMBER ONE
INT I03;NUMBER THREE
INT I10;NUMBER TEN
INT I16;NUMBER 16
INT IAA;WORKING
```

```
 CHOOSE ON CC
 CASE XM;MATCH SPECIFIC STRING
    FL=+000
    BB=+002
    WHILE
       AA=BB<!PI
    ON AA
       AA=RI(BB)
       DD=NS(PL)
       AA=AA!=DD
       IF AA
          GOTO 99;NO MATCH
       ENDIF
       BB++
       GOSUB LA
    ENDWHILE
    FL=+001
    GOSUB LB
 CASE XI;ID TEST
    FL=+000
    CC=NS(PL)
    PO=+000
    GOSUB ZA
    WHILE
    ON AA
       OS(PO)=CC
       PO++
       GOSUB LA
       CC=NS(PL)
       GOSUB ZA
       DD=AA
       GOSUB ZN
       AA=DD?AA
    ENDWHILE
    AA=PO==C0
    IF AA
       GOTO 99
    ENDIF
    GOSUB MS;SEARCH ALL LEVELS
    IPR(C0)=IAA
    FL=+001
 CASE XN; POSITIVE INTEGER TEST
    FL=+000
    IAA=I00
    WHILE
       CC=NS(PL)
       GOSUB ZN
    ON AA
       FL=+001
       IAA=IAA*I10
       CC=CC-X0
       IBB=CC
       IAA=IAA+IBB
       GOSUB LA
    ENDWHILE
    IPR(C0)=IAA
```

```
INT IBB;WORKING
INT IBK;BLOCK NUMBER FOR INPUT OR OUTPUT FILE
INT ICC;WORKING
INT IDD;WORKING
INT ILB;POINTER INTO ILT
INT ILN;LINE NUMBER OF INPUT
INT ILT(01000);LOCATION TABLE FOR NUMBER LABELS
INT IMB;MEMORY BASE INDEX FOR CURRENT LEVEL
INT IMD;DIMENSION OF MC AND IMI
INT IMF;MEMORY FREE INDEX
INT IMI(03000);SYMBOLIC MEMORY INDEX VECTOR
INT IML;NUMBER OF LOCAL PARAMETERS PER MEM LEVEL
INT IMM;MAX MEM COUNTER
INT IMT;TOP OF FREE MEMORY, REST TAKEN BY CELL STACK
INT IMX;NULL MEMORY INDEX
INT IMZ;TEM VBL FOR MEM SEARCH
INT INL;NUMBER OF NUMERICAL LABELS
INT IPC;PROGRAM COUNTER, INDEX TO CURRENT INST
INT IPL;CODE POINTER WHILE LOADING
INT IPR(010);REGISTER VECTOR
INT IPT;POINTER INTO IST, SUBROUTINE STACK
INT IRN;NUMBER RETURNED FROM READ NUMBER ROUTINE
INT ISM;SYMBOL NUMBER OF INPUT
INT IST(00600);SUBROUTINE AND LABEL STACK
INT ITU;RESULT OF DIRECT FETCH
INT IUU;UNIQUE -- SYMBOL GENRATOR
INT IXX;WORK DURING NUMBER MANIP
INT IYS(080);Y STACK
INT IYY;WORK DURING NUMBER MANIP
INT IZC;ERROR PROGRAM COUNTER
INT IZS(080);ZSTACK
INT IZT;ERROR STACK POINTER
BEGINMAIN(AC,IAV)
NL=+010;!!!!!!!
MS 'COPYRIGHT'
MS ' (C) 1981'
MS ' W.A.GALE'
WRITE NL
GOSUB IN
GOSUB RC
GOSUB LI;LEXICAL INITIALIZATION
IPC=+00000
LOC 00
GOSUB GI
CC=RI(C0)
CHOOSE ON CC
CASE XL;LEXICAL ANALYSIS COMMANDS
   AA=PI==C1
   IF AA
      IF FL
         GOSUB LW;SEEK WHITE SPACE AND MOVE UP BASE
         ISM++;INCREMENT THE SYMBOL COUNT
      ELSE
         PL=PM;RESET THE LOOKAHEAD PINTER
      ENDIF
   ELSE;LONGER THAN ONE
   CC=RI(C1)
```

*(Continued on top of page 28)*

```
CASE XH;HEX NUMBER TEST
   FL=+000
   IAA=+00000
   WHILE
      CC=NS(PL)
      GOSUB ZH
   ON AA
      FL=+001
      IAA=IAA*I16
      IBB=CC
      IAA=IAA+IBB
      GOSUB LA
   ENDWHILE
   IPR(C0)=IAA
CASE XQ;STRING QUOTED BY
   DD=RI(C2)
      CC=NS(PL)
      PO=+000
      AA=CC==DD
      IF AA
         GOSUB LA
         WHILE
            CC=NS(PL)
            AA=CC!=NL
            BB=CC!=DD
            AA=AA&BB
         ON AA
            OS(PO)=CC
            PO++
            GOSUB LA
         ENDWHILE
         GOSUB LA
         AA=CC==NL
         IF AA
            ILN++
            ISM=I00
         ENDIF
         FL=+001
      ELSE
         FL=+000
      ENDIF
   DEFAULT
      WRITE CC
      MS ' NOT LEX!'
      WRITE NL
   ENDCHOOSE
   ENDIF;TWO LETTER COMMANDS
CASE XF;FALSE JUMP
   IF FL
   ELSE
      GOTO 20
   ENDIF
CASE XP;PRINT STRING GIVEN
   BB=+001
   WHILE
      AA=BB<!PI
   ON AA
```

*(Continued on next page)*

# Write Your Own Compiler (Listing III continued, text on pages 6-14)

```
      CC=RI(BB)
      BO(PB)=CC
      PB++
      BB++
   ENDWHILE
CASE XO;OUT
   BB=+000
   WHILE
      AA=BB<!PB
   ON AA
      CC=BO(BB)
      BB++
      WRITE CC INTO F2
   ENDWHILE
   PB=C0

   AA=PI==C1
   IF AA
      WRITE NL INTO F2
   ENDIF
CASE XX;ERROR JUMP
   AA=PI==C1
   IF AA;JUST AN X
      IF FL
      ELSE
   LOC 98
      MS 'ERROR AT '
      MS 'LINE NUM '
      IAA=ILN
      GOSUB PN
      MS ' SYMBOL  '
      IAA=ISM
      GOSUB PN
      WRITE CB
      WRITE NL
      WHILE
         CC=NS(PL)
         AA=CC!=ZX;AND COMPARE TO SPECIAL CHARACTER
         BB=CC!=C0;AND END OF FILE SIGNAL
         AA=AA&BB
      ON AA
         AA=CC==NL
         IF AA
            ILN++
            ISM=+00000
         ENDIF
         GOSUB LA;READ ONE MORE
         GOSUB LB;AND BRING UP REAR
      ENDWHILE;HAVE JUST READ SPECIAL CHARACTER
      BB=CC==C0
      IF BB;OR END OF FILE CHAR
         MS 'END OFILE'
         GOTO 21
      ENDIF
      GOSUB LA;NOW MOVE BEYOND THE ZX
      GOSUB LB
      GOSUB LW;NOW EAT UP ANY WHITE SPACE AFTER IT
   ENDIF
   IPT=IPT-I03
CASE XS;SET
   AA=PI==C1
   IF AA
      FL=+001
   ELSE
      CC=RI(C1)
      CHOOSE ON CC
      CASE XF;SET FALSE
         FL=+000
      CASE XC;SET CHANGED
         FL=C1-FL
      DEFAULT
         MS 'SET ERROR'
         WRITE NL
      ENDCHOOSE
   ENDIF
CASE XU;UNIQUE NUMBER GENERATED AND STACKED
AA=PI==C1
IF AA;THEN DO A LOT OF WORK
   IAA=IPT
   IAA++;POINTS TO UN1
LOC 10
   IBB=IST(IAA);THE CURRENT UNIQUE
   AA=IBB<!I01;IE NEVER FILLED IN YET
   IF AA
      IUU++
      IBB=IUU
      IST(IAA)=IUU
   ENDIF
   IAA=IBB
   IPR(C0)=IAA
   GOSUB WN;WRITE NUMBER INTO BUFFER
ELSE;THIS IS A FETCH FROM U
   GOTO 22
ENDIF
CASE XC;COPY INPUT
   BB=+000
   WHILE
      AA=BB<!PO
   ON AA
      CC=OS(BB)
      BO(PB)=CC
      PB++
      BB++
   ENDWHILE
CASE XV;UNIQUE NUMBER 2
AA=PI==C1
IF AA;THEN THIS IS TO GENERATE A NUMBER
   IAA=IPT
   IAA++
   IAA++
   GOTO 10;IAA POINTS TO SECOND UNIQUE
ELSE;JUST A FETCH
   GOTO 22
```

```
   IPC=IZC;RESTORE PROGRAM COUNTER
   IPT=IZT;AND STACK POINTER
   FL=+001;AND DECLARE TRUE
   ENDIF
ELSE;A LONGER COMMAND
CC=RI(C1)
CHOOSE ON CC
CASE XN;WRITE LINE NUMBER INTO OUTBUF
   IAA=ILN
   GOSUB WN
CASE XO;WRITE OUTBUF TO ERROR OUTPUT
   BB=+000
   WHILE
      AA=BB<!PB
   ON AA
         CC=BO(BB)
         BB++
         WRITE CC
      ENDWHILE
      WRITE NL
      PB=C0
   CASE XM;MARK FOR ERROR RETURN
      IZC=IPC;SAVE CODE POSITION
      IZT=IPT;SAVE STACK POSITION
      ZX=RI(C2);AND SPECIAL CHARACTER TO READ THROUGH
   DEFAULT
   ENDCHOOSE
   ENDIF;X + LETTER
CASE XT;TRUE JUMP
   IF FL
      GOTO 20
   ENDIF
CASE XG;GOSUB LABEL MUST BE ALPHA
   WA=RI(C1)
   WB=RI(C2)
   IPT=IPT+I03
   IAA=+00597;STACK DEPTH-3
   AA=IAA<=IPT
   IF AA
      MS 'STACK OVE'
      MS 'R FLOW>>>'
      GOTO 98
   ENDIF
   IST(IPT)=IPC
   PACK(IPC,WA,WB)
   IAA=IPT
   IAA++
   IST(IAA)=I00
   IAA++
   IST(IAA)=I00
CASE XR;RETURN
   IPC=IST(IPT)
   AA=IPT<!I03
   IF AA
      MS 'STACK UND'
      MS 'ERFLOW...'
      GOTO 98
```

*(Continued on top of page 30)*

```
ENDIF
CASE XM;MEMORY OPERATIONS
   CC=RI(C1)
   CHOOSE ON CC
   CASE XS;STACK MEMORY
      GOSUB MH
   CASE XP;POP MEMORY
      GOSUB MP
   CASE XE;DEFINE A CELL IN TOP
      GOSUB ME
      IPR(C0)=IAA
   CASE XQ;QUERY
      GOSUB MS
      IPR(C0)=IAA
   CASE XC;CREATE CELL
      GOSUB MC
      IPR(C0)=IAA
   CASE XD;DESTROY CELL
      GOSUB MD
      IPR(C0)=IAA
   CASE XI;INITIALIZE
      CC=RI(C2)
      GOSUB ZN
      IF AA
         MK=CC-X0
      ELSE
         MK=C2
      ENDIF
      GOSUB MI
   DEFAULT
      MS 'ILG MEMOP'
      WRITE NL
   ENDCHOOSE
CASE XJ;JUMP UNCOND_ LABEL MUST BE NUMBER
LOC 20
   AA=RI(C1)
   BB=RI(C2)
   PACK(ILB,AA,BB)
   IPC=ILT(ILB)
CASE XE;END --STOP HERE
LOC 21
   CLOSE F1
   CLOSE F2
   IAA=IMM;MAX MEM USED
   GOSUB PN
   MS 'MAXMEMUSE'
   WRITE NL
   STOP 0
DEFAULT;LOOK FOR FETCH AND STORE INSTRUCTION
LOC 22
   QI=+000
   GOSUB FT
   GOSUB FI
   GOSUB ST
ENDCHOOSE
GOTO 00
LOC 99
```

*(Continued on next page)*

# Write Your Own Compiler (Listing III continued, text on pages 6-14)

```
FL=+000
GOTO 00
ENDMAIN
        SUB CK;CHECK OPENED
AA=ER!=C0
IF AA
   MS 'CANT OPEN'
   IAA=IBK
   GOSUB PN
   WRITE NL
   STOP 1
ENDIF
ENDSUB

        SUB DS;STACK DIGITS FOR PN, WN
AA=IAA<!I00
IF AA
   BB=+001
   IAA=-IAA
ELSE
   BB=+000
ENDIF
AA=IAA==I00
IF AA
   ND=C1
   DS(C0)=X0
ELSE
   ND=C0
   WHILE
      AA=I00<!IAA
   ON AA
      IYY=IAA/I10
      IBB=I10*IYY
      IXX=IAA-IBB
      IAA=IYY
      AA=IXX
      AA=AA+X0
      DS(ND)=AA
      ND++
   ENDWHILE
ENDIF
DS(ND)=CM;'-'
ND=ND+BB;IE ONLY INCR IF NEG
ENDSUB
        SUB FI;FETCH INDIRECT
QI++
CC=RI(QI)
CHOOSE ON CC
CASE XM;MEMORY FETCH
   QI++
   CC=RI(QI)
   GOSUB ZN
   IF AA
      BB=CC-X0
   ELSE
   LOC 11
   ITU=AA
CASE XB;FETCH BREDTH OF STRING
   ITU=PO
CASE XU;FIRST STACK NUMBER
   IAA=IPT
LOC 38
   IAA++
   ITU=IST(IAA)
CASE XV;SECOND STACK NUMBER
   IAA=IPT
   IAA++
   GOTO 38
DEFAULT;NUMBER OR ERROR
   GOSUB ZN
   IF AA
      AA=CC-X0
   ELSE
      MS 'ILLG FTCH'
      WRITE NL
      AA=C0
   ENDIF
   ITU=IPR(AA)
ENDCHOOSE
ENDSUB;FT
        SUB GI;GET INSTRUCTION
PI=+000
LI=KS(IPC)
IPC++
WHILE
   AA=PI<!LI
ON AA
   AA=KS(IPC)
   IPC++
   RI(PI)=AA
   PI++
ENDWHILE
ENDSUB;GI
        SUB IN;INITIALIZE  SIGH
ZX=+000
YP=+000
ZP=+000
IZC=+00000
IZT=+00000
XA='A'
XB='B'
XC='C'
XD='D'
XE='E'
XF='F'
XG='G'
XH='H'
XI='I'
XJ='J'
XK='K'
XL='L'
XM='M'
```

```
   MS 'INDEX TO '
   MS 'MEM CELL '
   BB=+000
ENDIF
AA=BB<!MK
IF AA
   IAA=BB
   IAA=IAA+ITU
   ITU=IMI(IAA)
   RETURN
ELSE
   BB=BB-MK
   AA=BB<!MK
   IF AA
         IAA=BB
         IAA=ITU+IAA
         AA=MC(IAA)
         ITU=AA
      ELSE
         GOTO 11
      ENDIF
   ENDIF
CASE XS;FETCH FROM STRING REGISTER
   AA=ITU
   BB=OS(AA)
   ITU=BB
DEFAULT
   QI--
ENDCHOOSE
ENDSUB
         SUB FT;FETCHES DIRECT
CC=RI(QI)
CHOOSE ON CC
CASE XY;Y STACK
   ITU=IYS(YP)
CASE CX;'!' POP Y
   ITU=IYS(YP)
   GOSUB PY
CASE XZ;Z STACK ALWAYS POP
   ITU=IZS(ZP)
   AA=ZP==C0
   IF AA
      MS 'Z STACKER'
      WRITE NL
      ZP=C1
      FL=+000
   ENDIF
   ZP--
CASE XN;LITERAL FETCH OF NUMBER
   QI++
   AA=RI(QI)
   QI++
   BB=RI(QI)
   PACK(ITU,AA,BB)
CASE XH;FETCH HIGH OF STACK
   ITU=IYS(YP)
   UNPACK(ITU,AA,BB)
```

*(Continued on top of page 32)*

```
XN='N'
XO='O'
XP='P'
XQ='Q'
XR='R'
XS='S'
XT='T'
XU='U'
XV='V'
XW='W'
XX='X'
XY='Y'
XZ='Z'
X0='0'
X1='1'
X2='2'
X3='3'
X9='9'
C9=+009
CV=+025
C0=+000
C1=+001
C2=+002
C3=+003
CB=' '
CX='!'
CS='*'
CM='-'
CP='+'
CG='>'
CU='='
CL='<'
SD=+080
CT='    '
CE='/'
CD='.'
CQ='''
I00=+00000
I03=+00003
I01=+00001
I10=+00010
I16=+00016
MN=+079;DIMENSION OF NS LESS 1
IBK=IAV(C3)
CLOSE F2
OPEN F2 FOR XW AT IBK
FL=+000
PI=+000
PB=+000
IPC=+00000
PO=+000
IPT=+00000
ILB=+00000
PN=+000
IUU=+00000
ILN=+00001
ISM=+00000
```

*(Continued on next page)*

# Write Your Own Compiler (Listing III continued, text on pages 6-14)

```
INL=+00000
MK=+002
GOSUB MI;INITIALIZE MEMORY
ENDSUB
          SUB LA;L IS FOR LEX A IS FOR AHEAD
AA=PL==MN;MAX FOR NS
IF AA
   PL=C0
ELSE
   PL++
ENDIF
ENDSUB
          SUB LB; IN THIS MOVE UP THE BASE
WHILE
   AA=PL!=PM
ON AA
   READ CC FROM F1
   AA=ER!=C0
   IF AA
      CC=+000
   ENDIF
   NS(PM)=CC
   AA=PM==MN
   IF AA
      PM=+000
   ELSE
      PM++
   ENDIF
ENDWHILE
ENDSUB
          SUB LI;INITIALIZE
PM=+000
PL=+000
BB=+000
WHILE
   AA=BB<=MN
   CC=ER==C0
   AA=AA&CC
ON AA
   READ CC FROM F1
   NS(BB)=CC
   BB++
ENDWHILE
ENDSUB
          SUB LW;TEST AND DISCARD WHITESPACE
CC=NS(PL)
WHILE
   AA=CC==NL
   IF AA
      ILN++
      ISM=+00000
   ENDIF
   BB=CC==CB
   AA=AA?BB
   BB=CC==CT
   AA=AA?BB
```

```
IMB=+00000
IMM=+00000
IMD=+03000;!!!!!!!DIMENSION OF MC AND IMI
IMT=IMD;START TOP AT DIMENSION
IML=+00001;NUMBER OF LOCAL PARAMS
;IMB IS FIRST, IMF IS SECOND
IMF=IMB+IML
IMX=I00
IMI(IMB)=I00
MC(IMF)=C0
IMI(IMF)=I00
ENDSUB;MI
          SUB ML;LEVEL SEARCH PATTERN ENDS WITH NUL
IBB=IMB+IML;FIRST CHAR IN MEM
BB=+000;SUBSCRIPT FOR PATTERN
WHILE
   CC=OS(BB);PATTERN CHAR
   DD=MC(IBB);MEM CHAR
   EE=CC==DD
   IF EE;MATCHING
      EE=CC==C0;END OF STRIG SYMBOL
      IF EE;ENTIRE MATCH
         IAA=IBB+I01
         GOTO 77
      ENDIF
      IBB++
      BB++
   ELSE;FAILED
      IAA=IMI(IBB)
      EE=IAA==IMX
      IF EE;END OF ROAD
         IAA=I00
         GOTO 77
      ENDIF
      IBB=IAA
   ENDIF
   EE=IBB<!IMF
ON EE
ENDWHILE
IAA=I00;FAILED TO FIND BELOW FREE MARKER
LOC 77
ENDSUB;ML
          SUB MO;CHECK FOR MEMORY OVERFLOW
AA=IMT<!IMF
IF AA
   MS 'OUT OFMEM'
   WRITE NL
   STOP 3
ENDIF
AA=IMM<!IMF
IF AA
   IMM=IMF;THIS IS THE MAXIMUM STORE USED
ENDIF
ENDSUB;MO
          SUB MP;POP A LEVEL
AA=IMB!=I00
```

```
ON AA
   GOSUB LA
   CC=NS(PL)
ENDWHILE
GOSUB LB
ENDSUB
         SUB MC;CREATE CELL AT TOP
IAA=MK
IMT=IMT-IAA;LOWER TOP BY CELL SIZE
GOSUB MO;CHECK FOR OVERFLOW
IAA=IMT;POIT TO CELL
GOSUB MZ;AND ZERO
ENDSUB;MC
         SUB MD;DESTROY CELL
IAA=MK
IMT=IMT+IAA;RAISE TOP BY CELL SIZE
AA=IMD<!IMT
IF AA
   MS 'DESTROY C'
   MS 'ELL ERROR'
   WRITE NL
ENDIF
IAA=IMT
ENDSUB;MD
         SUB ME;ENTER INTO TOP LEVEL
;RETURNS IAA AS INDEX TO CELL
GOSUB ML;SEARCH TOP LEVEL
EE=IAA!=I00;FOUND IT HERE
IF EE
   RETURN
ENDIF
IMI(IBB)=IMF;IBB FROM ML, STORE TO SHOW THIS OPTION
WHILE
   MC(IMF)=CC;CC FROM ML ONCE
   IMI(IMF)=IMX; NO ALTERNATIVES NOW
   IMF++
   GOSUB MO;CHECK OVERFLOW
   AA=CC!=C0;NOT LAST CHAR
ON AA
   BB++;BB FROM ML TO START
   CC=OS(BB)
ENDWHILE
IAA=IMF
IAA=IMF
IDD=MK
IMF=IMF+IDD
GOSUB MO
GOSUB MZ
ENDSUB;ME
         SUB MH;PUSHHH A NEW LEVEL
IMI(IMF)=IMB;POINT TO CURRENT BASE
IMB=IMF
IMF=IMF+IML;LOCAL PARAM SPACES
MC(IMF)=C0
IMI(IMF)=I00
ENDSUB
         SUB MI;MEMORY INITIALIZE
```

*(Continued on top of page 34)*

```
IF AA
   IMF=IMB
   IMB=IMI(IMB)
ELSE
   IMF=IML
   MC(IMF)=C0
   IMI(IMF)=I00
ENDIF
ENDSUB;MP
         SUB MS;SEARCH ALL LEVELS
OS(PO)=C0

IMZ=IMB;SAVE CURRENT BASE
WHILE
   GOSUB ML;SEARCH A LEVEL
   EE=IAA==I00;NOT MATCHED
   IMB=IMI(IMB);FOR NEXT LOWER
   DD=IMB!=I00;NOT BOTTOM
   CC=EE&DD
ON CC
ENDWHILE
IMB=IMZ;BACK TO TOP LEVEL
ENDSUB;MS
         SUB MZ;ZERO A MEMORY CELL
BB=+000
IDD=IAA;PRESERVE POINTER TO CELL
WHILE
   AA=BB<!MK
   BB++
ON AA
   MC(IDD)=C0
   IMI(IDD)=I00
   IDD++
ENDWHILE
ENDSUB;MZ
         SUB PN;WRITE NUMBER IAA ON TERMINAL
GOSUB DS
WHILE
   IBB=ND
   AA=I00<!IBB
ON AA
   ND--
   AA=DS(ND)
   WRITE AA
ENDWHILE
WRITE CB
ENDSUB
         SUB PY;POP Y STACK
AA=YP==C0
IF AA
   MS 'Y STACKER'
   WRITE NL
   YP=C1
   FL=+000;AND TELL AN ERROR
ENDIF
YP--
ENDSUB
```

*(Continued on next page)*

# Write Your Own Compiler (Listing III continued, text on pages 6-14)

```
         SUB RA;READ ALPHA LABEL
;FIRST COPY LABEL TO STRING INDEX
OS(CØ)=CC
PO=+ØØ1
WHILE
   READ CC FROM F1
   GOSUB ZA
   DD=AA;SAVE THE RESULT
   GOSUB ZN;ALLOW EITHER ALPHA OR NUMERIC AFTER FIRST
   AA=AA?DD
ON AA
   OS(PO)=CC
   PO++
ENDWHILE
;NOW EAT UP LINE
AA=CC==NL
IF AA
ELSE
   WHILE
      READ CC FROM F1
      AA=CC!=NL
   ON AA
   ENDWHILE
ENDIF
ENDSUB
         SUB RC;READ COMMANDS
CLOSE F1
IBK=IAV(C1)
OPEN F1 FOR XR AT IBK
GOSUB CK;CHECK OPENED OK
IPC=+00001
LL=+ØØØ
IPL=+00000
WHILE
   READ RC FROM F1
LOC 33
   AA=ER==CØ
ON AA
   CHOOSE ON RC
   CASE CE;'/' NUMBER FOLLOWS
      READ CC FROM F1
      AA=CC==CM
      IF AA;IF NEGATIVE SIGN
         READ CC FROM F1
         GOSUB RN;THEN READ NUMBER
         IRN=-IRN;AND INVERT
         GOTO 37
      ENDIF
      GOSUB ZN
      IF AA
         GOSUB RN;READ A NUMBER
      LOC 37
         UNPACK(IRN,AA,BB)
         KS(IPC)=AA
         IPC++
         LL++
      ENDIF
   DEFAULT;ACCEPT
   LOC 35
      KS(IPC)=RC
      IPC++
      LL++
   ENDCHOOSE
ENDWHILE
;HAVE REACHED END
IPC--
AA=ER!=C1
IF AA;NO, AN ERROR
   MS 'CANT READ'
   MS 'COMMANDS.'
   WRITE NL
   STOP 1
ENDIF
CLOSE F1
IBK=IAV(C2)
OPEN F1 FOR XR AT IBK
GOSUB CK
;NOW PATCH ALL THE GOSUBS THAT WERENT DEFINED
IAA=+00000
WHILE
   AA=IAA<!IPC
ON AA
   LL=KS(IAA)
   IBB=IAA+IØ1
   AA=KS(IBB);THE COMMAND
   AA=AA==NL
   IF AA;A GOSUB NEEDING PATCH
      KS(IBB)=XG;RESTORE THE RIGHT COMMMAND
      IBB++
      AA=KS(IBB);GET THE INDEX TO DEFN
      IBB++
      BB=KS(IBB)
      PACK(IRN,AA,BB)
      BB=MC(IRN)
      AA=BB!=C1
      IF AA;THE SUB WAS NEVER DEFINED. BUT WE DONT KNOW NAME
         ICC=IRN-I1Ø
         WHILE
            AA=ICC<!IRN
         ON AA
            BB=MC(ICC)
            WRITE BB
            ICC++
         ENDWHILE
         WRITE CB
         MS 'SUB UNDEF'
         WRITE NL
      ENDIF
      ICC=IMI(IRN)
      UNPACK(ICC,AA,BB);ICC IS THE REAL LOCATION OF THE SUB
      KS(IBB)=BB;AND STORE IT
      IBB--
```

```
      KS(IPC)=BB
      IPC++
      LL++
      RC=CC
      AA=RC==CB
      IF AA
      ELSE
         GOTO 33
      ENDIF
   ELSE
      KS(IPC)=CE
      IPC++
      LL++
      RC=CC
      GOTO 33
   ENDIF
CASE NL;END OF COMMAND
   KS(IPL)=LL
   IPL=IPC
   IPC++
   LL=C0
CASE CD;'.' LABEL FOLLOWS
   AA=LL==C0
   IF AA
      GOSUB RL
      RC=NL
      IPC--
      GOTO 33
   ELSE
      GOTO 35
   ENDIF
CASE XG;IF AT BEGINNING, A GOSUB TO COMPILE
   AA=LL==C0
   IF AA
      READ CC FROM F1
      GOSUB RA;READ ALPHA LABEL
      OS(PO)=C0
      GOSUB ME;THIS WILL FIND OR DEFINE
      IRN=IMI(IAA);DEFINED VALUE
      AA=IRN==I00
      IF AA
         IRN=IAA;NOT DEFINED YET
         KS(IPC)=NL;SET WARNING
      ELSE
         KS(IPC)=RC;WAS DEFINED SO WE HAVE FILLED VALUE
      ENDIF
      IPC++
      UNPACK(IRN,AA,BB)
      KS(IPC)=AA
      IPC++
      KS(IPC)=BB
      IPC++
      LL=C3
      RC=NL
      GOTO 33
   ELSE
      GOTO 35
```

*(Continued on top of page 36)*

```
      KS(IBB)=AA
   ENDIF
   IBB=LL
   IAA=IAA+IBB;THREAD WAY TO NEXT COMMAND
   IAA=IAA+I01
ENDWHILE
GOSUB MP;POP MEMORY SO FORGET SUB NAMES
IAA=IPC
GOSUB PN
MS 'CMD BYTS '
IAA=INL
GOSUB PN
MS 'NUM LABS '
IAA=PN
GOSUB PN
MS 'SUBROUTIN'
WRITE NL
ENDSUB
         SUB RL;READ LABEL
;FIRST DISCARD ANY NON ALPHANUMERIC
WHILE
   READ CC FROM F1
   GOSUB ZA
   IF AA
      GOTO 80;ALPHA LABEL
   ENDIF
   GOSUB ZN
   IF AA
      GOTO 85;NUMBERLABEL
   ENDIF
   AA=CC!=NL;IF NO LABEL, NO ACTION
ON AA
ENDWHILE
RETURN
LOC 80;ALPHA LABEL
GOSUB RA;READ ALPHA LABEL
OS(PO)=C0
GOSUB ME;SEARCH MEMORY
IMI(IAA)=IPL;SAVE VALUE
MC(IAA)=C1;MARK DEFINED
PN++;COUNT THE SUBS
RETURN
LOC 85;NUMBER LABEL
INL++
GOSUB RN;READ NUMBER
ILT(IRN)=IPL
ENDSUB
         SUB RN;READ NUMBER
IRN=+00000
WHILE
   CC=CC-X0
   IAA=CC
   IRN=IRN*I10
   IRN=IRN+IAA
   READ CC FROM F1
   GOSUB ZN
ON AA
```

*(Continued on next page)*

# Write Your Own Compiler (Listing III continued, text on pages 6-14)

```
ENDWHILE
ENDSUB
      SUB ST;STORE A NUMBER
QI++
CC=RI(QI)
CHOOSE ON CC
CASE XY;Y STACK
   YP++
   AA=SD<=YP
   IF AA
      MS 'Y OVERFLW'
      WRITE NL
      YP=SD
      FL=+000
   ENDIF
   IYS(YP)=ITU
CASE XZ;Z STACK
   ZP++
   AA=SD<=ZP
   IF AA
      MS 'Z OVERFLW'
      WRITE NL
      ZP=SD
      FL=+000
   ENDIF
   IZS(ZP)=ITU
CASE CP;'+' ADD TO S
   IAA=IYS(YP)
   IAA=IAA+ITU
   IYS(YP)=IAA
CASE CM;'-' SUB FROM S TOP
   IAA=IYS(YP)
   IAA=IAA-ITU
   IYS(YP)=IAA
CASE CS;'*' MULTIPLY
   IAA=IYS(YP)
   IAA=IAA*ITU
   IYS(YP)=IAA
CASE CG;'>'
   IAA=IYS(YP)
   AA=IAA<!ITU
LOC 12
   IF AA
      FL=+001
   ELSE
      FL=+000
   ENDIF
   GOSUB PY;POP Y STACK
CASE CL;'<'
   IAA=IYS(YP)
   AA=ITU<!IAA
   GOTO 12
CASE CU;'='
   IAA=IYS(YP)
   AA=ITU==IAA
   GOTO 12
CASE XV;SECOND STACK NUMBER
   IAA=IPT
   IAA++
   GOTO 39
CASE XD;DUMP IT--NULL OP
CASE XH;WRITE LOW BYTE IN HEX
   AA=ITU
   ITU=AA
   IAA=ITU/I16;HIGH HEXIT
   IBB=IAA*I16
   IBB=ITU-IBB;LOW HEXIT
   CC=IAA
   GOSUB WH
   CC=IBB
   GOSUB WH;WRITE HEXIT
DEFAULT;NUMBER OR ERROR
   CC=RI(QI)
   GOSUB ZN
   IF AA


      AA=CC-X0
   ELSE
      MS 'ILLG STOR'
      WRITE NL
      AA=C0


   ENDIF
   IPR(AA)=ITU
ENDCHOOSE
ENDSUB;ST
         SUB WH;WRITE HEXIT TO OUT BUF


AA=CC<=C9
IF AA
   CC=CC+X0
ELSE
   CC=CC+XA
   CC=CC-C9
   CC=CC-C1


ENDIF
BO(PB)=CC
PB++
ENDSUB;WH
         SUB WN;WRITE NUMBER INTO OUT BUFFER


GOSUB DS
WHILE
   IBB=ND
   AA=I00<!IBB
ON AA
```

```
CASE XI;INDIRECT TO MEMORY
   QI++
   DD=RI(QI)
   IBB=ITU
   QI++
   GOSUB FT
   CC=DD
   GOSUB ZN
   IF AA
      BB=CC-X0
   ELSE
   LOC 13
      MS 'BAD INDIR'
      MS 'ECT INDEX'
      WRITE NL
      BB=+000
   ENDIF
   AA=BB<!MK
   IF AA
      IAA=BB
      IAA=IAA+ITU
      IMI(IAA)=IBB
      RETURN
   ELSE
      BB=BB-MK
      AA=BB<!MK
      IF AA
         IAA=BB
         IAA=ITU+IAA
         AA=IBB
         MC(IAA)=AA
      ELSE
         GOTO 13
      ENDIF
   ENDIF
CASE XC;CONVERT TO STRING AND CAT OUT BUF
   IAA=ITU
   GOSUB WN
CASE XL;WRITE LOW BYTE OF FETCHED INTO OUTBUF
   AA=ITU
   BO(PB)=AA
   PB++
CASE XA;APPEND LOW BYTE OF FETCHED TO STRING BUFFER
   AA=ITU
   OS(PO)=AA
   PO++
   OS(PO)=C0
CASE XB;SET LENGTH OF STRING BUFFER
   PO=ITU
   OS(PO)=C0
CASE XG;SET GENERATOR
   IUU=ITU
CASE XU;FIRST STACK NUMBER
   IAA=IPT
LOC 39
   IAA++
   IST(IAA)=ITU
```

*(Continued on top of page 38)*

```
   ND--
   AA=DS(ND)
   BO(PB)=AA
   PB++
ENDWHILE
ENDSUB

         SUB ZA;CC IZ ALPHA!!!!!!!ASSUMES LINEAR
AA=CC-XA
BB=XZ-CC
AA=AA<=CV;25 (VINGT ET V)


BB=BB<=CV
AA=AA&BB
ENDSUB;ZA
         SUB ZH;TEST AND CONVERT HEXIT
AA=X0<=CC
BB=CC<=X9
AA=AA&BB
IF AA


   CC=CC-X0
   RETURN
ENDIF
AA=XA<=CC
BB=CC<=XF
AA=AA&BB
IF AA
   CC=CC-XA
   BB=+010
   CC=CC+BB
   RETURN
ENDIF
;AA IS NOW FALSE
ENDSUB;CH
         SUB ZN;CC IS NUMBER


AA=CC-X0
BB=X9-CC
AA=AA<=C9
BB=BB<=C9
AA=AA&BB
ENDSUB


         SUB ZW;CC IS WHITE SPACE
AA=CC==CB;BLANK
BB=CC==CT;TAB
AA=AA?BB
BB=CC==NL;NEWLINE
AA=AA?BB
ENDSUB
BOTTOM
```

**End Listing III**

# Dr. Dobb's Clinic

## We're Going to Answer Our Mail . . .

The Clinic continues to receive a gratifying flow of mail. Many of the letters reflect hours of work and thought. That isn't a requirement, by the way; we don't have space to do justice to all the beautiful analyses we're getting, while a good problem or discovery dashed off in ten minutes is very useful.

## Decimation Delayed

Dennis Allison politely points out that he described two solutions to the problem of dividing by ten in an article in *DDJ* #42. Readers have already produced yet a third class of solutions. We'll display the best algorithms for binary-to-decimal conversion in the next issue. If you're thinking about writing, read Dennis' article first.

## Beyond MAX

The solutions to MIN and MAX revolved around using the relational functions of BASIC to select one of two values. Michael Wochnik of St. Louis, MO extended that general principle to other functions. For example, suppose you wanted to examine bytes, filtering out ASCII control characters but keeping the printable ones. For simplicity, assume the bytes are carried as numbers. Then this function would do it:

```
DEF FNAS(X,C)=
    -((31<X) AND (X<127))*X
    -((X<32) OR (127<X))*C
```

If X is a valid, printable ASCII byte, it is returned by the function; if it isn't, the replacement character C is returned. Change the minuses to pluses if "true" is +1 in your BASIC.

Suppose that you are using an undependable value (user input) to select the target of an ON-GOTO. Bad input will cause an error and stop the program. This function will produce either the input number plus one, or an error number of 1:

```
DEF FNON(S,K)=
    S*(S>0)*(S<=K)+1
```

where S is the goto index and K is the number of valid alternatives. FNON would be used in code like this:

```
1000 . . . INPUT ACODE . . .
1010 ON FNON(ACODE,3)
       GOTO 1020,2000,3000,
       4000
1020 PRINT "from 1 to 3,
       please" : GOTO 1000
```

FNON is an interesting demonstration that logical "and" and arithmetical "multiply" are closely related. It's also interesting in that it works correctly regardless of whether "true" is -1 or +1, although that isn't immediately apparent.

## One More Flea-Hop

Several late responses came in regarding what John H. Martin of Cambridge, MA chose to call the "Cortesian" approximation, a groaner if there ever was one. Martin's letter makes it clear that the Intern was altogether too hasty in saying Edmund Butler's choice of constants was "probably the most accurate," and also explains why different readers' analyses of the problem produced different "best" constants.

For those who just tuned in, we proposed an approximation to the hypotenuse of a right triangle that can be stated as

happrox = x + y/2

where x and y are the lengths of the sides and x is greater than y. Several readers generalized it to

happrox = A*x + B*y

(our approximation falling out as A=1, B=0.5), and proposed "best" values for A and B, "best" meaning "least error." Martin points out that there are three different kinds of errors that one could consider, and that different values of A and B minimize different measures of error. He proposes three solutions:

.9285x + 3978y (a)
.9412x + 4101y (b)
.9605x + 3978y (c)

Where (a) minimizes the average absolute-value fractional error, (b) minimizes the root-mean-square fractional error, and (c) minimizes the worst-case fractional error. "I found most of these solutions by analysis," he writes, "but it might be an interesting exercise for your readers to look for them by computer. I considered writing an article on how one finds such solutions, but it will have to take a back seat to other work right now." All who want to read (or write!) an article on finding computer approximations raise their hands. An article is what it will be, because the Intern's math phobia is only barely under control now.

Russel Brown of Holland Patent, NY noted that the hypotenuse approximation implies a use of MIN and MAX, but a combined function could do it all at once. Readers with a strong liking for algebra might like to verify that if

```
happrox=A*MAX(ABS(X),ABS(Y))+
        B*MIN(ABS(X),ABS(Y))
```

(choosing A and B from Martin's list), then if you pre-calculate constants

A2 = (A+B) / 2
B2 = (A-B) / 2

you could define a hypotenuse function this way:

```
DEF FNHYP(X,Y) = A2*(ABS(X)+
    ABS(Y)) + B2*ABS(ABS(X)-
    ABS(Y))
```

## A System Solution for Apple

We descend from the rarified air of mathematics to find a letter from J. G. Wendel of the University of Michigan. He writes of a discovery he made about MBASIC running on the Apple with a Z80 Softcard. Suppose you're working on a program that does lots of printing. During development you have it print to the screen, but for the production version you want all the PRINTs to go to the hard-copy printer. Wendel says "In Applesoft you could add a line PRINT D$"PR#1" (where D$ = ctrl-D) at the start of the program, and . . .#0 at the end."

Not so under MBASIC which doesn't recognize the execution of DOS commands through PRINT statements. However, "In Z80 MBASIC add POKE 3,150 at the beginning and POKE 3,149 at the end. This in effect converts all PRINTs to LPRINTs. Note: if you forget the final POKE your keyboard will be locked out!"

## A Glitch in the Z80 CPU

Wayne Farmer of San Diego, CA passed on an item from *UZER*, the Zilog User's Group newsletter. It describes the kind of subtle hardware problem that could give fits to someone trying to implement MP/M or another multi-tasking operating system on the Z80. Critical code in such a system might need to preserve the state of the system, including the state of the interrupt-enable flag in the CPU. The usual method to read that flag is to execute either the "LD A,I" or the "LD A,R" instruction; both set the carry flag to the value of the interrupt-enable latch.

According the *UZER*, there is one rare condition under which that isn't true. If interrupts are enabled when the instruction is done, and if an interrupt occurs at just that instant, the carry flag is set falsely. Solutions are given in the

newsletter; they entail using the stack to detect that an interrupt happened at the crucial moment. The one programmer in the thousand who gets hit by this might spend weeks trying to fix an unreproducible system hang.

## Two Calls for Help

Turner Caldwell of Burgess, VA writes of a problem with the popular MX-80 printer. Caldwell says the printer is, in general, an excellent piece of gear. Unfortuantely, it may be a bit too smart for its own good – or does it have a bug? He notes that many of the machine's features are switched on and off by sending it control codes. "There is a catch, however. To speed up printing, the control logic is bi-directional and seeks the shortest way to print a line. The CR code tells the printer to print the data in the buffer. Thus a control code embedded in the text may be recognized or not, depending on the direction the print head is directed to go."

We assume Caldwell is talking about situations where some control change (to wide characters, for example) is triggered by a control code leading the data. If the printer is going right to left, what was a leading control sequence is read as a trailing one and affects the wrong characters or no characters at all. "There is no way of casually determining this during entry of data with a word processor," he writes, "so except for special situations many of the features of the printer seem unuseable." Can this be true? Is it true of other printers with clever internal logic? Is there an easy solution?

Meanwhile, J. R. Snow of San Francisco, CA wants to decode Touch-Tone pulses from a phone line in order to drive a single-board computer from, say, a pay phone. The problem is to keep the cost down; Touch-Tone decoders run close to $200, it seems and the decoder still has to be interfaced after that. An experienced homebrewer might have found something clever along these lines. A solution might fit the Clinic, or it might well justify a full article.

## Cut The Cards

We have a number of Atari game tapes around, including three card games: Stud Poker[1], Bridge 2.0[2] and Casino Blackjack[3]. We were poking around in the code of the games and noticed something odd about the way they

### Listing I

```
{   Pascal fragment illustrating card shuffling as done in Dynacomp's
    STUD POKER for Atari   }

type card = 1. .52;

var deck: array[card] of boolean;

procedure shuffle;
    var r: card;
    begin
        for r:=1 to 52 do
            deck[r] := true
    end {shuffle} ;

function deal_a_card: card;
    var r: card;
    begin
        repeat
            r := fnrandi(52)
        until deck[r] ; {unpredictable!}
        deck[r] := false;
        deal_a_card := r
    end {deal_a_card} ;
```

### Listing 2

```
{Pascal fragment illustrating card shuffling as done in Manhatten Software's
 CASINO BLACKJACK for Atari. In Blackjack suits are ignored. The original
 code handles 1, 2, or 4 decks. This fragment assumes a 2-deck pack. }

const ndecks = 2;
    decksize = 104; {52*ndecks}
    cards_of_a_rank = 8; {decksize/13}

type card = 1. .13;

var ranks: array[card] of 0. .cards_of_a_rank;
    deck: array[1. .decksize] of card;

procedure shuffle;
    var r: card; j: 1. .decksize;
    begin
        for r:=1 to 13 do
            ranks[r] := cards_of_a_rank;
        for j:=1 to decksize do
            begin
                repeat
                    r := fnrandi(13)
                until ranks[r] > 0; {unpredictable!}
                ranks[r] := ranks[r]-1;
                deck[j] := r
            end
    end {shuffle};
```

shuffle their simulated decks.

We've known fanatic cribbage players who swore that using a cribbage deck for any other game "spoiled the deck" for many hands afterward. There may be a grain of truth to this; some order will be preserved when the deck is gathered in at the end of any card game. Card shuffling is not a random process; it can be seen as the systematic permutation of a starting sequence to an ending sequence. The ending sequence retains a "memory" of the original sequence; the permutation can be inverted, at least in principle.

Two of the three games do not treat the simulated deck as a sequence to be permuted. We've recast the logic of the three games into Pascal fragments, just for a change. In all listings, "fnrandi(j)" is some Pascal equivalent of the BASIC expression

INT(RND(0)*J)+1

Listing 1 (page 41) abstracts the logic used in Stud Poker. a "shuffle" resets a vector of flags that indicate whether a card has been dealt or not. To deal a card, the game generates random integers until it finds a card it hasn't dealt before. That takes an unpredictable number of iterations, but Stud Poker never deals more than 10 cards before shuffling so the odds are good that it will only loop once before finding a card.

Casino Blackjack has to deal out an entire pack of cards before shuffling, yet it uses a variant of the same logic (Listing 2, page 41). Suits are ignored in blackjack, so Casino Blackjack works with a deck that contains multiple occurrences of each rank rather than unique card values. The boolean vector of Stud Poker is replaced with a vector that counts the number of each rank that have been dispensed to the new deck. Like Stud Poker, the game picks random integers until it finds a rank that hasn't been exhausted. All the cards must be picked. In execution it is clear that the program "sticks" in the random-choice loop for several seconds near the end of the shuffle.

Bridge 2.0 does treat a deck as a permutation sequence. The resulting algorithm (Listing 3, below) has two advantages. It has a predictable execution time and, since the next deal is a permutation of the prior one, it may produce hands that are more realistic. The use of a descending index to limit the range of the random choice is especially clever. The loop invariant assertion is "all cards j+1 to end are shuffled, all cards 1 . . . j are unshuffled." Comments on card-shuffling, and on the difficulty of coding "fnrandi" in Pascal, are welcome. DDJ

**Footnotes**

[1] STUD POKER by Jerry White, published by Dynacomp Inc., Rochester, NY.

[2] BRIDGE 2.0 (author unknown), published by Dynacomp Inc., Rochester, NY.

[3] CASINO BLACKJACK by S. Silverman, published by Manhatten Software, Pacific Palisades, CA.

## Listing 3

```
{Pascal fragment illustrating card shuffling as done in Dyna-
 comp's BRIDGE 2.0 for Atari. A deck is permuted by
 sweeping from high end to low, swapping the rightmost
 unshuffled card with a random lower unshuffled card.  }

type  card = 1. .52;
      deck = array[card] of card;

procedure shuffle( var d:deck );


    procedure swap( var a,b : card );
        var t: card;
        begin
            t:=a; a:=b; b:=t
        end {swap};


    var r,j : card;
    begin {shuffle}
        for j:=52 downto 1 do
            begin
                r:=fnrandi(j); {note declining limit}
                swap(deck[r],deck[j])

            end
    end {shuffle};
```

*DDJ's* responsive readership
can give you invaluable feedback
on your ideas and innovations.
Find out how you can reach them
by contacting us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# Of Interest

by Dave Cortesi

## Two Useful Workshops

. . . which might interest a few readers. The National Bureau of Standards will hold a three-day working seminar on the measurement of electromagnetic interference in Boulder, Colorado August 25-27. The seminar appears to address the measurement methods needed to certify a new computer to the FCC. To register at this late date, you'd better call the organizer, **M. Gerald Arthur.**

You've more time to prepare for what **Virginia Tech** claim to be very popular workshops on microcomputer interfacing. Each three-day workshop has the participants designing and testing actual hardware. The titles and dates are: Digital Electronics for Automation and Instrumentation, Oct. 26-28; Microcomputer Interfacing and Programming with the Z80/8085, Oct. 29-31; and Scientific Instrument Automation and Interfacing with the TRS-80, Nov. 2-4.

## MUMPS for 8080 and Apple

It took heroic restraint to make a straight headline out of the acronym MUMPS, but the product is earnest enough. MUMPS is a language and operating system that began life as one of the earliest serious applications of minicomputers (remember them?). The national **MUMPS Users' Group** claims that since the language was made an ANSI standard in 1977 it "has been used for virtually all types of decision support . . . and information systems," although its heaviest use remains in the medical arena where it began.

The School of Medicine at UC Davis has been distributing successive versions of an ANSI-standard implementation of MUMPS for the 8080 CPU and CP/M for some time now. They sent us a comparison chart that shows the latest version running two orders of magnitude faster than the first version, so they are working hard. Now they have set up a Users' Group for 8080 MUMPS. Members will get automatic updates of the system and a newsletter for $93 a year. To join, contact **Richard F. Walters, Ph.D.**

The same developers have prepared a version of 8080 MUMPS on Apple disks, for use with an Apple with Z80 Softcard. **Larry L. Stoneburner, M.D.** is handling the distribution of this version.

## Small David Selects Stones, Eyes Giant

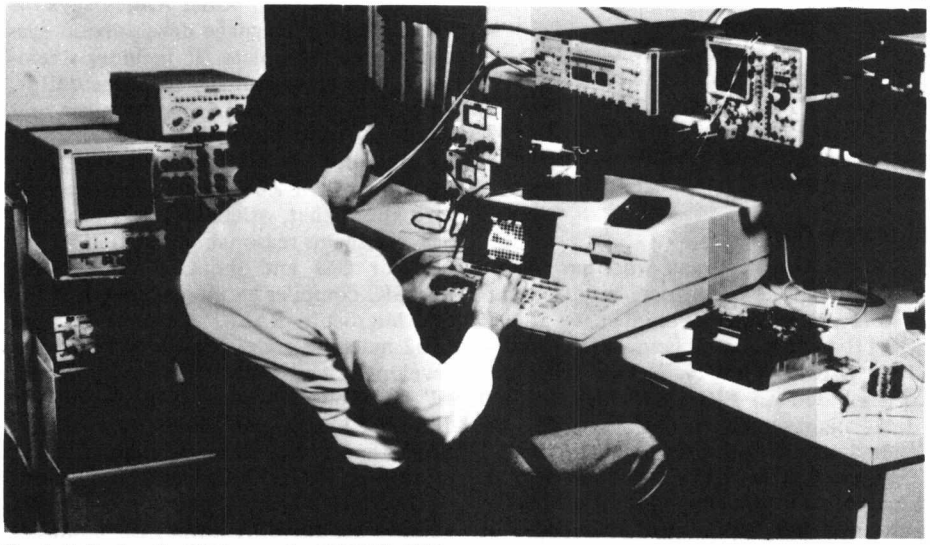
That's the image that entered our mind on reading a release from Tandy Corporation. They've announced three software products that place the TRS-80 Model II in head-on competition with products of IBM's Data Processing Division. The least is a diskette reformatter that lets the Model II work with diskettes formatted for IBM's 3741, which "in essence . . . delivers the . . . equivalent of an IBM workstation for substantially less than the cost of IBM equipment."

The second product is a communications program which provides Binary Synchronous Communications using the protocol of the IBM 2780 and 3780 remote batch workstations. That protocol is something of a de facto standard, being supported by DEC (and by OASIS; see last issue) as well as by IBM's batch operating systems. The real zinger is a second communications package that lets the Model II emulate the IBM 327x display terminal and so hook up to any of those famous initials TSO, CICS, IMS, VSPC, and VM/370.

All three products are to be sold under a site license that allows free duplication within an installation. This suggests that Tandy are thinking of corporate customers that might buy truckloads of Model II machines. The whole announcement presents an interesting challenge not only to IBM but to a number of companies that make 327x look-alike terminals. Interesting times.

## HP and the 68000

Hewlett-Packard has announced a new desk-top computer, the 9826, which is the first commercial product we've seen to use the M68000 CPU chip. The machine resembles the 80 series in size and styling, with a 5" diskette drive in place of the tape unit. The CPU is an M68000 operating at 8MHz; storage starts at 64KB and can go to 512KB. Your choice of BASIC, Pascal, or something called HPL is built in. The 7-inch CRT has graphics resolution of 400 by 300 pixels. The diskette holds 264 KB, and that's unique: the mass storage might be smaller than the RAM! A number of interface cards are to be available. The system costs around $9000, depending on options.



**Hewlett-Packard's 9826 uses the M68000 CPU.**

## Some Information Sources

Press releases told us of a few unique sources of information this month. We hold back only the title of the book that claims to show us how astrology can direct our financial decisions; *after* we've raided the stock market under its guidance we'll let you in on it.

**Datapro** have surveyed 4200 computer users to find out, in detail, how they like their systems. The report includes responses on micro-based systems and it is interesting to compare the responses of micro users to those of minicomputer and mainframe users. Here's a quote from the report: "We asked the users, 'Have you integrated word processing functions on this computer?' A surprising 55% of the . . . personal . . . computer users said 'yes.' Twenty-five percent of the minicomputer users indicated that they had

integrated word processing functions on their system, as did 11 percent of the mainframe users." Many other illuminating differences can be found; anyone engaged professionally with micros might find the report worth its $25 price.

**InfoPro Systems** publishes "the only industry-wide directory of UNIX and C products." The sample they sent us lists thirty-odd contact points for suppliers and user groups, each annotated with comments having a tone not unlike that used in this column. Clearly an outstanding buy at $18 for a year's subscription. Only thing, the release neglects to state how many issues a year's subscription includes.

An organization called **Associated Technology**, in an admirably succinct release, offers a free, 7-page report on careers in consulting for engineers and computer scientists. The report is said to be "devoted to dispelling common misconceptions and identifying successful career paths."

The IEEE Computer Society has released a catalog of the 300 books and reports they publish. The catalog displays a feast of technical know-how; it's great reading for catalog lovers. Write to the **Computer Society Press** and request the "1981 Pubs Catalog."

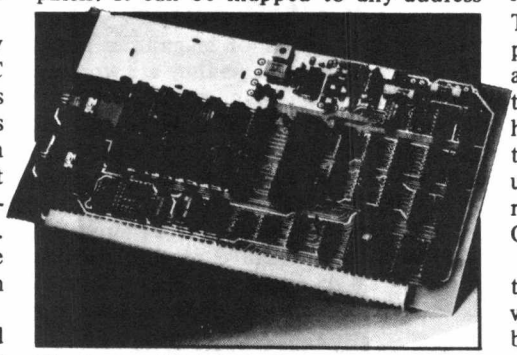
## Books on the 68xx Machines . . .

**Group Technology, Ltd.** have published three books by Andrew Staugaard, Jr., a man who certainly must know the 6800 series of chips very well indeed. The three titles (with order numbers and prices in parens) are: *6809 Microcomputer Programming and Interfacing With Experiments* (21798, $13.95), *6801, 68701, 6803 Microcomputer Programming and Interfacing* (21726, $13.95), and *How to Program and Interface the 6800* (21684, $15.95).

## . . . and Hardware for Them

A very satisfying number of announcements relating to 6800 systems and the SS-50 bus came in this month (so how come nobody submits articles on these fine machines?). Aside from the books mentioned above, we heard of a color graphics board, a universal interface, and a 64K RAM card.

**Percom** have produced a color video display generator for SS-50 and SS-50C systems. The card is memory-mapped, so they call it the ColoRAMa; we'll forgive them for that because it sounds like a very capable device. It is claimed to support 11 different display formats yielding resolutions from 64x64 to 256x192 pixels. It can be mapped to any address



**The Percom "ColoRAMa" color graphics board for SS-50 systems.**

space bank, has provision for a TV modulator, a cassette interface (!), and a 2K EPROM. The manual includes both 6800 and 6809 listings of a "display operating system." The package is priced at $219 with 1KB of display memory.

The "Universal Interface Card" is aimed at homebrewers. It is a bare I/O card designed for the SS-50C I/O bus, with circuit pads provided for an on-card voltage regulator, connectors, and two ACIA or one PIA chip. The board with a set of bus connectors costs $17 from **Quality Research**.

**Gimix** have added an unusual 64KB RAM board to their line of SS-50C products. The board uses very low-power CMOS chips and contains a battery for backup so that data is retained when the system is powered down. The board is claimed to be capable of operation at 2MHz with no wait states. It can be write-protected by a switch and so used to emulate a PROM in development. The price is $1088, quite reasonable for 64K of static storage.

## Software for OASIS . . .

**Phase One Systems** who have been cranking out new products monthly for so long it's getting indecent, have added FORTH to the line of languages supported by OASIS. They claim that their version is based on FIG standards, and the "data management functions can be handled at the operating system level." They do not say that the dictionary is sharable and re-entrant, so presumably each user has his or her own copy, a minor disadvantage.

Phase One also wrote to tell us that, "avoiding possible confusion with a mainframe package of the same title," their relational database system has been renamed from RDBMS to CONTROL. Then **Quaydata** wrote to tell us that they produce CONTROL, and that they've added a COBOL Program Generator to the BASIC Program Generator it already had. With the package, they claim, "a typical application requires about 40 minutes from screen design to compiled and running object code." Bet it isn't the Last One, either.

Finally, the **OASIS Users' Group** notified us that they've compiled their first volume of public domain software. Membership is $35 a year.

## . . . And for Five Other Systems

For Hewlett-Packard desktop machines: **Structured Software Systems** has a line of BASIC enhancers for the various 9835 and 9845 models, including a CRT to printer dump, a cross referencer, a memory mapper, and others. Prices range from $150 to $300.

For CP/M systems: there are several un-eraser programs available, including one from **Elliam Associates**. But that organization has thought into the problem more deeply and has come up with CONFLICT, which checks out the present state of all disk blocks of an erased file, reporting where the erased space is now part of an active file and where it has been part of another erased file. CONFLICT is $35, or $60 with Elliam's UNERA command.

For TRSDOS: **Allen Ashley** says his PDS assembly language development system is now complete. It includes a relocating macro assembler, a linkage editor, an editor, a trace and debug monitor, and a number of utilities. One caution: Allen has plumped for extended Intel mnemonics rather than Zilog ones. Some people care about that, others don't. The whole package seems reasonably priced at $99.

For OSI and the OS65D3 system: FBASIC compiles an integer subset of the OSI-Microsoft BASIC language to true 6502 machine code. **Pegasus Software**, its developers, call it the "first true compiler" for the OSI systems. The compiler is disk-based and can compile programs larger than the available storage. It is said to support WHILE loops, and GOTOs and GOSUBs to absolute addresses, an idea we find repulsive. Anyway, the compiler costs $155.

For Heath/Zenith H/Z-89 systems: **Zenith** sent us two releases. In the first, they said that the price of the Zenith Electronic Typing word processing package was to be reduced from $395 to $295. In the second, they announced that both Zenith and Heath would be marketing MicroPro's WordStar for the x89. The rest of the MicroPro line – MailMerge, DataStar, SuperSort – will also be available from Heath and Zenith later this year. DDJ

## Contact Points

**M. Gerald Arthur**, Electromagnetic Fields Division, National Bureau of Standards, Boulder, CO 80303; (303) 497-3603.

**Allen Ashley**, 395 Sierra Madre Villa, Pasadena CA 91107; (213) 793-5748.

**Associated Technology**, POB 19, Manchester, TN 37355; (205) 837-4718.

**Computer Society Press**, POB 639, Silver Springs, MD 20901.

**Datapro** Research Corp., 1805 Underwood Blvd., Delran, NJ 08075.

**Elliam Associates**, 24000 Bessemer Street, Woodlands Hills, CA 91367.

**Gimix**, Inc., 1337 West 37th Place, Chicago, IL 60609; (312) 927-5510.

**Group Technology, Ltd.**, POB 87, Check, VA 24072; (703) 651-3153.

**InfoPro Systems**, POB 33, East Hanover, NJ 07936; (201) 625-2925.

**MUMPS Users' Group**, 1001 Connecticut Ave. N., Suite 800, Washington, DC 20036.

**OASIS Users' Group**, POB 2400, Santa Barbara, CA 93120; (805) 965-0265.

**Pegasus Software**, POB 10014, Honolulu, Hawaii 96816.

**Percom** Data Company, 211 N. Kirby, Garland, TX 75042; (214) 272-3421; toll free orders to 800-527-1592.

**Phase One Systems**, 7700 Edgewater Drive, Suite 830, Oakland, CA 94621; (415) 562-8085.

**Quality Research** Company, POB 7202, Spokane, WA 99207.

**Quaydata** Inc., 2415 15th Street, Suite 301, Denver, CO 80202; (303) 433-5476.

**Larry L. Stoneburner, M.D.**, 280 Hospital Circle #202, Westminster, CA 92683.

**Structured Software Systems**, Inc., Box 1072 Irick Road, Mt. Holly, NJ 08060; (609) 267-1616.

Dr. Lindy Leffel, C.E.C., **Virginia Tech**, Blacksburg, VA 24061; (703) 961-5241.

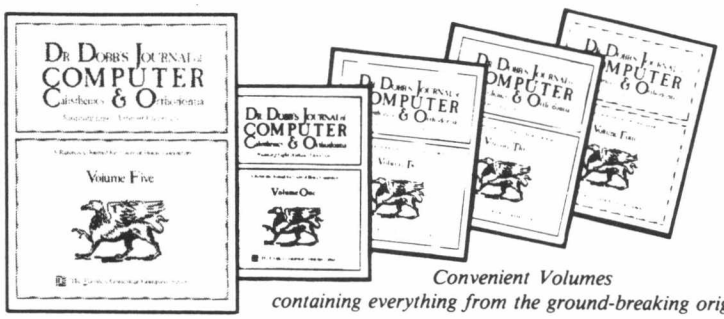
**Richard F. Walters, Ph.D.**, Department of Community Health, University of California, Davis, CA 95616.

**Zenith** Data Systems, 1000 Milwaukee Avenue, Glenview, IL 60025; (312) 391-8181.


For Your Personal

# REFERENCE COLLECTION

## Thousands of dollars' worth of microcomputer information and listings!




*Convenient Volumes containing everything from the ground-breaking originals*

**Dr. Dobb's Journal Vol. V** – All the ground-breaking issues from 1980 in one volume! Systems software reached a new level with the advent of CP/M, chronicled herein by Gary Kildall and others (*DDJ*'s all-CP/M issue sold out within weeks of publication). Software portability became a topic of greater import, and *DDJ* published Ron Cain's immediately famous Small-C compiler – reprinted here in full!

Contents include: The Evolution of CP/M, a CP/M-Flavored C Interpreter, Ron Cain's C Compiler for the 8080, Further with Tiny BASIC, a Syntax-Oriented Compiler Writing Language, CP/M to UCSD Pascal File Conversion, Runtime Library for the Small-C Compiler, and, as always, even more!

**Dr. Dobb's Journal Vol. I** – The material brought together in this volume chronicles the development in 1976 of Tiny BASIC as an alternative to the "finger blistering" front-panel machine-language programming which was then the only way to do things. This is always pertinent for bit crunching and byte saving, language design theory, homebrew computer construction and the technical history of personal computing.

Topics include: Tiny BASIC, the (very) first word on CP/M, Speech Synthesis, Floating Point Routines, Timer Routines, Building an IMSAI, and more.

**Dr. Dobb's Journal Vol. II** – 1977 found *DDJ* still on the forefront. These issues offer refinements of Tiny BASIC, plus then state-of-the-art utilities, the advent of PILOT for microcomputers and a great deal of material centering around the Intel 8080, including a complete operating system. Products just becoming available for reviews were the H-8, KIM-1, MITS BASIC, Poly Basic and NIBL.

Articles about Lawrence Livermore Lab's BASIC, Alpha-Micro, String Handling, Cyphers, High Speed Interaction, I/O, Tiny Pilot & Turtle Graphics, many utilities, and even more.

**Dr. Dobb's Journal Vol. III** – The microcomputer industry entered its adolescence in 1978. This volume brings together the issues which began dealing with the 6502, with mass-market machines and languages to match. The authors began speaking more in terms of technique, rather than of specific implementations; because of this they were able to continue laying the groundwork industry would follow. These articles relate very closely to what is generally available today.

Languages covered in depth were SAM76, Pilot, Pascal and Lisp, in addition to RAM Testers, S-100 Bus Standard Proposal, Disassemblers, Editors and much, much more.

**Dr. Dobb's Journal Vol. IV** – This volume heralds a wider interest in telecommunications, in algorithms, and in faster, more powerful utilities and languages. Innovation is still present in every page, and more attention is paid to the best ways to use processors which have proven longevity – primarily the 8080/Z80, 6502, and 6800. The subject matter is invaluable both as a learning tool and as a frequent source of reference.

Main subjects include: Programming Problems/Solutions, Pascal, Information Network Proposal, Floating Point Arithmetic, 8-bit to 16-bit conversion, Pseudorandom Sequences, and Interfacing a Micro to a Mainframe – more than ever!

**YES!** ☐ Please send me the following Volumes of *Dr. Dobb's Journal* for $20.75 each.
☐ ALL 5 for ONLY $88, a savings of over 15%!

Please charge my: ☐ Visa ☐ MasterCard ☐ American Express
I enclose ☐ Check/money order

Card # _____ Expiration Date _____

Signature _____

Name _____ Address _____

City _____ State _____ Zip _____

*Prices subject to change.*

**Mail to: Dr. Dobb's Journal, P.O. Box E, Menlo Park, CA 94025**
Allow 6-9 weeks for delivery.

Vol. I ____ x $20.75 = ____
Vol II ____ x $20.75 = ____
Vol. III ____ x $20.75 = ____
Vol IV ____ x $20.75 = ____
Vol. V ____ x $20.75 = ____
ALL 5 ____ x $88.00 = ____
Sub-total $____

**Postage & Handling** ____
**Must be included with order.**
*Please add $1.25 per book in U.S. ($1.75 each outside U.S.)*

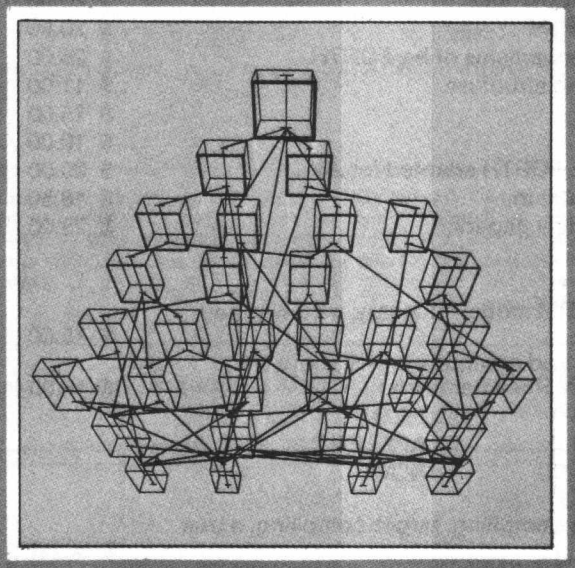
**TOTAL** $____

$2.50
($3.00 in Canada)

# DR. DOBB'S JOURNAL

For Users of Small Computer Systems

Number 59 September 1981 Volume 6, Issue 9



**The FORTH Philosophy**
*Kim Harris*

**A FORTH Assembler for the 6502**
*William F. Ragsdale*

**A Screen-Oriented Editor in FORTH**
*Henry Laxen*

*plus:*
**FORTH Data Base Design,**
**a Decompiler...and even more!**

## Special FORTH Issue!

## Utilities & Tutorials

## Software Tools

# DR. DOBB'S JOURNAL
For Users of Small Computer Systems

September 1981 Volume 6, Issue 9, Number 59


# CONTENTS


**The FORTH Philosophy** ..... 6
*by Kim Harris*

**A FORTH Assembler for the 6502** ..... 12
*by William F. Ragsdale*

**Sallying FORTH to Battle** ..... 25
*by R. G. Loeliger*

**A Screen-Oriented Editor in FORTH** ..... 27
*by Henry Laxen*

**Elements of a FORTH Data Base Design** ..... 42
*by Glen B. Haydon*

**A FORTH Decompiler** ..... 48
*by Ray Duncan*

**The FORTH Alternative** ..... 57
*by Mitchell E. Timin*

Letters ..... 4
Of Interest ..... 54
Dr. Dobb's Clinic ..... 60



**Publisher** Michael Madaj
**Editor** Marlin Ouverson
**Assistant Editor** Julie Anton
**Contributing Editors** Dave Caulkins, Dave Cortesi, Kim Harris
**Art Director** Clifford West
**Artist** Barbara Ruzgerian
**Typesetter** Renny Wiggins
**Proofreader** Nancy Heubach
**Marketing Director** Craig S. Harper
**Advertising Sales** Kent Safford
**Circulation Manager** Peter Clark
**Circulation Assistant** Leah Dansby
**Business Manager** Maureen Christine



*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025.



© 1981 by People's Computer Company unless otherwise noted on specific articles. All rights reserved.


# EDITORIAL

*Dr. Dobb's* is pleased to present this all-FORTH issue. The material should prove of interest both to FORTH followers and to the uninitiated. We believe it presents a rounded viewpoint, although as one of our frequent contributors noted, those who do not like the language lose interest quickly, also losing the desire to express any opinion on it.

Not so the FORTH fans! During early planning stages, we noted that many FORTH people possess a trait in common: they proselytize. They are fanatic, born-again programmers. As work progressed it became clear that these folks realize their own addiction.

Is this attitude, this conviction of rightness, a defense mechanism for the benefit of members of the C, Pascal, or other language community? Or is it the natural result of being in on a good thing? To answer that question would be an invitation to mayhem. Study this issue well and decide for yourself!

I mention all this merely to outline the peculiar emotional tone of the FORTH movement. It is something which inspires strong feelings for or against. There will be many more important articles to come on the topic. Some will emphasize the strengths, and others the weaknesses, of FORTH. Each will shed a little more light for the reader to more clearly discern his own interest, to better understand the utility of software written in FORTH.

There has always been a plethora of voices willing to criticize what they feel is wrong with other people's thoughts on FORTH, but very few who, as Doc sometimes says, are willing to "put their foot where their mouth is." Critics abound, while competent authors such as this month's contributors come only in small handfuls. We do not claim this material is a definitive study of FORTH, for there is no unified, definitive opinion of the language. Some, in fact, do not even believe it is a language!

A few excellent articles were received which are not included in this issue simply because space did not allow it. We will be using them as "seed" material in what we hope will become an ongoing dialogue and exploration of FORTH. The subject is not closed! – *MDO*


**Subscription Rates:** $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. dollars, drawn on a U.S. bank.

**Writer's Guidelines:** All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

**Donating Subscribers** **Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives *five* one-year subscriptions).

**Contributing Subscribers:** Eric H. Johnson, Jerry Horn, Edward W. Carr, Steven Fisher, Thomas H. Grohne, G. V. Elkins, Robert M. Connors, William J. McEown, Robert C. Luckey, Kim Leary, Robert N. Stabler, DeWitt S. Brown, John B. Palmer, E. Glion Curtis, Jack Miller, Burks A. Smith, Transdata Corporation, Mark Ketter, John W. Campbell, Friden Mailing Equipment, Howard E. Decker, Jim Wilson, Frank Lawyer. **Retaining Subscribers:** Osamu Shimizu, Scott B. Guthery, Robert L. Edens, Titus Purdin. **Sustaining Subscriber:** Gordon Eubanks, Jr. **Lifetime:** Michael S. Zick.

**Foreign Distributors** **UK & Europe:** LP Enterprises, 8/11 Cambridge House, Cambridge Road, Barking, Essex, 1G11 8NT, England (Tel. 01-591-6511, TLX 892395). Hofacker-Verlag, Tegernseer Strass 18, D-8150 Holzkirchen, West Germany. **Asia & Australia:** Computer Store, POB 31-261, 22B Milford Road, Milford, Auckland 9, New Zealand. ASCII Publishing Co., Ltd., Aoyama Building 5F, 5-16-1 Minami Aoyama, Minato-ku, Tokyo 107, Japan.

# THE FORTH PHILOSOPHY

## The Abstract

FORTH is an unconventional software package which is easily misunderstood. That is because it is based on a philosophy which is moderately different from that of traditional programming systems. This article describes a set of goals, assumptions, and techniques which are practiced by most experienced FORTH programmers. Part I discusses the philosophy and part II (to be published in a following issue of *DDJ*) relates FORTH's implementation of this philosophy. FORTH is described as a unified approach to programming which provides a complete environment for the execution or the development of programs. Its goals are high quality, simple, and general software systems which can solve a wide range of systems and applications problems.

## The Introduction

FORTH, another computer software package, has recently come to the attention of the small computer industry. *DDJ* has published many articles in the last two years, and there was a big "splash" in the August 1980 issue of *Byte* magazine. A steady stream of articles has followed in several magazines. Software products are available for a large number of hardware processors from a variety of vendors. They range from cheap, hobby-grade systems to sophisticated, professional systems. It is quite unconventional and therefore easily misunderstood. Outrageous claims have been made about FORTH:

- Ten times better programmer productivity compared to conventional software systems
- Fast execution: 90% to 50% of machine language
- Small programs: memory size is 1/2 that required by equivalent assembly language programs
- Programs are transportable, independent of the hardware processor

FORTH has been called many things:

- A language (both high and low level)
- An operating system
- An interpreter and a compiler
- A religion, a cult, and an addiction
- Unreadable, hard to learn, and only for geniuses
- A quantum jump in software evolution, and a mistake

It can be all of these things! This article will attempt to explain this enigma called FORTH in terms of its goals, tradeoffs, and achievements. Many statements are the opinion of the author; they describe his current view of FORTH.

FORTH was invented in the late 1960's by Charles Moore, who was trying to increase his own productivity as an applications programmer.[1] It evolved out of his dissatisfaction with conventional approaches. It is a *unified* approach to developing software which solves practical problems. Its "rules" are few and are applied uniformly throughout the system. An example of the unified aspect of FORTH is its use of a single language to bridge the gap between some hardware and an application. The conventional approach is to use several languages which each span a small portion of the gap (see Figure 1).

A FORTH software system also supplies the total environment to develop or execute programs. It contains its own operating system, interpreter for interactive execution, compiler, assembler, editor, and utilities. All these parts are normally resident in a development system, and are written in FORTH's own language. FORTH is both a usable system and a meta system (i.e., it describes itself).

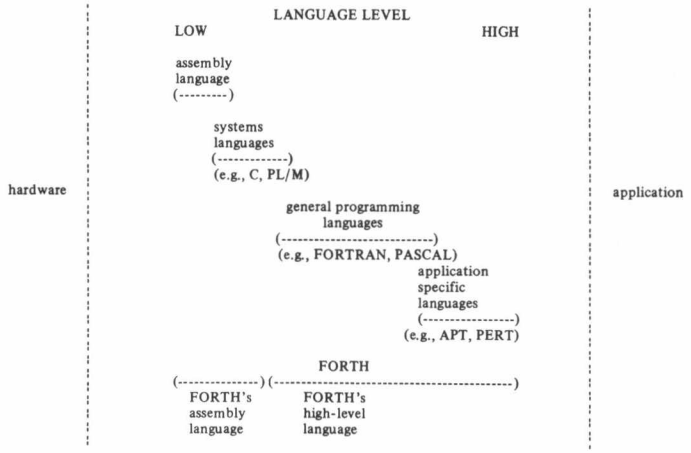



**Figure 1 – The Software Gap**

Mr. Moore wanted a simple, general, and frugal software system which would allow him to implement a program in a direct way and would execute it efficiently. After continuous refinements during the last decade, FORTH has developed a set of tradeoffs which benefit a wide range of applications areas. Here is a list of areas in which FORTH has been used:

- Mathematical, numerical applications
- Systems programs, compilers, operating systems
- Scientific, simulation problems
- Real-time acquisition and control, robotics
- Business, accounting
- Information processing, data base systems
- Communications, networks
- Graphics, image, and speech processing
- Art, music

Because the system is "extensible" (i.e., any part of it may be modified), FORTH continues to evolve. Many variations have been tried. The better results are adopted by the FORTH community voluntarily, based on merit. This is "natural selection" applied to software. A voluntary standard has helped achieve substantial uniformity at the applications program level across different FORTH implementations.[2]

## The Elements of FORTH

To understand FORTH, one must look at two complementary aspects. FORTH is a *philosophy* and an *implementation* of that philosophy. The philosophy gives the goals and the techniques used in the implementation. It is easier to describe an implementation than a philosophy, so most previous articles on FORTH have concentrated on implementation details. But that is like knowing how to build a clock, and not understanding the concept of time; or learning what a needle and thread are for, but not knowing how to sew. The philosophy gives the "rules of usage" for FORTH, and it explains the rationale behind its structure. The philosophy applies to technical activites (e.g., program organization, style) and to

**by Kim Harris**

*Kim Harris, Forthright Enterprises, P. O. Box 50911, Palo Alto, CA 94303.*

the management of projects.

## Basic Assumptions

The origins of FORTH philosophy are the Scientific Method and some human factors. The Scientific Method is a technique for understanding complex natural systems. It has been used for hundreds of years, and a key element was a belief of ancient Greek philosophers. The method is iterative: it is a never-ending cycle of discovery and refinement. A diagram of the method is shown in Figure 2.

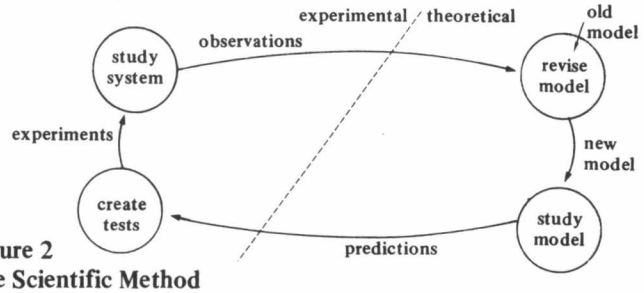



**Figure 2**
**The Scientific Method**

It first studies a natural system and gathers observations about its behavior. Then the observations are modeled to produce a theory about the natural system. Next, analysis tools are applied to the model which produces predictions about the real system's behavior. Experiments are devised to compare actual behavior to the predicted behavior. The natural system is again studied, and the model is revised. The *goal* of the method is to produce a model which accurately predicts all observable behavior of the natural system. But it is likely that several different models will meet the goal. Which model should be chosen? The answer has always been, "Choose the simplest!" This is the principle of "parsimony":

Given two adequate theories,
the "correct" one is the simpler.

This principle is at the heart of FORTH's philosophy. From the Scientific Method, FORTH has borrowed the following concepts:

Specialized tools
Modeling to understand complex systems
Simplicity to guide decisions
Repetition to achieve goals.

A software development project attempts to solve a problem by producing a solution which includes a software system. Any acceptable solution must meet a set of functional requirements and a set of constraints. Constraints are negative requirements (e.g., compatibility, performance, interface specifications). The more requirements a project has, the more difficult it will be to meet them all.

A particular problem has some fixed requirements and many flexible ones (i.e., unspecified or loosely specified). When a system gets designed for execution on a computer, additional constraints must be imposed (e.g., sequential execution, control structures). Additional constraints are imposed when it is implemented (e.g., finite resources, discrete arithmetic). These constraints are "artificial" in that they did not come with the original problem.

It is important not to impose any more artificial constraints than necessary in order to aid productivity. Then more room is left for tradeoffs which can produce a more desirable result. These views come directly from the belief that simplicity is better than complexity. A FORTH development system has a minimum of rules (e.g., almost no syntax, universal control interfaces for executing procedures, general data interfaces between procedures), and minimum restrictions (e.g., no prohibitions, no impenetrable boundaries, little built-in error checking). The result of this unconventional approach is maximum control by the programmer, extensibility (i.e., the ability to change, add to, or remove any part of the system), and transportability (i.e., programs are independent of the hardware, except for resource needs).

During a development project, many decisions must be made (e.g., tradeoffs between speed and size, the user's interface, robustness). Wiser decisions can be made if we could know the relative costs of each solution. Software development in FORTH seeks first to find the "simplest solution to a given problem." This is done by implementing selected parts of the problem separately and by ignoring as many constraints as possible. Then one or a few constraints are imposed and the program is modified. Comparing the modified program with the original provides an easy measure of the cost of those constraints or tradeoffs. For example, in a data base storage-and-retrieval application, if a particular user interface (i.e., command language) "costs" 10% more than the simplest solution, it may be worth the price. But if it costs 200% more, a wise manager may look for a less costly alternative. In traditional software development, the result is usually the prototype – the first and only system produced. No cost tradeoffs are available, so poor decisions are unknowingly but routinely made.

Considering human factors, the faster the repetition, the faster we learn. In a programming project, many small decisions must be made. If you could quickly determine how good each was, you could make better decisions and be more productive. The army discovered this applied to rifle training. If a soldier shot many rounds at a target before seeing the results, he would improve slowly. But if the result of each shot was visible immediately, he could achieve better accuracy than before and do it in less time. FORTH applies this principle by providing a very fast development cycle. The time to correct, compile, and test a module is typically less than a minute.

## Software Development Methodology

The software development cycle is similar to the Scientific Method (see Figure 3).

1. A problem is analyzed to determine *what* functions are required in the solution.

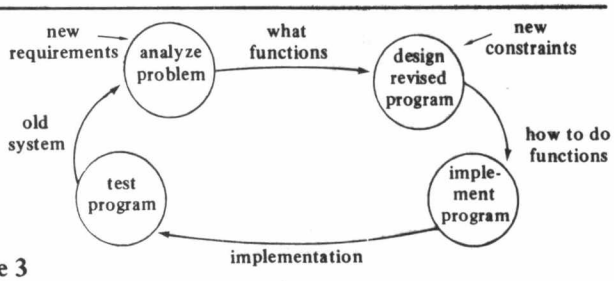



**Figure 3**
**The Software Development Cycle**

2. Decisions are made about *how* to achieve those functions with the available resources.
3. A program is written which attempts to implement the design.
4. The program is tested to determine if the functions were implemented correctly.

The analysis and design steps are applied in "top-down" order: the system is decomposed into inner modules starting with the problem as a whole. This is common to both FORTH and modern industry practice. However, FORTH departs from the norm for the implementation and testing steps. FORTH performs these in "bottom-up" order: the lowest level modules are written and tested first; the top level modules are last. This order has been demonstrated to be superior to the traditional top-down order. Top-down implementation and testing requires a lot of retesting which wastes time and lowers productivity. In order to test a top level module before lower modules are implemented, "stubs" must be supplied for them. Stubs are modules which do nothing when executed. A previously tested top level module must be retested when stubs are replaced by fully-implemented modules. With bottom-up order, no stubs are needed, only implemented modules are tested, and each module needs to be verified once.

Another departure FORTH makes in software development concerns the number of times the software development cycle is followed. Traditionally, the cycle is used only once, as shown in Figure 4a. FORTH applies these iteratively, as shown in Figure 4b.




**Figure 4**
**Stepping Through the Software Development Cycle**

The technique is to separate the required functions and constraints of a problem into several groups. Each of the groups will be processed by making another pass through the development cycle. FORTH typically applies the development in three major stages:

1. *Familiarization* – select a small set of requirements which defines a "skeleton problem;" ignore constraints. Consider alternative approaches; select the most promising. Solve this skeleton problem to determine the feasibility and difficulty of the original problem. An entire cycle should be used; the result will be an executable solution. Schedule and budget estimates can be made now.
2. *Expansion* – repeat the development cycle for each remaining group of functions and constraints until the whole problem is solved. Each group is added by modifying the previous implementation; after each cycle, the result is executable.
3. *Reduction* – reorganize the solution to achieve a higher quality result. Finally, add initiation and termination functions.

## System Structure

Another part of FORTH's philosophy guides the structure of software systems. It is a technique for organizing the components of a complicated model. A complex system should be *decomposed* into simpler systems. The components of a model or program are called "modules." Each module should consist of smaller, simpler modules (down to some limit). Thus, a system is a hierarchical collection of modules. This process of decomposition is common practice in the computing industry, but there are wide differences in the organization of the modules, the size of each module, the interfaces between modules, and the limit of modularization. FORTH has chosen a structure (see Figure 5) which has the following properties:

1. Each module should contain a single function.

**Figure 5a**
**Desired Hierarchy of Modules**

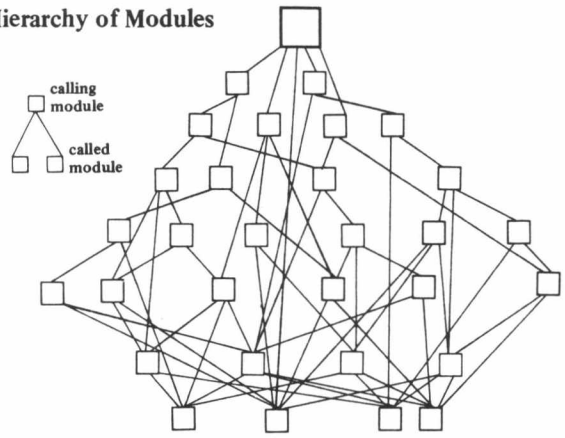



**Figure 5b**
**Ordered Complexity Layers**

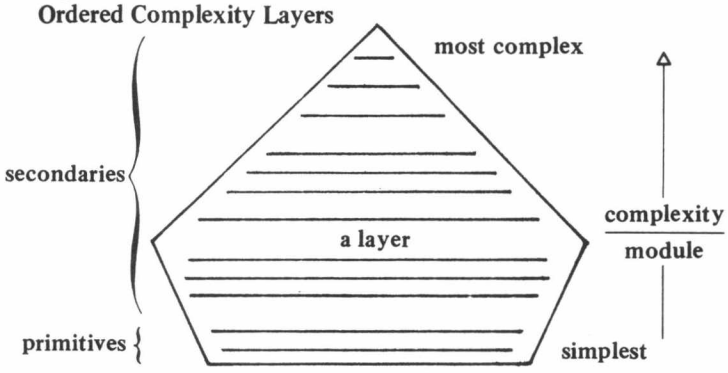



**Figure 5. Structure of a System**

2. Each module should have a simple interface to others.
3. The modules should be grouped into "layers" of equal complexity (see Figure 5b).
4. The layers should be ordered by complexity such that the bottom layer contains the simplest functions and the top would have the most complex.
5. Modules should be small, generally not referencing more than 9 others.

The fifth property limits the size of each module to approximately 9 components. This comes from considering some human factors. Psychologists have observed that all people have a limit to the number of concepts that can be understood at once. The "psychological juggling limit" is 7 ± 2 concepts (for most people).[3] This is the maximum number of things we can simultaneously analyze, trade-off, or optimize. If software modules obey this limit, they will be simpler and easier to understand. In FORTH's source langauge, this limit can be stated as, "Definitions should be 1 to 3 lines long."

This is in sharp contrast to modules in other languages, which are typically one to several pages long. FORTH occasionally has longer modules, but the longest is around 10 lines. A result of having small modules is that there must be a lot of them to build a large, complex system. A FORTH program typically will have as many module names as other languages would have labels.

One benefit of this structure is that every module can be defined in terms of lower-level modules and ultimately in terms of a few primitive operations. Hence FORTH's compiler supports this practice efficiently and directly. FORTH calls a module a "word." Every word has a name and a "definition." The lowest words are called "primitives;" all others are "secondaries." The collection of all executable words is called the "dictionary." A new word is added to the dictionary by compiling its definition. All secondary words' definitions contain references to numbers and previously defined words only. For this to be practical, the execution of words and the passing of data between them must be efficient and uniform.

A FORTH development system has six major layers (see Figure 6). These separate major functional capabilities. Each layer has many smaller, internal layers.

1. The innermost layer is the available hardware processor.
2. The *nucleus* layer implements FORTH's "virtual machine." This provides an environment for the execution of secondary FORTH words (e.g., the rest of the FORTH system and application programs). This environment is independent of the hardware processor. The nucleus is implemented in the instruction set of the processor. It is also possible to implement the nucleus in microcode or in hardware.
3. The *interactive* layer supports terminal-based activities between the system and users. This includes character string processing and a FORTH language interpreter (called the "text interpreter").
4. The *extensible* layer supports the addition, modification, or removal of application and system functions. This includes FORTH language compilers and assemblers.


**You Read**

**Dr. Dobb's Journal**

**And You Don't Subscribe?!**

**Save $13**
**off newsstand prices for 2 yrs.**
**Save $5 for 1 yr.**

Can you afford to miss an issue with information vital to your interests? As a subscriber you can look forward to articles on Small-C, FORTH, CP/M, S-100, Compiler optimization, Concurrent Programming and more, delivered right to your door. And you'll never miss the issue that covers your project.

**DR. DOBB'S JOURNAL**
**P.O. Box E**
**Menlo Park, CA 94025**

**Yes!** Sign me up for

____ 2 yrs. $47 ____ 1 yr. $25

____ I enclose a check/money order
____ Charge my Visa, MasterCard, American Express
____ Please bill me later

Name _____

Address _____

_____

_____ Zip _____

Credit Card _____ Exp. date _____

Account No. _____

Signature _____

9

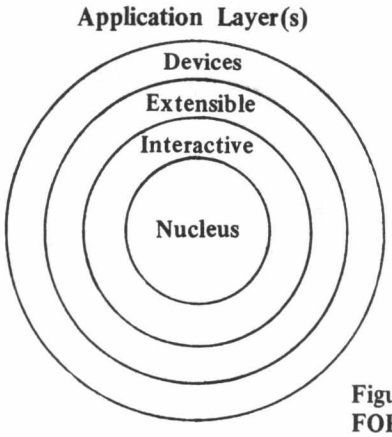



**Figure 6**
**FORTH's Layers**

5. The *devices* layer supports the use of peripheral devices. Serial devices (e.g., terminals and printers) and mass storage devices (e.g., disks and tapes) are supported. FORTH uses a "virtual memory management" scheme for mass memory transfers.
6. The *application* layer(s) contain user programs, utilities, tools, etc.

## Software Quality

Another important aspect of FORTH's philosophy is an emphasis on quality.[4] It is common for a FORTH programmer to set some quality standards and then evaluate all programs against them. The programs are revised or rewritten until the desired level of quality is reached.

For every problem to be solved, there is a set of requirements which any "acceptable" solution must satisfy. There are always many possible solutions which satisfy the minimum requirements and are therefore "acceptable." But all are not equal. One may be fastest, another smallest, and yet another most understandable. How do we make the best choice? We evaluate the quality of each solution and make decisions accordingly. The criteria used to judge the quality of a solution depends on the problem. The quality of a wine, a machined part, and a computer program have different measurement criteria. Computer software quality criteria is made of the following:

Correctness: complete, contained, consistent, and verifiable.
Performance: execution time, memory size, compile time, development time and cost, accuracy, and reliability.
Suitability: appropriate, usable, and understandable.
Modifiability: maintainable and changeable.

The overall quality of an entire application program (or one internal module) is determined by measuring each criterion, assigning to each a relative importance (i.e., weighting factor), and then combining them like a "weighted average." Although several criteria can be objectively measured, a quality assessment is always partly subjective. For example, an interrupt routine usually emphasizes minimum execution time but it sacrifices transportability, development time, and modifiability. On the other hand, a business data processing application strives for good usability, modifiability, and convenience, and it trades these for increased development time, execution time, and memory size. Therefore, to get a quality software product, we must produce an acceptable solution, judge its quality, then repeat this process until we produce a good product.

## Software Tools

Related to FORTH's emphasis on quality is its emphasis on software tools. A tool is a special technique which performs a desired operation on some material or information. In woodworking, there are kinds of cutting tools (e.g., circular table saw, jig saw, router, chisel). Each is designed for a particular combination of cutting requirements. Similarly, many special-purpose measuring tools exist (e.g., tape measure, vernier caliper). In software, a compiler is a tool which translates a source language into an object langauge. Tools can be used in both the production and judging activities of problem solving.

A tool is selected to improve the quality of the result. The more useful a tool is, the better. For one specific problem, the most useful tool will be one that closely matches the problem and the desired goal. So a tool should be special purpose, not general purpose. Tools in other disciplines are specialized (e.g., dental instruments, wrenches, cooking utensils). Specialization in software means that a tool either acts on selected parts of a system (not affecting the rest) or it acts on selected properties of the system. For example, data flow analysis and diagramming highlight the complexity of the interfaces between modules, but hides their internals. On the other hand, control flow analysis (i.e., flow charting) details the internal structure of a module, but it does not show interfaces.

It is likely that many tools well be developed for and used solely for one project. FORTH efficiently supports the making and using of tools during the development of a software system. Tools also share some common properties. They are built out of general purpose parts or utilities. For instance, there are thousands of different kinds of screwdrivers, but they are all built around the same general purpose "simple machines," namely the wedge and the lever. A tool must be "more powerful" than the stuff it is used on. A saw must be harder than the material it is trying to cut. For software tools, this means tools must be "higher level" than the software they are applied to. It also requires that tools have unrestricted access to and control of the environment surrounding the solution (e.g., computer hardware, operating system, language). The quality of a tool limits the quality of the result (see Figure 7). For example, a lathe must have been made to ten times better precision than that required of its products.

How do we decide when a tool should be used and what the tool should do? A good, experienced woodworker wisely chooses the right tool for each task. His experience was achieved after a lot of experiments during his entire career. He tries to solve a problem one way using a set of tools. Then he determines the quality of the result. If it does not meet his "standards," or if he predicts that he can do better, he tries another solution using different tools and techniques. So, if you fail to produce a good quality product, you take the time required to develop a tool which will allow you to produce a better product. It is important that individual programmers (as well as programming departments) establish quality standards and refine them as experience is gained. The standards should include both the quality requirements and the ways they will be measured.

## The Summary

FORTH is an unconventional software package. It is a unified approach to computer programming. It provides a complete programming environment: an operating system, an interpreter for interactive execution, a compiler, an assembler, an editor, and utilities. These are all written in the FORTH language; it is both a system and a meta system.

FORTH is a philosophy and an implementation of that philosophy. The philosophy provides the goals and techniques used in the implementation. The origins of the philosophy are the Scientific Method and human factors. FORTH uses modeling and specialized tools to understand complex systems. Decisions are based on the goals of simplicity and generality, and all results are improved using rapid repetition. Programming projects use several passes through the software development cycle (i.e., analysis, design, implementation, and testing). This technique allows problems to be solved incrementally, and the result of each pass is executable software. FORTH performs the analysis and design steps using top-down problem decomposition, but the implementation and testing steps are done in bottom-up order.

The decomposition separates a complex system into a hierarchy of simpler systems (i.e., modules). Each module should be small (containing a single function) and should have a simple interface to others. All modules of similar complexity are grouped into layers, and the layers are ordered so that the simplest is on the bottom and the most complex is on top. FORTH provides incentives to produce high quality results. Quality standards and measures should be assigned to each project, and they should guide the decisions made during development. High quality tools aid in the development of quality software. FORTH aids the development and use of specialized software tools.

## References

[1] Moore, Charles H., "The Evolution of FORTH, An Unusual Language," *Byte*, Volume 5, Number 8, August 1980.

[2] FORTH-79 Standard, produced by Forth Standards Team, distributed by Forth Interest Group.

[3] Miller, G. A., "Some Limits on Our Capacity for Processing Information," *Psychological Review*, Volume 63, 1956.

[4] Harris, Kim R., "Software Quality and Forth Programs," *Proceedings of the 6th West Coast Computer Faire*, April 1981. DDJ

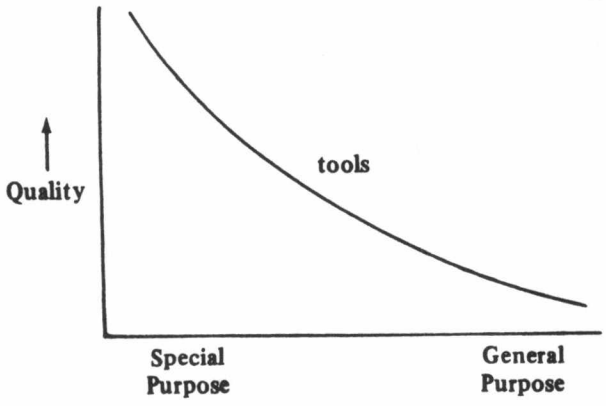



**Figure 7**
**The quality of a tool limits the quality of the result.**

**You Read**

# Dr. Dobb's Journal

**And You Don't Subscribe?!**

**Save $13**
**off newsstand prices for 2 yrs.**
**Save $5 for 1 yr.**

Can you afford to miss an issue with information vital to your interests? As a subscriber you can look forward to articles on Small-C, FORTH, CP/M, S-100, Compiler optimization, Concurrent Programming and more, delivered right to your door. And you'll never miss the issue that covers your project.

**DR. DOBB'S JOURNAL**
**P.O. Box E**
**Menlo Park, CA 94025**

**Yes!** Sign me up for
____ 2 yrs. $47 ____ 1 yr. $25

____ I enclose a check/money order
____ Charge my Visa, MasterCard, American Express
____ Please bill me later

Name _____

Address _____

_____

_____ Zip _____

Credit Card _____ Exp. date _____

Account No. _____

Signature _____

# A FORTH Assembler for the 6502

by William F. Ragsdale

*William F. Ragsdale, P. O. Box 3628, Hayward, CA 94540.*

## Introduction

This article should further polarize the attitudes of those outside the growing community of FORTH users. Some will be fascinated by a label-less macro-assembler whose source code is only 96 lines long! Others will be repelled by reverse Polish syntax and the absence of labels.

The author immodestly claims that this is the best FORTH assembler ever distributed. It is the only such assembler that detects all errors in op-code generation and conditional structuring. It is released to the public domain as a defense mechanism: three good 6502 assemblers were submitted to the FORTH Interest Group but each had some lack. Rather than merge and edit for publication, I chose to publish mine with all the submitted features plus several more.

Imagine having an assembler in 1300 bytes of object code with:

1. User macros (like IF, UNTIL) definable at any time.
2. Literal values expressed in any numeric base, alterable at any time.
3. Expressions using any resident computation capability.
4. Nested control structures without labels, with error control.
5. Assembler source itself in a portable high level language.

## Overview

FORTH is provided with a machine-language assembler to create execution procedures that would be time inefficient, if written as colon-definitions. It is intended that "code" be written similarly to high-level, for clarity of expression. Functions may be written first in high-level, tested, and then recoded into assembly, with a minimum of restructuring.

## The Assembly Process

Code assembly just consists of interpreting with the ASSEMBLER vocabulary as CONTEXT. Thus, each word in the input stream will be matched according to the FORTH practice of searching CONTEXT first then CURRENT.

| | |
|---|---|
| **ASSEMBLER** | (now CONTEXT) |
| **FORTH** | (chained to ASSEMBLER) |
| **user's vocab** | (CURRENT if one exits) |
| **FORTH** | (chained to user's vocab) |
| try for literal number | |
| else, do error abort | |

The above sequence is the usual action of FORTH's text interpreter, which remains in control during assembly.

During assembly of CODE definitions, FORTH continues interpretation of each word encountered in the input stream (not in the compile mode). These assembler words specify operands, address modes, and op-codes. At the conclusion of the CODE definition a final error check verifies correct completion by "unsmudging" the definitions name, to make it available for dictionary searches.

## Run-Time, Assembly-Time

One must be careful to understand at what time a particular word definition executes. During assembly, each assembler word interpreted executes. Its function at that instant is called "assembling" or "assembly-time." This function may involve op-code generation, address calculation, mode selection, etc.

The later execution of the generated code is called "run-time." This distinction is particularly important with the conditionals. At assembly time each such word (i.e., IF, UNTIL, BEGIN, etc.) itself "runs" to produce machine code which will later execute at what is labeled "run-time" when its named code definition is used.

## An Example

As a practical example, here's a simple call to the system monitor, via the NMI address vector (using the BRK opcode).

```
CODE MON  ( exit to monitor )
        BRK, NEXT JMP, END-CODE
```

The word CODE is first encountered, and executed by FORTH. CODE builds the following name "MON" into a dictionary header and calls ASSEMBLER as the CONTEXT vocabulary.

The "(" is next found in FORTH and executed to skip till ")". This method skips over comments. Note that the name after CODE and the ")" after "(" must be on the same text line.

## Op-codes

BRK, is next found in the assembler as the op-code. When BRK, executes, it assembles the byte value 00 into the dictionary as the op-code for "break to monitor via NMI."

Many assembler word names end in ",". The significance of this is:

1. The comma shows the conclusion of a logical grouping that would be one line of classical assembly source code.
2. "," compiles into the dictionary; thus a comma implies the point at which code is generated.
3. The "," distinguishes op-codes from possible hex numbers ADC and ADD.

## NEXT

FORTH executes your word definitions under control of the address interpreter, named NEXT. This short code routine moves execution from one definition to the next. At the end of your code definition, you must return control to NEXT or else to code which returns to NEXT.

## Return of Control

Most 6502 systems can resume execution after a break, since the monitor saves the CPU register contents. Therefore, we must return control to FORTH after a return from the monitor. NEXT is a constant that specifies the machine address of FORTH's address interpreter (say $0242). Here it is the operand for JMP,. As JMP, executes, it assembles a machine code jump to the address of NEXT from the assembly time stack value.

## Security

Numerous tests are made within the assembler for user errors:

1. All parameters used in CODE definition must be removed.
2. Conditionals must be properly nested and paired.
3. Address modes and operands must be allowable for the op-codes.

These tests are accomplished by checking the stack position (in CSP), as the "smudged" creation of the definition name remains in the "smudged" condition and will not be found during dictionary searches.

The user should be aware that one error not trapped is referencing a definition in the wrong vocabulary:

i.e., 0= of ASSEMBLER when you want
0= of FORTH

## Summary

The object code of our example is:

```
3059  83  4D  4F  CE    CODE MON
305D  4D  30              link field
305F  61  30              code field
3061  00                  BRK
3062  4C  42  02          JMP NEXT
```

## Op-codes, revisited

The bulk of the assembler consists of dictionary entries for each op-code. The 6502 one mode opcodes are:

BRK, CLC, CLD, CLI, CLV, DEX, DEY, INX, INY, NOP, PHA, PHP, PLA, PLP, RTI, RTS, SEC, SED, SEI, TAX, TAY, TSX, TXS, TXA, TYA,

When any of these are executed, the corresponding op-code byte is assembled into the dictionary.

The multi-mode op-codes are:

ADC, AND, CMP, EOR, LDA, ORA, SBC, STA, ASL, DEC, INC, LSR, ROL, ROR, STX, CPX, CPY, LDX, LDY, STY, JSR, JMP, BIT,

These usually take an operand, which must already be on the stack. An address mode may also be specified. If none is given, the op-code uses z-page or absolute addressing. The address modes are determined by:

| SYMBOL | MODE | OPERAND |
|---|---|---|
| .A | accumulator | none |
| # | immediate | 8 bits only |
| ,X | indexed X | z-page or absolute |
| ,Y | indexed Y | z-page or absolute |
| X) | indexed indirect X | z-page only |
| )Y | indirect indexed Y | z-page only |
| ) | indirect | absolute only |
| none | memory | z-page or absolute |

## Examples

Here are examples of FORTH vs. conventional assembler. Note that the operand comes first, followed by any mode modifier, and then the op-code mnemonic. This makes best use of the stack at assembly time. Also, each assembler word is set off by blanks, as is required for all FORTH source text.

```
     .A ROL,     ROL A           (.A distinguishes from
    1 # LDY,     LDY #1                hex number 0A)
 DATA ,X STA,    STA DATA,X
 DATA ,Y CMP,    CMP DATA,Y
    6 X) ADC,    ADC (06,X)
 POINT )Y STA,   STA (POINT),Y
VECTOR  ) JMP,   JMP (VECTOR)
```

The words DATA and VECTOR specify machine addresses. In the case of "6 X) ADC," the operand memory address $0006 was given directly. This is occasionally done if the usage of a value doesn't justify devoting the dictionary space to a symbolic value.

## 6502 Conventions

### Stack Addressing

The data stack is located in z-page, usually addressed by "Z-PAGE,X". The stack starts near $009E and grows downward. The X index register is the data stack pointer. Thus, incrementing X by two removes a data stack value; decrementing X twice makes room for one new data stack value.

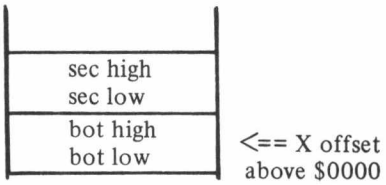
Sixteen-bit values are placed on the stack according to the 6502 convention; the low byte is at low memory, with the high byte following. This allows "indexed, indirect X" directly off a stack value.

The bottom and second stack values are referenced often enough that the support words BOT and SEC are included. Using

```
BOT LDA,  assembles  LDA (0,X)
SEC ADC,  assembles  ADC (2,X)
```

BOT leaves 0 on the stack and sets the address mode to ,X. SEC leaves 2 on the stack, also setting the address mode to ,X.

Here is a pictorial representation of the stack in z-page:




The 0 or 2 left by BOT or SEC is the base address above which the X register indexes. You may further modify this at assembly time to address at any byte in the data stack.

Here is an example of code to "or" to the accumulator four bytes on the stack:

```
BOT    LDA,   LDA (0,X)
BOT 1+ ORA,   ORA (1,X)
SEC    ORA,   ORA (2,X)
SEC 1+ ORA,   ORA (3,X)
```

To obtain the 14th byte on the stack: BOT 13 + LDA,

### Return Stack

The FORTH Return Stack is located in the 6502 machine stack in Page 1. It starts at $01FE and builds downward. No lower bound is set or checked as Page 1 has sufficient capacity for all (non-recursive) applications.

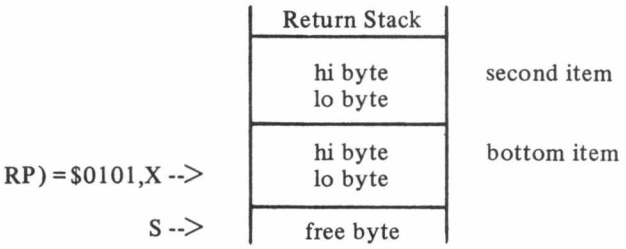
By 6502 convention the CPU's S register points to the next free byte below the bottom of the Return Stack. The byte order follows the convention of low significance byte at the lower address.

Return stack values may be obtained by: PLA, PLA, which will pull the low byte, then the high byte from the return stack. To operate on arbitrary bytes, the method is:

1. save X in XSAVE
2. execute TSX, to bring the S register to X.
3. use RP) to address the lowest byte of the return stack. Offset the value to address higher bytes. (Address mode is automatically set to ,X.)
4. restore X from XSAVE.

As an example, this definition non-destructively tests that the second item on the return stack (also the machine stack) is zero.

```
CODE IS-IT    ( zero ? )
    XSAVE STX,  TSX,    ( setup for return stack )
    RP) 2+ LDA,  RP) 3 + ORA,  ( or 2nd item's two bytes together)
  0= IF,  INY,  THEN,    ( if zero, bump Y to one )
    TYA,  PHA,  XSAVE LDX,  ( save low byte, restore data stack)
      PUSH JMP,  END-CODE (  push boolean )
```




## FORTH Registers

Several FORTH registers are available only at the assembly level and have been given names that return their memory addresses. These are:

- **IP** address of the Interpretive Pointer, specifying the next FORTH address which will be interpreted by NEXT.
- **W** address of the pointer to the code field of the dictionary definition just interpreted by NEXT. W-1 contains $6C, the op-code for indirect jump. Therefore, jumping to W-1 will indirectly jump via W to the machine code for the definition.
- **UP** User Pointer containing address of the base of the user area.
- **N** a utility area in z-page from N-1 through N+7.

## CPU Registers

When FORTH execution leaves NEXT to execute a CODE definition, the following conventions apply:

1. The Y index register is zero. It may be freely used.
2. The X index register defines the low byte of the bottom data stack item relative to machine address $0000.
3. The CPU stack pointer S points one byte below the low byte of the bottom return stack item. Executing PLA, will pull this byte to the accumulator.
4. The accumulator may be freely used.
5. The processor is in the binary mode and must be returned in that mode.

## XSAVE

XSAVE is a byte buffer in z-page, for temporary storage of the X register. Typical usage, with a call which will change X, is:

```
CODE DEMO
    XSAVE STX,   USER'S JSR,  ( which will change X )
    XSAVE LDX,   NEXT JMP,   END-CODE
```

## N Area

When absolute memory registers are required, use the "N Area" in the base page. These registers may be used as pointers for indexed/indirect addressing or for temporary values. As an example of use, see CMOVE in the system source code.

The assembler word N returns the base address (usually $00D1). The N area spans 9 bytes, from N-1 through N+7. Conventionally, N-1 holds one byte and N, N+2, N+4, N+6 are pairs which may hold 16-bit values. See SETUP for help on moving values to the N Area.

It is very important to note that many FORTH procedures use N. Thus N may only be used within a single code definition. Never expect that a value will remain there, outside a single definition!

```
CODE DEMO    HEX
    6 # LDA,  N 1 - STA,  ( setup a counter )
BEGIN,  8001 BIT,         ( tickle a port )
        N 1 - DEC,        ( decrement the counter )
     0< UNTIL,  NEXT JMP,  END-CODE   (  loop till negative )
```

## SETUP

Often we wish to move stack values to the N area. The sub-routine SETUP has been provided for this purpose. Upon entering SETUP the accumulator specifies the quantity of 16-bit stack values to be moved to the N area. That is, A may be 1, 2, 3, or 4 only:

```
3 # LDA,  SETUP JSR,
```

| stack before | | N after | | stack after |
|---|---|---|---|---|
| H high | | | | H |
| G_low | | | bot --> | G_ |
| F | | F | | |
| E_ | | E_ | | |
| D | | D | | |
| sec --> C_ | | C_ | | |
| B | | B | | |
| bot--> A_ | N--> | A_ | | |

## Control Flow

FORTH discards the usual convention of assembler labels. Instead, two replacements are used. First, each FORTH definition name is permanently included in the dictionary. This allows procedures to be located and executed by name at any time as well as be compiled within other definitions.

Secondly, within a code definition, execution flow is controlled by label-less branching according to "structured programming." This method is identical to the form used in colon-definitions. Branch calculations are done at assembly time by temporary stack values placed by the control words:

BEGIN, UNTIL, IF, ELSE, THEN,

Here again, the assembler words end with a comma, to indicate that code is being produced and to clearly differentiate from the high-level form.

One major difference occurs! High-level flow is controlled by run-time boolean values on the data stack. Assembly flow is instead controlled by processor status bits. The programmer must indicate which status bit to test, just before a conditional branching word (IF, and UNTIL,).

Examples are:

```
PORT LDA,   0= IF, <a>   THEN,
  (read port, if equal to zero do  <a> )

PORT LDA,   0= NOT IF, <a>   THEN,
  (read port,  if not equal to zero do  <a> )
```

The conditional specifiers for 6502 are:

| | | | | |
|---|---|---|---|---|
| CS | | test carry set | C=1 | in processor status |
| 0< | | byte less than zero | N=1 | |
| 0= | | equal to zero | Z=1 | |
| CS | NOT | test carry clear | C=0 | |
| 0< | NOT | test positive | N=0 | |
| 0= | NOT | test not equal zero | Z=0 | |

The overflow status bit is so rarely used, that it is not included. If it is desired, compile:

```
ASSEMBLER  DEFINITIONS   HEX
50  CONSTANT  VS      ( test overflow set )
```

## Conditional Looping

A conditional loop is formed at assembler level by placing the portion to be repeated between BEGIN, and UNTIL,:

```
        6 # LDA,    N STA,          ( define loop counter in N )
BEGIN,   PORT DEC,                  ( repeated action )
         N DEC,      0= UNTIL,      ( N reaches zero )
```

First, the byte at address N is loaded with the value 6. The beginning of the loop is marked (at assembly time) by BEGIN. Memory at PORT is decremented, then the loop counter in N is decremented. Of course, the CPU updates its status register as N is decremented. Finally, a test for Z=1 is made; if N hasn't reached zero, execution returns to BEGIN,. When N reaches zero (after executing PORT DEC, 6 times) execution continues ahead after UNTIL,. Note that BEGIN, generates no machine code, but is only an assembly time locator.

## Conditional Execution

Paths of execution may be chosen at assembly in a similar fashion as done in colon-definitions. In this case the branch is chosen based on a processor status condition code.

```
PORT LDA,   0= IF,   <for zero set>   THEN,  <continuing code>
```

In this example, the accumulator is loaded from PORT. The zero status is tested if set (Z=1). If so, the code <for zero set> is executed. Whether the zero status is set or not, execution will resume at THEN,.

The conditional branching also allows a specific action for the false case. Here we see the addition of the ELSE, part,

```
PORT LDA,  0= IF,   <for zero set>
                ELSE,  <for zero clear>
                THEN,   <continuing code>
```

The test of PORT will select one of two execution paths, before resuming execution after THEN,. The next example increments N based on bit D7 of a port:

```
PORT LDA,                ( fetch one byte )
 0< IF,   N DEC,         ( if D7=1, decrement N )
    ELSE,  N INC,        ( if D7=0, increment N )
    THEN,                ( continue ahead )
```

## Conditional Nesting

Conditionals may be nested, according to the conventions of structured programming. That is, each conditional sequence begun (IF, BEGIN,) must be terminated (THEN, UNTIL,) before the next earlier conditional is terminated. An ELSE, must pair with the immediately preceeding IF,.

```
BEGIN,    <code always executed>
   CS IF,  <code if carry set>
      ELSE,  <code if carry clear>   THEN,
   0= NOT UNTIL,  ( loop till condition flag is non-zero )
         <code that continues  onward>
```

Next is an error that the assembler security will reveal.

```
BEGIN,  PORT LDA,
   0= IF,  BOT INC,
         0= UNTIL,    THEN,
```

The UNTIL, will not complete the pending BEGIN, since the immediately preceding IF, is not completed. An error trap will occur at UNTIL, saying "conditionals not paired."

## Return of Control, revisited

When concluding a code definition, several common stack manipulations often are needed. These functions are already in the nucleus, so we may share their use just by knowing their return points. Each of these returns control to NEXT.

```
POP     POPTWO      Remove one or two 16-bit stack values.
PUSH                Add two bytes to the data stack.
PUT                 Write two bytes to the data stack, over the
                    present bottom of the stack.
```

Our next example complements a byte in memory. The byte's address is on the stack when INVERT is executed.

```
CODE INVERT  ( a memory byte )   HEX
   BOT X) LDA,     ( fetch byte addressed by stack )
    FF  # EOR,     ( complement accumulator )
    BOT X) STA,    ( replace in memory )
    POP JMP,   END-CODE ( discard pointer from stack, return to NEXT )
```

A new stack value may result from a code definition. We could program placing it on the stack by:

```
CODE ONE                  ( put 1 on the stack )
     DEX,  DEX,           ( make room on the data stack )
     1 # LDA,  BOT STA,   ( store low byte )
               BOT 1+ STY,  ( hi byte stored from Y since = zero )
               NEXT JMP,  END-CODE
```

*(Continued)*

A simpler version could use PUSH:

```
CODE ONE
   1 # LDA,  PHA,  ( push low byte to machine stack )
   TYA,  PUSH JMP,  ( high byte to accumulator, push to data stack )
   END-CODE
```

The convention for PUSH and PUT is:

1. push the low byte onto the machine stack.
2. leave the high byte in the accumulator.
3. jump to PUSH or PUT.

PUSH will place the two bytes as the new bottom of the data stack. PUT will over-write the present bottom of the stack with the two bytes. Failure to push exactly one byte on the machine stack will disrupt execution upon usage!

## Fooling Security

Occasionally we wish to generate unstructured code. To accomplish this we can control the assembly time security checks, to our purpose. First, we must note the parameters utilized by the control structures at assembly time. The notation below is taken from the assembler glossary. The --- indicates assembly time execution, and separates input stack values from the output stack values of the words' execution.

```
BEGIN,   ==>                          ---  addrB  1
UNTIL,   ==>     addrB   1    cc      ---

IF,      ==>                     cc   ---  addrI  2
ELSE,    ==>     addrI   2            ---  addrE  2
THEN,    ==>     addrI   2            ---
           or    addrE   2            ---
```

The address values indicate the machine location of the corresponding 'B'EGIN, 'I'F, or 'E'LSE,. cc represents the condition code to select the processor status bit referenced. The digit 1 or 2 is tested for conditional pairing.

The general method of security control is to drop off the check digit and manipulate the addresses at assembly time. The security against errors is less, but the programmer is usually paying intense attention to detail during this effort.

To generate the equivalent of the high level:

```
BEGIN  <a>  WHILE  <b>  REPEAT
```

we write in assembly:

```
BEGIN,  DROP  ( the check digit 1, leaving addrB)
          <a>
  CS IF,     ( leaves addrI and digit 2 )
          <b>
     ROT  ( bring addrB to bottom )  JMP, ( to addrB of BEGIN, )
    THEN,  ( complete false forward branch from IF, )
```

It is essential to write the assembly time stack on paper, and run through the assembly steps, to be sure that the check digits are dropped and re-inserted at the correct points and addresses are correctly available. **DDJ**

**NOTE: See page 18 for Assembler Glossary.**

**LISTING ON PAGE 23.**

## Bibliography


1. Ragsdale, William F., *fig-FORTH Installation Manual and Model*, $10.00 USA, $13.00 overseas. FORTH Interest Group, P. O. Box 1105, San Carlos, CA 94070.
2. Feierbach, Gary, "FORTH, The Language of Machine Independence," *Computer Design*, June 1981. An excellent article with an 8080 Assembler.
3. Ting, Dr. C. H., *Systems Guide to fig-FORTH*, Mountain View Press, P. O. Box 4656, Mountain View, CA 94040.



Why wait
to start
increasing your sales?
Call now
to reserve
advertising space
in the next
*Dr. Dobb's Journal*
(415) 323-3111

# A FORTH Assembler (Text on pages 12-16)

ASSEMBLER GLOSSARY

#

Specify 'immediate' addressing mode for the next op-code generated.

)Y

Specify 'indirect indexed Y' addressing mode for the next op-code generated.

,X

Specify 'indexed X' addressing mode for the next op-code generated.

,Y

Specify 'indexed Y' addressing mode for the next op-code generated.

.A

Specify accumulator addressing mode for the next op-code generated.

0<  --- cc (assembling)

Specify that the immediately following conditional will branch based on the processor status bit being negative (Z=1), i.e. less than zero. The flag cc is left at assembly time; there is no run-time effect on the stack.

0=  --- cc (assembling)

Specify that the immediately following conditional will branch based on the processor status bit being equal to zero (Z=1). The flag cc is left at assembly time; there is no run-time effect on the stack.

;CODE

Used to conclude a colon-definition in the form:

: <name> . . . ;CODE <assembly code> END-CODE

Stop compilation and terminate a new defining word <name>. Set the CONTEXT vocabulary to ASSEMBLER, assembling to machine code the following nmemonics. An existing defining word must exist in <name> prior to ;CODE.

When <name> later executes in the form:

<name> <namex>

the definition <namex> will be created with its execution proceedure given by the machine code following <name>. That is, when <namex> is executed, the address interpreter jumps to the code following ,CODE in <name>.

ASSEMBLER in FORTH

Make ASSEMBLER the CONTEXT vocabulary. It will be searched first when the input stream is interpreted.

BEGIN,  --- addr 1 (assembling)
        --- (run-time)

Occurs in a CODE definition in the form:

BEGIN, . . . cc UNTIL,

At run-time, BEGIN, marks the start of an assembly sequence

repeatedly executed. It serves as the return point for the corresponding UNTIL,. When reaching UNTIL, a branch to BEGIN, will occur if the processor status bit given by cc is false; otherwise execution continues ahead.

At assembly time, BEGIN, leaves the dictionary pointer address addr and the value 1 for later testing of conditionary pairing by UNTIL,.

BOT --- n (assembling)

Used during code assembly in the form:

```
BOT LDA,    or    BOT 1+ X) STA,
```

Addresses the bottom of the data stack (containing the low byte) by selecting the ,X mode and leaving n=0, at assembly time. This value of n may be modified to another byte offset into the data stack. Must be followed by a multi-mode op-code nmemonic.

CODE

A defining word used in the form:

```
CODE <name>  . . . .  END-CODE
```

to create a dictionary entry for <name> in the CURRENT vocabulary. <name>'s code field contains the address of its parameter field. When <name> is later executed, the machine code in this parameter field will execute. The CONTEXT vocabulary is made ASSEMBLER, to make available the op-code nmemonics.

CPU n --- (compiling assembler)

An assembler defining word used to create assembler nmemonics that have only one addressing mode:

```
EA CPU NOP,
```

CPU creates the word NOP, with its op-code EA as a parameter. When NOP, later executes, it assembles EA as a one byte op-code.

CS --- cc (assembling)

Specify that the immediately following conditional will branch based on the processor carry is set (C=1). The flag cc is left at assembly time; there is no run-time effect on the stack.

ELSE, --- (run-time)
addr1 2 --- addr2 2 (assembling)

Occurs within a code definition in the form:

```
cc IF, <true part> ELSE, <false part> THEN,
```

At run-time, if the condition code specified by cc is false, execution will skip to the machine code following ELSE,. At assembly time ELSE, assembles a forward jump to just after THEN, and resolves a pending forward branch from IF. The values 2 are used for error checking of conditional pairing.

END-CODE

An error check word marking the end of a CODE definition. Successful execution to and including END-CODE will unsmudge the most recent CURRENT vocabulary definition, making it available for execution. END-CODE also exits the ASSEMBLER making CONTEXT the same as CURRENT. This word previously was named C;

*(Continued on next page)*

IF, cc --- addr 2 (assembly-time )
--- addr 2 (assembly-time)

Occurs within a code definition in the form:
cc IF, <true part> ELSE, <false part> THEN,
At run time, IF, branches based on the condition code cc, (0< or 0= or CS). If the specifed processor status is true, execution continues ahead, otherwise branching occurs to just after ELSE, (or THEN, when ELSE, is not present). At ELSE, execution resumes at the corresponding THEN,.

When assembling, IF, creates an unresolved forward branch based on the condition code cc, and leaves addr and 2 for resolution of the branch by the corresponding ELSE, or THEN,. Conditionals may be nested.

INDEX --- addr (assembling)
An array used within the assembler, which holds bit patterns of allowable addressing modes.

IP --- addr (assembling)
Used in a code definition in the form:
IP STA, or IP )Y LDA,
A constant which leaves at assembly time the address of the pointer to the next Forth execution address in a colon-definition to be interpreted.

At run-time, NEXT moves IP ahead within a colon-definition. Therefore, IP points just after the execution address being interpreted. If an in-line data structure has been compiled (i.e. a character string), indexing ahead by IP can access this data:

IP STA, or IP )Y LDA,
loads the third byte ahead in the colon-definition being interpreted.

M/CPU n1 n2 --- (compiling assembler)
An assembler defining word used to create assembler nmemonics that have multiple address modes:
1C6E 60 M/CU ADC,
M/CPU creates the word ADC, with two parameters. When ADC, later executes, it uses these parameters, along with stack values and the contents of MODE to calculate and assemble the correct op-code and operand.

MEM Used within the assembler to set MODE to the default value for direct memory addressing, z-page.

MODE --- addr
A variable used within the assembler, which holds a flag indicating the addressing mode of the op-code being generated.

N --- addr (assembling)
Used in a code definition in the form:
N 1 - STA, or N 2+ )Y ADC,

A constant which leaves the address of a 9 byte workspace in z-page. Within a single code definition, free use may be made over the range N-1 thru N+7. See SETUP.

NEXT ---- addr (assembling)

A constant which leaves the machine address of the Forth address interpreter. All code definitions must return execution to NEXT, or code that returns to NEXT (i.e. PUSH, PUT, POP, POPTWO).

NOT cc1 --- cc2 (assembly-time)

When assembling, reverse the condition code for the following conditional. For example:

```
0= NOT IF, <true part> THEN,
```

will branch based on 'not equal to zero'.

POP --- addr (assembling)
n --- (run-time)

A constant which leaves (during assembly) the machine address of the return point which, at run-time, will pop a 16-bit value from the data stack and continue interpretation.

POPTWO --- addr (assembling)
n1 n2 --- (run-time)

A constant which leaves (during assembly) the machine address of the return point which, at run-time, will pop two 16-bit values from the data stack and continue interpretation.

PUSH --- addr (assembling)
--- n (run-time)

A constant which leaves (during assembly) the machine address of the return point which, at run-time, will add the accumulator (as high-byte) and the bottom machine stack byte (as low-byte) to the data stack.

PUT --- addr (assembling)
n1 --- n2 (run-time)

A constant which leaves (during assembly) the machine address of the return point which, at run-time, will write the accumulator (as high-byte) and the bottom machine stack byte(as low-byte) over the existing data stack 16-bit value (n1).

RP) --- (assembly-time)

Used in a code definition in the form:

```
RP) LDA,    or   RP) 3+ STA,
```

Address the bottom byte of the return stack (containing the low byte) by selecting the ,X mode and leaving n=$101. n may be modified to another byte offset. Before operating on the return stack the X register must be saved in XSAVE and TSX, be executed; before returning to NEXT, the X register must be restored.

SEC --- n (assembling)

Identical to BOT, execpt that n=2. Addresses the low byte of the second 16-bit data stack value (third byte on the data stack).

THEN, --- (run-time)

*(Continued on next page)*

```
        addr  2  ---   (assembly-time)
```
Occurs in a code definition in the form:
```
        cc  IF,  <true part>  ELSE,  <false part>  THEN,
```
At run-time THEN, marks the conclusion of a conditional structure. Execution of either the true part or false part resumes following THEN,. When assembling addr and 2 are used to resolve the pending forward branch to THEN,.

UNTIL,
```
                    ---   (run-time)
   addr  1  cc      ---   (assembling)
```
Occurs in a CODE definition in the form:
```
        BEGIN,  . . .  cc  UNTIL,
```
At run-time, UNTIL, controls the conditional branching back to BEGIN,. If the processor status bit specified by cc is false, execution returns to BEGIN,; otherwise execution continues ahead.

At assembly time, UNTIL, assembles a conditional relative branch to addr based on the condition code cc. The number 1 is used for error checking.

UP
```
        ---  addr    (assembling)
```
Used in a code definition in the form:
```
          UP LDA,    or  UP )Y STA,
```
A constant leaving at assembly time the address of the pointer to the base of the user area. i.e.
```
       HEX    12 # LDY,    UP )Y LDA,
```
load the low byte of the sixth user variable, DP.

UPMODE
```
     addr  f  ---  addr  f
```
Used within the assembler to adjust the addressing mode based on the operand size and opcode type.

W
```
        ---  addr    (assembling)
```
Used in a code definition in the form:
```
         W 1+ STA,    or    W 1 - JMP,    or   W )Y ADC,
```
A constant which leaves at assembly time the address of the pointer to the code field (execution address) of the Forth dictionary word being executed. Indexing relative to W can yield any byte in the definitions parameter field. i.e.
```
         2 # LDY,   W )Y LDA,
```
fetches the first byte of the parameter field.

X)

Specify 'indexed indirect X' addressing mode for the next op-code generated.

XSAVE
```
        ---  addr    (assembling)
```
Used in a code definition in the form:
```
        XSAVE STX,   or  XSAVE LDX,
```
A constant which leaves the address at assembly time of a temporary buffer for saving the X register. Since the X register indexes to the data stack in z-page, it must be saved and restored when used for other purposes.

# A FORTH Assembler (Listing)

```
FORTH Assembler for 6502 by W. F. Ragsdale        July 1, 1980

SCR # 81
  0 (  FORTH-65 ASSEMBLER                                WFR-79JUN03 )
  1 HEX
  2 VOCABULARY  ASSEMBLER  IMMEDIATE     ASSEMBLER  DEFINITIONS
  3
  4 ( REGISTER ASSIGNMENT SPECIFIC TO IMPLEMENTATION  )
  5 E0  CONSTANT  XSAVE        DC  CONSTANT  W       DE  CONSTANT  UP
  6 D9  CONSTANT  IP           D1  CONSTANT  N
  7
  8 (  NUCLEUS LOCATIONS ARE IMPLEMENTATION SPECIFIC )
  9 '  (DO)  0E  +    CONSTANT  POP
 10 '  (DO)  0C  +    CONSTANT  POPTWO
 11 '  LIT  13  +  CONSTANT  PUT
 12 '  LIT  11  +  CONSTANT  PUSH
 13 '  LIT  18  +  CONSTANT  NEXT
 14 '  EXECUTE  NFA  11  -  CONSTANT  SETUP
 15

SCR # 82
  0 (     ASSEMBLER, CONT.                               WFR-78OCT03 )
  1 0   VARIABLE  INDEX        -2  ALLOT
  2 0909 , 1505 , 0115 , 8011 , 8009 , 1D0D , 8019 , 8080 ,
  3 0080 , 1404 , 8014 , 8080 , 8080 , 1C0C , 801C , 2C80 ,
  4
  5 2  VARIABLE  MODE
  6 : .A   0 MODE ! ;     : #    1 MODE ! ;     : MEM  2 MODE ! ;
  7 : ,X   3 MODE ! ;     : ,Y   4 MODE ! ;     : X)   5 MODE ! ;
  8 : )Y   6 MODE ! ;     : )    F MODE ! ;
  9
 10 : BOT        ,X    0  ;      ( ADDRESS THE BOTTOM OF THE STACK *)
 11 : SEC        ,X    2  ;        ( ADDRESS SECOND ITEM ON STACK *)
 12 : RP)        ,X  101  ;       ( ADDRESS BOTTOM OF RETURN STACK *)
 13
 14
 15

SCR # 83
  0 (  UPMODE,  CPU                                      WFR-78OCT23 )
  1
  2 : UPMODE   IF  MODE @ 8 AND 0=  IF  8 MODE +!  THEN THEN
  3      1 MODE @  OF AND  -DUP  IF  0  DO  DUP + LOOP  THEN
  4      OVER 1+ @ AND  0=  ;
  5
  6 : CPU   <BUILDS  C,  DOES>  C@   C,  MEM  ;
  7      00 CPU BRK,   18 CPU CLC,   D8 CPU CLD,   58 CPU CLI,
  8      B8 CPU CLV,   CA CPU DEX,   88 CPU DEY,   E8 CPU INX,
  9      C8 CPU INY,   EA CPU NOP,   48 CPU PHA,   08 CPU PHP,
 10      68 CPU PLA,   28 CPU PLP,   40 CPU RTI,   60 CPU RTS,
 11      38 CPU SEC,   F8 CPU SED,   78 CPU SEI,   AA CPU TAX,
 12      A8 CPU TAY,   BA CPU TSX,   8A CPU TXA,   9A CPU TXS,
 13      98 CPU TYA,
 14
 15
```

*(Continued on next page)*

# A FORTH Assembler (Listing continued, text on pages 12-16)

```
SCR # 84
  0 (  M/CPU,    MULTI-MODE OP-CODES                        WFR-79MAR26 )
  1 : M/CPU   <BUILDS  C,  ,  DOES>
  2         DUP 1+ @ 80 AND   IF  10 MODE +! THEN   OVER
  3         FF00 AND UPMODE UPMODE    IF MEM CR LATEST ID.
  4         3  ERROR  THEN  C@  MODE  C@
  5         INDEX + C@ + C,    MODE C@ 7 AND   IF MODE C@
  6         0F AND 7 <   IF C, ELSE , THEN   THEN  MEM  ;
  7
  8     1C6E 60 M/CPU ADC,   1C6E 20 M/CPU AND,   1C6E C0 M/CPU CMP,
  9     1C6E 40 M/CPU EOR,   1C6E A0 M/CPU LDA,   1C6E 00 M/CPU ORA,
 10     1C6E E0 M/CPU SBC,   1C6C 80 M/CPU STA,   0D0D 01 M/CPU ASL,
 11     0C0C C1 M/CPU DEC,   0C0C E1 M/CPU INC,   0D0D 41 M/CPU LSR,
 12     0D0D 21 M/CPU ROL,   0D0D 61 M/CPU ROR,   0414 81 M/CPU STX,
 13     0486 E0 M/CPU CPX,   0486 C0 M/CPU CPY,   1496 A2 M/CPU LDX,
 14     0C8E A0 M/CPU LDY,   048C 80 M/CPU STY,   0480 14 M/CPU JSR,
 15     8480 40 M/CPU JMP,   0484 20 M/CPU BIT,

SCR # 85
  0 (  ASSEMBLER CONDITIONALS                              WFR-79MAR26 )
  1 : BEGIN,    HERE  1  ;                    IMMEDIATE
  2 : UNTIL,    ?EXEC >R 1 ?PAIRS R> C, HERE 1+ - C, ; IMMEDIATE
  3 : IF,       C,  HERE  0  C,  2  ;        IMMEDIATE
  4 : THEN,     ?EXEC  2  ?PAIRS  HERE  OVER  C@
  5           IF SWAP ! ELSE  OVER 1+ - SWAP C!  THEN  ; IMMEDIATE
  6 : ELSE,     2 ?PAIRS HERE 1+   1 JMP,
  7             SWAP HERE OVER 1+ - SWAP  C!  2  ;  IMMEDIATE
  8 : NOT    20  +  ;                          ( REVERSE ASSEMBLY TEST )
  9 90  CONSTANT  CS                  ( ASSEMBLE TEST FOR CARRY SET )
 10 D0  CONSTANT  0=                ( ASSEMBLER TEST FOR EQUAL ZERO )
 11 10  CONSTANT  0<              ( ASSEMBLE TEST FOR LESS THAN ZERO )
 12 90  CONSTANT  >=    ( ASSEMBLE TEST FOR GREATER OR EQUAL ZERO )
 13                    ( >= IS ONLY CORRECT AFTER SUB, OR CMP,  )
 14
 15

SCR # 86
  0 (  USE OF ASSEMBLER                                    WFR-79APR28 )
  1 : END-CODE                              ( END OF CODE DEFINITION *)
  2       CURRENT @ CONTEXT !    ?EXEC ?CSP SMUDGE ;  IMMEDIATE
  3
  4 FORTH  DEFINITIONS    DECIMAL
  5 : CODE                    ( CREATE WORD AT ASSEMBLY CODE LEVEL *)
  6       ?EXEC  CREATE  [COMPILE]  ASSEMBLER
  7       ASSEMBLER  MEM  !CSP  ;       IMMEDIATE
  8
  9 ( LOCK ASSEMBLER INTO SYSTEM )
 10 '  ASSEMBLER  CFA    '  ;CODE  8  +  !  ( OVER-WRITE SMUDGE )
 11 LATEST  12  +ORIGIN  !  ( TOP NFA )
 12 HERE    28  +ORIGIN  !  ( FENCE )
 13 HERE    30  +ORIGIN  !  ( DP )
 14 '  ASSEMBLER  6  +    32  +ORIGIN  !  ( VOC-LINK )
 15 HERE  FENCE  !
```

**End Listing**

# Sallying FORTH to Battle

by R. G. Loeliger

As most *Dr. Dobb's* readers are aware, there has been a rash of activity on the FORTH front of late. Minor skirmishes, thrusts and parries, and assorted small sorties have been appearing in print. Fortunately there has been no reported blood-letting. Not being one to avoid such nonsense, I too take sword in hand (well, at least a pen) and whiffle off to battle.

What is FORTH and why all the furor? FORTH is after all only a computer language – right? Well, maybe! The jury is still out on the language issue in most respects. FORTH is a threaded language certainly, but it is just as helpful to consider it an environment as a language. Hopefully this and several other attributes of FORTH will become more evident in the following.

The essential idea behind FORTH or any other threaded language is to create a pseudo-machine from a real machine (or computer). The heart of the psuedo-machine is called the inner interpreter. It controls the environment for "instruction" execution on a dual stack psuedo-machine. The instructions the psuedo-machine recognizes are primitives (or zero address machine operations) and secondaries. The primitives always manipulate data relative to the stacks and are real-machine specific. That is, the primitives (and the inner interpreter) must be coded in the machine instructions of the computer at hand. Except for the input/output driver routines, which are always machine specific, no other machine-specific code is necessary to implement a threaded language. We shall leave open the question of which primitives constitute a sufficient set to assure construction of the threaded language.

Secondaries are lists of "instruction" addresses where each instruction address is the address of a primitive or the address of a previously defined secondary. Only the primitive code sequences are ever actually executed, as well as inner interpreter code. A secondary as an "instruction" is simply equivalent to a subroutine call. The secondaries are internal nodes of a tree structure, the end nodes of which are primitives. The inner interpreter simply walks the tree looking for end nodes to execute. The dual stack is necessary so that one stack can be used for data storage and the other can be used for tree walk node address storage.

The primitives are somewhat analogous to the pure instructions of the psuedo-machine (the machine code of a real computer). They always cause a specific machine state change to occur. A secondary is more like a macro-instruction. When primitives and secondaries are strung together in a proper sequence, a program can result. The program is itself only a single psuedo-machine instruction since it is itself a secondary. Does this make FORTH a pseudo-machine or a language? Unfortunately the answer is not clear.

There has been a recent flurry of activity to standardize FORTH. The most obvious and well-founded example is FORTH-79 (Ref. 1). This standard does not address the issue of the pseudo-machine itself but attempts to control the result (the machine state change) for each keyword in the language. The FORTH-79 nucleus words are not defined as the allowable primitives for example. The problem is that there are not rules for the division of the language between primitives and secondaries. Any secondary can be written as a primitive. Not all primitives can be written as secondaries, but some can be. There are advantages in primitive definitions over secondaries in that a primitive never executes inner interpreter code.

There is not one syntax diagram in the FORTH-79 standard. The same can be said of the C programming language standard (Ref. 2) although the latter does make a non-rigorous (read vague) attempt to outline a syntax. So what's the difference? The FORTH-79 language standard is more like an Instruction Set Architecture (ISA) standard than a language standard; i.e., a recipe for building a computer rather than a language. It comes complete with machine state change diagrams.

FORTH does have a syntax though, in much the manner that assembly language has a syntax. The syntax is much more apparent than real. Nothing in the "language" really signifies the expected machine state change. It is common practice to choose keyword names that allow easy recall of the associated action (e.g., DROP drops a data value from the data stack). Because of the machine state change implied by every keyword, it can be extremely difficult to decode a given secondary based only on its formal definition without a definitive definition of initial input stack data parameters. If all this sounds vaguely like assembly language, you're getting close.

All of this is further complicated by the extensibility of FORTH. In FORTH it is possible to interactively extend the keyword definitions by adding (or as it is called, compiling) new secondaries or by coding new primitives. The pseudo-machine is thus not a static environment in the FORTH environment. This flexibility is an advantage and a potential hazard. Since the environment can be easily changed, almost any keyword or program can be emulated by making an associated pseudo-machine instruction (a keyword) equivalent to the "program" action. Conversely a slip of the hand or mind while constructing the instruction tree can send the pseudo-machine into never-never land. A system reload is then necessary to re-establish the pseudo-machine in its pristine state.

Is FORTH then a language? In a classic computer language sense, probably not a higher order language. A language? Only if one believes in Santa Claus and assembly language. Is it the end-all of microcomputer languages? We shall discuss this further.

Computer languages almost always arise for practical purposes. A problem is at hand that begs for a solution and that cannot be easily solved using an existing language. Consider the following:

| Language | Original Purpose |
|---|---|
| BASIC | Introductory teaching language |
| FORTRAN | Engineering and scientific problems |
| COBOL | Business problems |
| LISP | List processing |
| Pascal | Computer Science instruction |
| C | Systems programming |
| SAIL | Artificial intelligence research |
| XPL | Compiler writing |
| Ada | Embedded avionics (real-time) systems |
| ALGOL 60 | Algorithms |
| ALGOL 68 | Everything in the world |

Almost all of these languages can be and

*Mr. Loeliger is the author of the book,* Threaded Interpretive Languages. *R. G. Loeliger, c/o Logicon, Inc., Kittyhawk Office Center, 5717 Huberville Ave., Suite 300, Dayton, OH 45431.*

are used for purposes far beyond their original intent, i.e., bent to fill a far different goal than their designer envisioned. I have, as an obvious example, a Pascal compiler written in BASIC. The simple fact that some language is misapplied successfully is not proof that it is ideally suited for the application.

So why in the world did Charles Moore of FORTH, Inc. ever invent FORTH? For a practical reason, of course! He was faced with a difficult real-time control problem and a small computer. The compilers available for the machine churned out dismal code and were very large, unwieldy, inefficient and not very user-friendly. His solution (FORTH) is small, relatively efficient, very user-friendly and quite capable of real-time control. FORTH did not spring forth in full bloom overnight but evolved over a period of some years. It retains all of the good features of its original design goal and most of its original faults.

For a constrained environment such as its design point problem, FORTH is an ideal solution. In a typically constrained environment such as a personal computer, it is a reasonable solution. It is not the answer to all problems and was never meant to be.

One of the very nice features about FORTH or any other threaded language is that the system is very memory conservative. An 8K threaded language has many powerful capabilities and about 150 to 200 user-available commands plus assembler commands. New keywords can be easily compiled or assembled and tested easily. The typical claim that FORTH definitions take less space than assembly langauge definitions is, of course, nonsense. The FORTH definition is a delta to the core machine definitions and depends on the core definitions. The assembly language program is stand-alone. For all that, a very small FORTH machine can be extremely versatile and a good bit of fun.

FORTH is an interactive environment. The entire machine state is available to the programmer at all times. Compilation is direct and rapid. This makes the user interface very friendly. It also makes the pseudo-machine very vulnerable. In a multi-user environment, it is next to impossible to protect the system from the vagrancies of the users. There is just not a convenient, reliable way to limit the "system command" as with a more conventional environment. In a single-user case, this is not usually a problem. System re-loading is not all that time consuming. Crashing a multi-user system is dangerous to one's health rather than simply irritating.

Program testing in FORTH is extremely easy. Since test values can be easily placed on the stack, the keyword executed, and the results displayed, a separate testing environment is never needed. Although programming *per se* is more difficult in FORTH than in a higher order language, program development is faster because of the ease of program testing. Designing the testing program itself is difficult in any event. A badly designed test program will not be improved simply by using FORTH.

FORTH does have its problems. The most damning of these are in readability and ease of documentation. It is almost impossible to read a FORTH definition and *know* what it is doing without resorting to a pencil. The pencil is necessary to keep track of the data stack at each keyword completion within the definition of a secondary or to decode a primitive machine or assembly language definition. Even if the keyword is defined on a screen as source text, the comments required to fully explain the action usually exceed the code by an order of magnitude. Documenting FORTH code is worse than documenting assembly language code in my opinion. I often do the documentation off-line and have been known to lose the laundry ticket on which my latest and most clever routine was noted. I have also lost unbound documentation to my crayon-crazed three-year-old. Off-line documentation is not foolproof.

The documentation of code on a screen is not too neat either. The screen is a 1K block of mass storage which is mapped to a 64-character by 16-line monitor display. Since it is a one-for-one mapping, one can not take advantage of control codes (e.g., tabs, etc.) to compact the lines and increase the density of the source code. The screen, for all its simplicity, is not very efficient for documentation. As a matter of fact, it isn't very efficient even for uncommented source code. Usable is the best adjective to describe the screen concept.

Because of the difficulty of documenting FORTH, it is non-optional for use in a production environment. If more than two programmers are involved, or if the program has a long life cycle, I would not use FORTH. Modifying even a well-documented program two years after the original programmer quit is bad enough. It is not impossible to use FORTH, only dicey. The coding and testing will go quickly but so might any chance for later modification and expansion.

FORTH is in reality an entire operating system environment. In its standard form it is not compatible with an operating system (OS), it is the OS! This is not to imply that it can't be made to operate under an OS such as Digital Research's CP/M, only that FORTH's virtual memory system as defined is incommensurate with an OS.

The basic method of accessing mass storage data in FORTH is in 1K units known as blocks. A block or a screen are equivalent in storage capacity. When FORTH is used under an OS environment, there is a required translation from a block number to a file structure reference. In FORTH there is no direct multi-block file structure. This forces all block references to the file structure to be handled at the 1K unit level regardless of the OS capability. As an OS environment, FORTH is primitive. Hiding FORTH under even a modestly sophisticated OS is somewhat like mounting a Model T Ford body on a Porsche chassis . . . not unusable, but decidedly weird.

As a stand-alone operating system environment, FORTH is not weird, only primitive. One disadvantage of its simplicity is that there is no directory structure. As a result there are keywords in most FORTH dialects to list the first line of (successive) FORTH screens. Comments can be placed on the first screen line to identify the screen contents in place of a true directory. There are two disadvantages to this. For starters there is no way to limit the search to a named file or group of files. At 1K per file there can be a lot of file scanning before the correct screen is located. Next (and last), the need for the identification comment in the screen further limits the amount of data one can squeeze on a screen. *C'est la vie.*

There are other problems in FORTH but most of the remaining are simply irritating rather than limiting. Contrary to the bad press Reverse Polish Notation (RPM) often receives, I personally prefer RPN to algebraic notation. FORTH's use of RPN is not only natural but pleasing. From a pure mathematics point of view, FORTH is easy to decode. It is only when

*(Continued on page 48)*

# Screen-Oriented Editor in FORTH

## by Henry Laxen

*Henry Laxen, 1259 Cornell Ave., Berkeley, CA 94706.*

Much has been said about FORTH in the past, but unfortunately very few real life applications have ever been presented. The following Screen-Oriented Editor is an attempt to alleviate this deficiency. The code more or less stands by itself, and has been written with transportability and modifiability in mind. It is entirely in high level FORTH, however to increase performance, one optional code word has been included. There are extensive comments on both what is going on line-by-line, and descriptive paragraphs explaining the function of each word.

I would like to say a few words about my experience in implementing this editor. I had been using FORTH for about 12 months, and considered myself quite fluent in it. I had just completed writing a little editor for the Quasar/Panasonic Hand Held Computer, and thus had a pretty good idea of what was involved in writing one. I sketched out a design top down, and proceeded to implement bottom up. I started about 10:00 AM on a Sunday morning, and had a working version, very similar to what is presented in this article, only 14 hours later. It included all of the features currently present with the exception of the automatic line 0 updating and a few minor enhancements I have since added. I was truly amazed at how easy the task was. It was quite difficult to write words that had bugs in them due to the usually trivial nature of the definitions. I estimate that if I had written an equivalent editor in Pascal, it would have taken me at least 3 days; had I tried to write it in assembly language, it would have required a week or two.

This editor, as it stands, has been implemented on several different CPU's with absolutely no modification whatsoever. These include such diverse processors as the Zilog Z-80 and a DEC PDP-11. Furthermore, it has been modified to run with many different types of terminals, as well as memory-mapped video boards. It stands as a testimonial to the extreme transportability of FORTH.

I hope those of you who bother to implement this editor on your system will find it useful and productive. I estimate it has improved my efficiency in editing source screens by a factor of two or so over the line-oriented editor supplied with the FIG model. I hope your experiences with FORTH are as pleasant as mine, and may the FORTH be with you.

DDJ

LISTING ON PAGE 28.

Don't let your hot utilities or computing ideas sit on the shelf. *DDJ* is a great place for bright authors to get a start. Contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# Screen Oriented Editor (Text on page 27)

```
10
( ACKNOWLEDGEMENTS                                    26APR81HHL
)  10 LIST 11 LOAD ;S

Welcome to the world of Full Screen Editing in Forth.  There
is documentation on how to use this editor on screens 12-23.
Furthermore, there are some modification notes on screens
42-44.  I hope you will have no problems using this editor.
If you have any questions or comments, please contact me at
the address below.  By the way, this takes a long time to
load, so please be patient.  May the Forth be with you.

Henry Laxen                    This editor is in the public
1259 Cornell Ave.              domain, and may be distributed
Berkeley, California           further with the inclusion
94706                          of this notice.
(415) 525-8582
```

```
11
( LOAD SCREEN FOR SYSTEM GENERATION                    27APR81HHL
)

  24 LOAD   ( LOAD GENERAL PURPOSE STUFF )
  38 LOAD   ( LOAD THE EDITOR )

( NOTE!!, THIS EDITOR ASSUMES THAT BLOCKS ARE 1024 BYTES
IN LENGTH, AND THAT THE FIG FORTH ENCLOSE BUG HAS BEEN
FIXED.  IF THIS IS NOT THE CASE, IT WILL REQUIRE MODIFICATION
IN SEVERAL PLACES, MOST NOTABLY IN:
   E-INIT
   E-UPDATE
   E-EXIT, E-SCRATCH
)
```

```
12
FORTH Development System Documentation                18SEP80DRB

   The FORTH editor is easy to learn and to use. It consists of
control key commands that allow cursor movement and text entry
and deletion so that editing can be done quickly and smoothly.

CURSOR MOVEMENT

   You can place the cursor anywhere on the screen by using a
few editing commands.

   The following table describes the commands that are relevant
to cursor movement.
```

```
13
CURSOR MOVEMENT COMMANDS                                18SEP80DRB

Command        Function        Description

  ^E           cursor up       Moves cursor up to the same
               a line          position in the preceding line.

  ^X           cursor down     Moves cursor down to the same
               a line          position in the next line.

*^R            cursor up       Moves cursor to the same
               four lines      position four lines up.

*^C            cursor down     Moves cursor to the same
               four lines      position four lines down.
```

```
14
CURSOR MOVEMENT COMMANDS                              18SEP80DRB

  ^S           cursor left     Moves cursor to the
  ^H           one character   previous character or
  delete                       space.

  ^D           cursor right    Moves cursor to the next
               one character   character or space.

  ^F           cursor right    Moves cursor forward to the
               a word          first letter of the next word
                               going to the beginning of the
                               next line if at the end of the
                               current line.
```

```
15
CURSOR MOVEMENT COMMANDS                                18SEP80DRB
  ^A           cursor left     Moves cursor back to the first
               a word          letter of the previous word
                               going to the end of the previous
                               line if at the beginning of the
                               current line.

  ^I           tab             Moves the cursor to the next tab
  TAB                          stop on the line going to the
                               next line if at the end of the
                               current line.
* The position of the cursor will differ depending on the number
of intervening lines between the top and the bottom of the
screen. If you enter ^R when the cursor is less than four lines
down from the top of the screen, the cursor moves to the first
space on the first line.
```

```
    16
 0 CURSOR HINTS                                    18SEP80DRB
 1
 2 Similarly, if the cursor is less than four lines up from the
 3 bottom of the screen, ^C moves it to the last space on the
 4 bottom line.
 5
 6    Using these commands singly or in combinations, you can move
 7 the cursor wherever you want it. If, for instance, you want to
 8 move the cursor to the end of the current line, you enter the
 9 following combination:
10
11                        <cr> carriage return
12                         ^A
13
14    When the cursor is anywhere in the last line, you can move
15 it to the beginning of that line by entering a carriage return.
```

```
    17
 0 TEXT ENTRY AND DELETION                         18SEP80DRB
 1
 2    With special editing commands, you can enter additional text
 3 or take out existing text in your FORTH screens. The following
 4 tables describe the commands that are relevant to text entry
 5 and deletion.
 6
 7                              TEXT ENTRY
 8
 9 Command        Function       Description
10
11  *^V          insertion      Types over characters when OFF
12               off/on         and inserts character when ON.
13                              When on, INSERT ON appears in
14                              the top (status) line. ^V is a
15                              toggle that turns insert off
```

```
    18
 0 SPEADING LINES                                  18SEP80DRB
 1                              when it is on and on when it is
 2                              off.
 3
 4   ^N          insert         Inserts a carriage return at the
 5               carriage       cursor postion, and leaves
 6               return         cursor unmoved. All text to the
 7                              right of and below cursor moves
 8                              down one line, but only if the
 9                              last line is blank. Text in the
10                              last line disables this command.
11
12 * A carriage return has the same effect on the text whether ^V
13 (insert mode) is on or off. In other words, pressing carriage
14 return while insert is on will not result in inserting a
15 carriage return.
```

```
    19
 0 TEXT DELETION                                   18SEP80DRB
 1
 2 Command        Function       Description
 3
 4   ^G          delete         Deletes the character under
 5               character      the cursor and moves everything
 6               right          to the right of it left one
 7                              character.
 8
 9   ^T          delete         Deletes word containing the
10               word right     cursor, specifically the
11                              character under the cursor and
12                              the characters to the right of
13                              it to the end of the line.
14
15
```

```
    20
 0 DELETING TEXT                                   25APR81HHL
 1
 2   ^B          delete line    Deletes the text in the current
 3               contents       line without deleting the line
 4                              (leaves a blank line).
 5
 6   ^Y          delete line    Deletes the current line and
 7                              moves the lines under it up
 8                              thereby shrinking the screen.
 9
10   ^K          delete         Deletes contents of the entire
11               (kill)         screen.
12               screen
13
14
15
```

```
    21
 0 EXITING FROM THE EDITOR                         18SEP80DRB
 1
 2 Command        Function       Description
 3
 4   ^Z          abandon        Exits from the editor. The
 5               screen         block is not marked as updated.
 6                              Hence, typing EMPTY-BUFFERS at
 7                              this point will allow you to
 8                              retrieve the contents of the
 9                              screen as it was before any
10                              editing took place. If you
11                              re-enter the editor before
12                              typing EMPTY-BUFFERS, you will
13                              get the edited version of the
14                              screen.
15
```

*(Continued on next page)*

# Screen Oriented Editor (Listing continued, text on page 27)

```
    22
 0 EXITING FROM THE EDITOR                          18SEP80DRB
 1
 2  ESC          exit          Exits the editor and writes the
 3               editor        screen back to disk if it was
 4                             modified. The used ID is auto-
 5                             matically inserted in the right-
 6                             most 10 characters of line 0.
 7
 8
 9
10
11
12
13
14
15
```

```
    23
 0 CURSOR POSITIONING WORDS                         18SEP80DRB
 1
 2 At the end of the editor you will find the cursor positioning
 3 routines required for some currently available terminals.
 4 Load the screen that corresponds to your terminal.  If it isn't
 5 there, you will have to write your own routines.  This is easy
 6 and you can use the ones provided as a model.
 7
 8 Good luck, and may the FORTH be with you.
 9
10
11
12
13
14
15
```

```
    24
 0 ( \            COMMENT TO END OF LINE            25APR81HHL
 1 )
 2
 3 : \     ( --- )
 4   IN @ C/L / 1+ C/L * IN ! ;
 5   IMMEDIATE
 6
 7 (  THIS IS THE COMMENT CHARACTER WHICH SHOULD ONLY BE USED
 8 INSIDE OF SCREENS.  IT WILL CAUSE ALL SUBSEQUENT CHARACTERS
 9 ON ITS LINE TO BE IGNORED.  INTERPRETATION WILL BEGIN ON THE
10 FOLLOWING LINE.  NOTE THAT \ MUST BE FOLLOWED IMMEDIATELY BY
11 A SPACE.  )
12
13 -->
14
15
```

```
    25
 0 \ (S   (P       DOCUMENTATION WORDS                26APR81HHL
 1
 2
 3 : (S      ( --- )
 4   41 WORD ;             \ SKIP TO )
 5   IMMEDIATE
 6
 7 ( (S IS USED FOR STACK COMMENTS.  A UTILITY MAY BE WRITTEN
 8 LATER TO EXTRACT THESE )
 9
10 : (P      ( --- )
11   41 WORD ;             \ SKIP TO )
12   IMMEDIATE
13
14 ( (P IS USED FOR DESCRIPTIVE COMMENTS.  A UTILITY MAY BE
15 WRITTEN LATER TO EXTRACT THESE )
   -->
```

```
    26
 0 \ DEPTH        RETURN DEPTH OF STACK              25APR81HHL
 1
 2 : DEPTH    (S --- N )
 3   SP@ S0 @ SWAP - 2 / 0 MAX ;
 4
 5 (P   RETURNS THE CURRENT DEPTH OF THE STACK.  THIS WORD MAY NEED
 6 TO BE REDEFINED WHEN RUNNING ON OTHER SYSTEMS, AS IT USES A
 7 COMPUTER DEPENDENT WORD CALLED S0, WHICH CONTAINS THE STARTING
 8 ADDRESS OF THE STACK. )
 9
10 -->
11
12
13
14
15
```

```
    27
 0 \ L            INTELLIGENT SCREEN LISTS           25APR81HHL
 1
 2 : L     (S [N] --- )
 3   DEPTH IF              \ IS THERE ANYTHING THERE
 4    DUP SCR !            \ YES, REMEMBER IT
 5   ELSE
 6    SCR @                \ NO, RETRIEVE LAST ONE
 7   ENDIF
 8   LIST ;                \ AND LIST IT
 9
10 (P   L WILL LIST THE SPECIFIED SCREEN AND REMEMBER
11 WHAT IS WAS.  IF THE DEPTH OF THE STACK IS ZERO, IT WILL
12 LIST THE LAST SCREEN PREVIOUS LISTED.  IT MAKES USE OF
13 THE USER VARIABLE SCR.  )
14
15 -->
```

```
    28
 0 \ 2DROP 2* BEEP                                    25APR81HHL
 1
 2 : 2DROP      (S N1 N2 --- )
 3    DROP DROP ;
 4 (P  2DROP DROPS 2 ITEMS OFF OF THE PARAMETER STACK. )
 5
 6 : 2*         (S N --- 2*N )
 7    DUP + ;
 8 (P 2* SIMPLY DOUBLES THE ITEM ON THE TOP OF THE STACK )
 9
10 : BEEP       (S --- )
11    7 EMIT ;
12 (P RING THE BELL ON THE TERMINAL.  USUALLY AFTER AN ERROR )
13
14 -->
15
```

```
    29
 0 \ BOUNDS       DO LOOP SETUP                       25APR81HHL
 1
 2 : BOUNDS    (S ADDR LEN --- ADDR+LEN ADDR )
 3    OVER + SWAP ;
 4
 5 (P BOUNDS IS A COMMON DO LOOP SETUP WORD.  IT ASSUMES THERE IS
 6 AN ADDRESS AND A LENGTH ON THE STACK.  BOUNDS CONVERTS THIS
 7 INTO A HIGH ADDRESS AND A LOW ADDRESS.  THE I INDEX OF A DO
 8 LOOP WILL THEN RUN THROUGH THIS RANGE OF VALUES WHILE EXECUTING
 9 )
10
11 -->
12
13
14
15
```

```
    30
 0 \ BMOVE        8080 CODE VERSION OF BMOVE          27APR81HHL
 1    -->         \ REMOVE THIS ARROW TO USE THIS WORD
 2 CODE BMOVE    ( FROM TO LEN --- )
 3    C L MOV
 4    B H MOV       \ (HL) <- (IP)
 5      B POP       \ (BC) <- LEN
 6      D POP       \ (DE) <- TO
 7       XTHL       \ (HL) <- FROM
 8      B DAD       \ (HL) <- FROM+LEN
 9      H DCX       \ (HL) <- FROM'
10       XCHG       \ (HL) <- TO, (DE) <- FROM'
11      B DAD       \ (HL) <- TO+LEN
12      H DCX       \ (HL) <- TO'
13       XCHG       \ (HL) <- FROM', (DE) <- TO', (BC) <- LEN
14     -->
15 \ THE ABOVE SETS UP FOR THE LOOP THAT IS TO FOLLOW
```

```
    31
 0 \ BMOVE        CONTD.                              27APR81HHL
 1    -->         \ REMOVE THIS --> TO USE THIS WORD
 2    BEGIN
 3      B A MOV       \ LEN IS IN (BC)
 4        C ORA       \ SEE IF 16 BIT 0
 5      WHILE JZ      \ LOOP WHILE NOT 0
 6        M A MOV     \ FETCH BYTE
 7          H DCX     \ FROM = FROM - 1
 8          D STAX    \ STORE BYTE
 9          D DCX     \ TO = TO - 1
10          B DCX     \ DECREMENT COUNT
11    REPEAT JMP      \ DO IT AGAIN
12    B POP           \ RESTORE IP
13    NEXT JMP        \ AND RETURN
14    C;
15 -->
```

```
    32
 0 \ BMOVE        HIGH LEVEL                          27APR81HHL
 1 \ -->          \ REMOVE THE FIRST \ IF THE CODE VERSION IS USED
 2
 3 : BMOVE     (S FROM TO LEN --- )
 4    -DUP IF                    \ ANYTHING TO MOVE?
 5     >R R + 1 - SWAP 1 -       \ YES, MOVE FROM HIGH
 6     R> BOUNDS SWAP DO         \ MEMORY TO LOW MEMORY
 7      I C@ OVER C!             \  *
 8      1 - -1                   \  *
 9     +LOOP DROP
10    ELSE
11     2DROP                     \ NOTHING TO DO, SO FORGET IT
12    ENDIF ;
13 (P BMOVE IS IDENTICAL TO CMOVE EXCEPT IT MOVES
14 CHARACTERS IN THE OTHER DIRECTION. )
15 -->
```

```
    33
 0 \ MOVE         WORKS IN EITHER DIRECTION           25APR81HHL
 1
 2 : MOVE   (S FROM TO LEN --- )
 3    ROT ROT 2DUP U< IF
 4      ROT BMOVE
 5    ELSE
 6      ROT CMOVE
 7    ENDIF ;
 8
 9 (P MOVE WILL MOVE LEN BYTES FROM ADDRESS FROM TO ADDRESS TO
10 AND WILL NOT OVERLAP THEM, NO MATTER WHAT THE RELATIVE VALUES
11 OF FROM TO AND LEN ARE.  MOVE SHOULD ALWAYS BE USED WHENEVER
12 THERE IS DANGER OF OVERLAPPING FIELDS )
13
14 -->
15
```

*(Continued on next page)*

```
    34                                                    35
 0 \ 1- 2-         DECREMENT                  25APR81HHL \ >= <> <= U>   CONDITIONALS                  25APR81HHL
 1
 2 : 1-    (S N --- N-1 )                                 : >=    (S N1 N2 --- BOOL )
 3   1 - ;                                                  < 0= ;
 4
 5 : 2-    (S N --- N-2 )                                 : <>    (S N1 N2 --- BOOL )
 6   2 - ;                                                  = 0= ;
 7
 8 -->                                                    : <=    (S N1 N2 --- BOOL )
 9                                                          > 0= ;
10
11                                                       : U>   (S N1 N2 --- BOOL )
12                                                         SWAP U< ;
13
14                                                       -->
15

    36                                                    37
 0 \ RE-FORTH      RE-ENTER FORTH FOR 1 LINE  25APR81HHL \ ?DUP         SAME AS -DUP FOR FORTH-79      21APR81HHL
 1
 2 : RE-FORTH    (S --- ??? )                           : ?DUP   (S N --- [N] N )
 3   IN @ >R                  \ SAVE INPUT BUFFER POINTER   -DUP ;
 4   BLK @ >R                 \ SAVE BLOCK NUMBER
 5   0 IN ! 0 BLK !           \ RESET FOR TERMINAL INPUT   (P DUPLICATES THE TOP OF THE STACK IF THE TOP IS
 6   QUERY INTERPRET          \ GET 1 LINE FROM TERMINAL   NONZERO.  IF THE TOP IS ZERO, THEN IT IS NOT DUPLICATED.
 7   R> BLK !                 \ RESTORE BLOCK NUMBER       THIS IS DEFINED PRIMARILY FOR FORTH-79 COMPATIBILITY )
 8   R> IN ! ;                \ RESTORE INPUT BUFFER POINTER
 9
10 (P RE-FORTH REENTERS THE FORTH INTERPRETER FROM THE TERMINAL
11 AND ALLOWS THE USER TO ENTER 1 LINE OF VALID FORTH COMMANDS.
12 THIS IS A SIMPLE WAY TO PROMPT FOR TERMINAL MESSAGES WHILE
13 IN THE MIDDLE OF LOADING. )
14
15 -->

    38                                                    39
 0 \ CASE:                                  25APR81HHL \ -TIDY                                      21APR81HHL
 1
 2 : CASE:    (S N --- )                                 : -TIDY   (S ADDR LEN --- )
 3   <BUILDS   ] SMUDGE                                   BOUNDS DO                  \ RUN THROUGH THE STRING
 4   DOES>                                                I C@ BL < IF               \ IS IT A CONTROL CHAR?
 5    SWAP 2* + @                                          BL I C!                   \ YES, REPLACE IT WITH A BLANK
 6    EXECUTE ;                                           ENDIF
 7                                                        LOOP ;
 8 (P CASE: IS A POOR MAN'S CASE STATEMENT.  AT COMPILE TIME, IT
 9 SIMPLY COMPILES CODE FIELD ADDRESS AS DOES : AT RUN TIME, IT   (P -TIDY REPLACES ALL CONTROL CHARACTERS IN A SPECIFIED
10 EXPECTS AN INDEX ON THE STACK, AND SIMPLY INDEXES INTO THE    RANGE WITH BLANKS )
11 DEFINED WORDS AND EXECUTES IT. )
12                                                       -->
13 -->
14
15
```

```
    40
 0 \ VARIABLE AND CONSTANT DEFINITIONS                  21APR81HHL
 1    VOCABULARY EDITOR IMMEDIATE EDITOR DEFINITIONS
 2  0 VARIABLE &MODE          \ CURRENT MODE ( OVERSTRIKE OR INSERT
 3  0 VARIABLE &CURSOR        \ CURSOR POSITION
 4  0 VARIABLE &OLD-MODE      \ PREVIOUS MODE
 5  0 VARIABLE &UPDATE        \ UPDATE FLAG
 6  0 VARIABLE &BUF-ADR       \ ADDRESS OF CURRENT BUFFER
 7  0 VARIABLE &E-ID          \ DATE AND USER ID LAST MODIFIED
 8     12 ALLOT
 9    &E-ID 14 BLANKS         \ INITIALIZE TO BLANKS
10
11  5 CONSTANT %X-OFF         \ X OFFSET FOR CURSOR POSITIONING
12  2 CONSTANT %Y-OFF         \ Y OFFSET FOR CURSOR POSITIONING
13 B/SCR B/BUF *    CONSTANT C/SCR    \ CHARACTER PER SCREEN
14 C/SCR C/L   /    CONSTANT L/SCR    \ LINES PER SCREEN
15 -->
```

```
    41
 0 \ CURSOR POSITIONING VECTORS                         25APR81HHL
 1
 2 0 VARIABLE 'CRTXY          \ CFA OF ROUTINE THAT MOVES CURSOR
 3 0 VARIABLE 'CRTCLR-SCR     \ CFA OF ROUTINE THAT CLEARS SCREEN
 4 0 VARIABLE 'CLEAR-TO-EOL   \ CFA OF ROUTINE THAT CLEARS TO EOL
 5
 6 : CRTXY        (S X Y --- )
 7    'CRTXY @ EXECUTE ;
 8
 9 : CRTCLR-SCR   (S --- )
10    'CRTCLR-SCR @ EXECUTE ;
11
12 : CLEAR-TO-EOL   (S --- )
13    'CLEAR-TO-EOL @ EXECUTE ;
14 -->
15
```

```
    42
 0 \ DESCRIPTION OF CURSOR COMMANDS                     25APR81HHL
 1 -->
 2 The CRTXY command must position the cursor at the X and Y
 3 co-ordinates given on the stack.  If your terminal does not
 4 support cursor positioning, give up.
 5
 6 The CRTCLR-SCR should clear the entire screen and home the
 7 cursor.  If such a command is not available, output a sufficient
 8 number of line-feeds to clear the screen via scrolling, and then
 9 call CRTXY with 0 0.
10
11 The CRTCLR-EOL command should clear the line from its current
12 location to the end of the line.  The following will always
13 work, but if your terminal supports something more sophisticated
14 you should use it.
15          C/L MOD C/L SWAP - SPACES
```

```
    43
 0 \ DESCRIPTION OF CURSOR POSITIONING COMMANDS         25APR81HHL
 1
 2 -->
 3
 4 NOTE!!
 5 The CLEAR-TO-EOL command is ALWAYS called with the cursor
 6 positioned at the correct place on the screen.  Hence if
 7 your terminal supports a clear to end of line command, the
 8 position passed can be simply dropped and the terminal
 9 command issued.  If the terminal does not support such a
10 command, you must blank out the remainder of the line based
11 on the cursor position that was passed.
12
13
14
15
```

```
    44
 0 \ DESCRIPTION OF CURSOR POSITIONING COMMANDS         26APR81HHL
 1
 2 -->
 3
 4 If you would like to see how some sample cursor positioning
 5 routines were written, take a look at screens 90-91.  Note
 6 that you may write your own routines even after the complete
 7 editor has been loaded.  All you need do is patch the execute
 8 vectors for your particular terminal.  Good luck.
 9
10
11
12
13
14
15
```

```
    45
 0 \ CURPOS +CURPOS   MOVE-CURSOR                       25APR81HHL
 1 : CURPOS   (S --- POS )
 2    &CURSOR @ ;             \ RETURN CURRENT CURSOR POSITION
 3
 4 : +CURPOS   (S N --- )
 5    &CURSOR +!
 6    CURPOS 0 MAX            \ AND DO BOUNDS CHECKING
 7    [ C/SCR 1- ] LITERAL    \ CHAR PER SCREEN - 1
 8    MIN &CURSOR ! ;         \ ALWAY VALID
 9
10 : MOVE-CURSOR   (S N --- )
11    +CURPOS                 \ MOVE THE CURSOR
12    CURPOS C/L /MOD         \ RAW X Y
13    %Y-OFF + SWAP           \ ADD IN Y OFFSET
14    %X-OFF + SWAP           \ ADD IN X OFFSET
15    CRTXY ;         -->     \ AND MOVE THERE
```

*(Continued on next page)*

# Screen Oriented Editor (Listing continued, text on page 27)

```
     46
 0 \ BUF-ADR   BUFPOS                                  25APR81HHL
 1
 2 : BUF-ADR   (S POS --- ADDR )
 3    &BUF-ADR @ + ;
 4
 5 (P BUF-ADR CONVERTS THE CURSOR POSITION IT IS CALLED WITH
 6 TO THE ADDRESS WITHIN THE DISK BUFFER WHICH CORRESPONDS
 7 TO THAT POSITION )
 8
 9 : BUFPOS   (S --- ADDR )
10    CURPOS BUF-ADR ;
11
12 (P BUFPOS RETURNS THE ADDRESS IN THE DISK BUFFER OF THE
13 CURRENT CHARACTER )
14
15 -->
```

```
     47
\ E-UPDATE                                             25APR81HHL

: E-UPDATE    (S --- )
   1 &UPDATE ! ;                \ SET UPDATE FLAG

(P E-UPDATE IS CALLED WHENEVER THE CONTENTS OF THE BUFFER
HAS CHANGED.  IT SETS AN THE UPDATE FLAG. )

-->
```

```
     48
 0 \ BUF-MOVE                                          25APR81HHL
 1
 2 : BUF-MOVE   (S FROM TO LEN --- )
 3    ROT BUF-ADR
 4    ROT BUF-ADR
 5    ROT MOVE
 6    E-UPDATE ;
 7
 8 (P BUF-MOVE PERFORMS A MOVE OPERATION ON THE CHARACTERS IN THE
 9 DISK BUFFER CORRESPONDING TO THE GIVEN CURSOR POSITIONS. )
10
11 -->
12
13
14
15
```

```
     49
\ ?PRINTABLE                                           25APR81HHL

: ?PRINTABLE   (S CHAR --- BOOL )
   DUP 32 <
   SWAP 126 >
   OR 0= ;

(P ?PRINTABLE RETURNS A TRUE FLAG IF THE CHARACTER IS PRINTABLE.
OTHERWISE IT RETURNS A FALSE FLAG )

-->
```

```
     50
 0 \ >LINE# LINE#>                                     25APR81HHL
 1
 2 : >LINE#    (S POS --- LINE# )
 3    C/L / ;
 4
 5 (P CONVERT A CHARACTER POSITION TO A LINE NUMBER )
 6
 7 : LINE#>    (S LINE# --- POS )
 8    C/L * ;
 9
10 (P CONVERT A LINE NUMBER TO A CHARACTER POSITION )
11
12 -->
13
14
15
```

```
     51
\ CHARS-TO-EOL                                         25APR81HHL

: CHARS-TO-EOL     (S POS --- N )
   C/L MOD
   C/L SWAP - ;

(P CHARS-TO-EOL RETURNS THE NUMBER OF CHARACTERS LEFT ON THE
LINE GIVEN THE CURRENT CHARACTER POSITION )

-->
```

```
    52
 0 \ DISPLAY-TO-EOL                                  25APR81HHL
 1
 2 : DISPLAY-TO-EOL    (S POS --- )
 3    DUP BUF-ADR           \ GET ADDRESS IN BUFFER
 4    OVER CHARS-TO-EOL     \ REST OF LINE
 5    -TRAILING             \ IGNORE BLANKS
 6    ROT OVER + >R         \ SAVE RESULTANT CURSOR POSITION
 7    TYPE                  \ DISPLAY WHATS THERE
 8    R> CLEAR-TO-EOL       \ AND REMOVE THE REST
 9    ;
10
11 (P DISPLAY-TO-EOL DISPLAYS THE REST OF THE LINE STARTING FROM
12 THE CURRENT CURSOR POSITION.  IT ASSUMES THAT THE TERMINAL
13 CURSOR IS PROPERLY POSITIONED BEFORE IT EXECUTES. )
14
15 -->
```

```
    53
 0 \ ?EMPTY-LINE                                     25APR81HHL
 1
 2 : ?EMPTY-LINE    (S LINE# --- BOOL )
 3    LINE#> BUF-ADR C/L       \ ADDR LEN
 4    -TRAILING                \ REMOVE TRAILING BLANKS
 5    SWAP DROP 0=             \ REPORT SUCCESS IF ALL BLANKS
 6    ;
 7
 8 (P ?EMPTY-LINE RETURNS TRUE IF THE SPECIFIED LINE NUMBER IS
 9 COMPLETELY BLANK.  OTHERWISE IT RETURNS FALSE. )
10
11 -->
12
13
14
15
```

```
    54
 0 \ DISPLAY-TO-EOS                                  25APR81HHL
 1
 2 : DISPLAY-TO-EOS   (S LINE# --- )
 3    CURPOS SWAP              \ SAVE CURRENT CURSOR POSITION
 4    L/SCR SWAP DO            \ RUN THROUGH REST OF SCREEN
 5     I LINE#>
 6     DUP &CURSOR !           \ SET CURSOR POSITION
 7     0 MOVE-CURSOR
 8     DISPLAY-TO-EOL          \ AND DISPLAY LINE FROM THERE
 9    LOOP
10    &CURSOR !                \ RESTORE CURSOR POSITION
11    0 MOVE-CURSOR ;
12 (P DISPLAY THE ENTIRE SCREEN FROM THE GIVEN LINE NUMBER TO
13 THE END OF THE SCREEN.  THIS IS USED WHEN A LINE IS
14 INSERTED OR DELETED FROM THE MIDDLE OF THE SCREEN )
15 -->
```

```
    55
 0 \ EXPAND                                          25APR81HHL
 1
 2 : EXPAND   (S POS --- )
 3    DUP DUP                  \ P P P
 4    C/L +                    \ P FROM TO
 5    C/SCR OVER -             \ P FROM TO LEN
 6    BUF-MOVE                 \ TEXT MOVED IN BUFFER
 7    BUF-ADR C/L BLANKS       \ INSERT BLANK LINE
 8    E-UPDATE ;
 9
10 (P EXPAND MOVES ALL OF THE LINES DOWN BY ONE AND INSERTS
11 A BLANK LINE AT THE SPECIFED POSITION. )
12
13 -->
14
15
```

```
    56
 0 \ SHRINK                                          25APR81HHL
 1
 2 : SHRINK   (S POS --- )
 3    DUP                      \ POS POS
 4    C/L + SWAP               \ FROM TO
 5    OVER C/SCR SWAP -        \ FROM TO LEN
 6    BUF-MOVE                 \ MOVE IT
 7    [ L/SCR 1- ] LITERAL     \ INSERT A BLANK LINE
 8    LINE#> BUF-ADR C/L BLANKS  \ AT THE BOTTOM OF THE SCREEN
 9    E-UPDATE ;
10
11 (P SHRINK DELETES THE SPECIFIED LINE IN THE DISK BUFFER AND
12 REPLACES THE LAST LINE OF THE SCREEN WITH A BLANK LINE )
13
14 -->
15
```

```
    57
 0 \ INSERT-LINE                                     25APR81HHL
 1
 2 : INSERT-LINE   (S POS --- )
 3    [ L/SCR 1- ] LITERAL     \ LAST LINE NUMBER
 4    ?EMPTY-LINE IF           \ IS IT EMPTY?
 5     DUP EXPAND              \ YES, EXPAND THE BUFFER
 6     >LINE# DISPLAY-TO-EOS   \ AND REDISPLAY THE SCREEN
 7    ELSE
 8     BEEP                    \ NO, WARN USER
 9    ENDIF ;
10
11 (P INSERT-LINE CHECKS TO SEE THAT THERE IS NO TEXT ON THE
12 LAST LINE OF THE SCREEN.  IF THERE IS NONE, IT EXPANDS THE
13 SCREEN AT THE GIVEN CURSOR POSITION AND RE-DISPLAYS THE
14 ALTERED SCREEN )
15 -->
```

*(Continued on next page)*

# Screen Oriented Editor (Listing continued, text on page 27)

```
    58
 0 \ DELETE-LINE                                          25APR81HHL
 1
 2 : DELETE-LINE  (S POS --- )
 3    >LINE# DUP LINE#> SHRINK
 4    DISPLAY-TO-EOS ;
 5
 6 (P DELETE-LINE REMOVE THE LINE THE CURSOR IS ON AND RE-DISPLAYS
 7 THE RESULTING SCREEN )
 8
 9 -->
10
11
12
13
14
15
```

```
    59
 0 \ INS-CHAR                                             25APR81HHL

 : INS-CHAR   (S CHAR POS --- )
    DUP DUP 1+                \ CHAR POS FROM TO
    OVER CHARS-TO-EOL 1-      \ CHAR POS FROM TO LEN
    BUF-MOVE                  \ MOVE IT
    BUF-ADR C! ;              \ AND STICK IN CHAR

(P INS-CHAR INSERTS THE GIVEN CHARACTER INTO THE DISK BUFFER.
NOTE THAT CHARACTERS FALLING OFF THE RIGHT END OF THE LINE
ARE LOST IF CAUTION IS NOT USED. )

-->
```

```
    60
 0 \ DEL-CHAR                                             25APR81HHL
 1
 2 : DEL-CHAR   (S POS --- )
 3    DUP DUP 1+ SWAP           \ POS FROM TO
 4    OVER CHARS-TO-EOL         \ POS FROM TO LEN
 5    BUF-MOVE                  \ MOVE IT
 6    DUP CHARS-TO-EOL + 1-     \ POSITION AT EOL
 7    BUF-ADR BL SWAP C! ;      \ AND STICK IN A BLANK
 8
 9 (P DEL-CHAR DELETES THE CHARACTER AT THE SPECIFIED CURSOR
10 POSITION )
11
12 -->
13
14
15
```

```
    61
 0 \ ARROW COMMANDS                                       25APR81HHL

 : R-ARROW   (S --- )
    1 +CURPOS ;               \ MOVE RIGHT BY ONE

 : L-ARROW   (S --- )
    -1 +CURPOS ;              \ MOVE LEFT BY ONE

 : U-ARROW   (S --- )
    C/L MINUS +CURPOS ;       \ MOVE UP BY ONE

 : D-ARROW   (S --- )
    C/L +CURPOS ;             \ MOVE DOWN BY ONE

-->
```

```
    62
 0 \ I-LINE D-LINE D-CHAR INSERT-MODE                     25APR81HHL
 1
 2 : I-LINE   (S --- )
 3    CURPOS INSERT-LINE ;
 4
 5 : D-LINE   (S --- )
 6    CURPOS DELETE-LINE ;
 7
 8 : D-CHAR   (S --- )
 9    CURPOS DEL-CHAR
10    CURPOS DISPLAY-TO-EOL ;
11
12 : INSERT-MODE   (S --- )
13    &MODE 1 TOGGLE ;
14
15 -->
```

```
    63
 0 \ RETURN EXIT-EDIT                                     21APR81HHL

 : RETURN   (S --- )
    CURPOS >LINE#             \ GET LINE NUMBER OF CURRENT LIN
    1+                        \ INCREMENT BY ONE
    [ L/SCR 1- ] LITERAL MIN  \ DON'T MOVE BELOW BOTTOM
    LINE#> &CURSOR ! ;        \ AND MOVE THERE
(P RETURN IS EXECUTED WHENEVER THE CARRIAGE RETURN KEY
IS PRESSED.  IT MOVES THE CURSOR TO THE BEGINNING OF THE
NEXT LINE.  IF THE CURSOR IS AT THE BOTTOM OF THE SCREEN,
IT REMAINS THERE. )

: EXIT-EDIT   (S --- )
   CR R> DROP R> DROP R> DROP R> DROP R> DROP ;
(P GET OUT OF THE EDITOR AND RETURN TO PREVIOUS ACTIVITY )
-->
```

```
    64
 0 \ EXIT-UPDATE                                    25APR81HHL
 1
 2 : EXIT-UPDATE    (S --- )
 3    C/SCR MOVE-CURSOR           \ GET TO BOTTOM OF SCREEN
 4    CR CR                       \ SKIP TWO LINES
 5    SCR @ .                     \ TELL USER SCREEN NUMBER
 6    &UPDATE @ IF                \ HAS IT CHANGED?
 7      &E-ID                     \ FROM
 8      [ C/L 10 - ] LITERAL      \ TO
 9      BUF-ADR 10 CMOVE          \ INDICATE BLOCK CHANGED
10     ." Modified" UPDATE FLUSH  \ YES, TELL USER
11    ELSE ." Unmodified" ENDIF   \ NO, LET HIM KNOW
12    EXIT-EDIT ;                 \ GET OUT OF EDITOR
13 (P EXIT-UPDATE LEAVES THE EDITOR AND RETURNS TO FORTH.  IF
14 THE SCREEN HAS BEEN MODIFIED, THE USER ID IS INSERTED ON LINE
15 0 IN THE RIGHT HAND CORNER )  -->
```

```
    65
 0 \ EXIT-SCRATCH                                   25APR81HHL
 1
 2 : EXIT-SCRATCH   (S --- )
 3    C/SCR MOVE-CURSOR           \ GET TO BOTTOM OF SCREEN
 4    CR CR                       \ SKIP TWO LINES
 5    SCR ?                       \ TELL USER SCREEN NUMBER
 6    ." Abandoned"               \ AND WARN ABOUT STATUS
 7    EXIT-EDIT ;                 \ GET OUT OF EDITOR
 8
 9 (P EXIT-SCRATCH WILL LEAVE THE EDITOR AND NOT FLUSH THE
10 SCREEN TO DISK.  THE SCREEN IS NOT MARKED AS UPDATED, BUT
11 THIS CAN BE DONE MANUALLY WITH THE UPDATE COMMAND.  IT MAY THEN
12 BE FLUSHED WITH THE FLUSH COMMAND, OR ERASED WITH THE
13 EMPTY-BUFFERS COMMAND )
14
15 -->
```

```
    66
 0 \ E-TAB                                          25APR81HHL
 1
 2 : E-TAB   (S --- )
 3    8 CURPOS 8 MOD -
 4    +CURPOS ;
 5
 6 (P MOVE THE CURSOR TO THE NEXT TAB STOP.  TABS ARE CURRENTLY
 7 DEFINED AS BEING 8 APART, CAN BE BE RE-DEFINED BY SIMPLY
 8 ALTERING E-TAB )
 9
10 -->
11
12
13
14
15
```

```
    67
 0 \ SCAN+=                                         25APR81HHL
 1 : SCAN+=        (S CHAR ADR1 ADR2 --- N )
 2    2DUP = IF              \ RETURN ZERO IF THERE
 3      DROP 2DROP 0         \ IS NOTHING TO SEARCH
 4    ELSE
 5      0 ROT ROT DO         \ OTHERWISE RUN THROUGH MEMORY
 6       OVER I C@ = IF      \ FROM LOW TO HIGH
 7         LEAVE             \ LOOKING FOR THE SPECIFIED CHARACTER
 8       ELSE 1+ ENDIF
 9      LOOP
10      SWAP DROP            \ RETURN RESULT
11    ENDIF ;
12 (P SCANS THROUGH MEMORY STARTING AT ADR2 TO ADR1 AND
13 INCREMENTING BY +1 LOOKING FOR THE SPECIFIED CHARACTER.
14 RETURNS THE NUMBER OF CHARACTERS SCANNED UNTIL SUCCESS )
15 -->
```

```
    68
 0 \ SCAN+<>                                        25APR81HHL
 1 : SCAN+<>       (S CHAR ADR1 ADR2 --- N )
 2    2DUP = IF              \ RETURN ZERO IF THERE
 3      DROP 2DROP 0         \ IS NOTHING TO SEARCH
 4    ELSE
 5      0 ROT ROT DO         \ OTHERWISE RUN THROUGH MEMORY
 6        OVER I C@ <> IF    \ FROM LOW TO HIGH
 7          LEAVE            \ UNTIL ANY CHARACTER BUT THE
 8        ELSE 1+ ENDIF      \ SPECIFIED ONE IS SEEN
 9      LOOP
10      SWAP DROP            \ RETURN RESULT
11    ENDIF ;
12 (P SCANS THROUGH MEMORY STARTING AT ADR2 TO ADR1 AND
13 INCREMENTING BY +1 UNTIL ANY CHARACTER NOT MATCHING THE ONE
14 SPECIFIED IS FOUND.  RETURNS COUNT OF CHARACTERS SCANNED )
15 -->
```

```
    69
 0 \ SCAN-=                                         25APR81HHL
 1 : SCAN-=        (S CHAR ADR1 ADR2 --- N )
 2    2DUP = IF              \ RETURN ZERO IF THERE
 3      DROP 2DROP 0         \ IS NOTHING TO SEARCH
 4    ELSE
 5      0 ROT ROT DO         \ OTHERWISE RUN THROUGH MEMORY
 6       OVER I C@ = IF      \ FROM HIGH TO LOW
 7         LEAVE             \ LOOKING FOR THE SPECIFIED CHARACTER
 8       ELSE 1- ENDIF
 9      -1 +LOOP
10      SWAP DROP            \ RETURN RESULT
11    ENDIF ;
12 (P SCANS THROUGH MEMORY STARTING AT ADR2 TO ADR1 AND
13 DECREMENTING BY -1 LOOKING FOR THE SPECIFIED CHARACTER.
14 RETURNS THE NUMBER OF CHARACTERS SCANNED UNTIL SUCESS )
15 -->
```

*(Continued on next page)*

```
     70
 0 \ SCAN-<>                                          25APR81HHL
 1 : SCAN-<>       (S CHAR ADR1 ADR2 --- N )
 2    2DUP = IF             \ RETURN ZERO IF THERE
 3      DROP 2DROP 0        \ IS NOTHING TO SEARCH
 4    ELSE
 5      0 ROT ROT DO        \ OTHERWISE RUN THROUGH MEMORY
 6        OVER I C@ <> IF   \ FROM HIGH TO LOW
 7          LEAVE           \ UNTIL ANY CHARACTER BUT THE
 8        ELSE 1- ENDIF     \ SPECIFIED ONE IS SEEN
 9      -1 +LOOP
10      SWAP DROP           \ RETURN RESULT
11    ENDIF ;
12 (P SCANS THROUGH MEMORY STARTING AT ADR2 TO ADR1 AND
13 DECREMENTING BY -1 UNTIL ANY CHARACTER NOT MATCHING THE
14 SPECIFIED ONE IS FOUND.  RETURNS COUNT OF CHARACTERS SCANNED )
15 -->
```

```
     71
 0 \ MOVE-LEFT-WORD                                   25APR81HHL
 1 : MOVE-LEFT-WORD   (S --- N )
 2    BL 0 BUF-ADR BUFPOS          \ SCAN BACKWARDS FOR THE
 3    SCAN-= >R                    \ FIRST BLANK
 4    BL 0 BUF-ADR BUFPOS R +      \ NOW SCAN BACKWARDS FOR THE
 5    SCAN-<> R> + >R              \ FIRST NON BLANK
 6    BL 0 BUF-ADR BUFPOS R +      \ FINALLY LOOK FOR THE FIRST
 7    SCAN-= R> +                  \ BLANK AGAIN.
 8    DUP BUFPOS + C@ BL = IF      \ CORRECT FOR THE POSSIBILITY
 9     1+                          \ THAT BLANK WAS NOT FOUND
10    ENDIF ;                      \ AT THE BEGINNING OF THE SCREEN
11 (P RETURNS THE NUMBER OF CHARACTERS THAT MUST BE SKIPPED TO
12 MOVE TO THE BEGINNING OF THE PREVIOUS WORD, RELATIVE TO THE
13 CURRENT CURSOR POSITION.  THE NUMBER RETURNED IS ALWAYS LESS
14 THAN OR EQUAL TO ZERO )
15 -->
```

```
     72
 0 \ MOVE-RIGHT-WORD                                  25APR81HHL
 1
 2 : MOVE-RIGHT-WORD   (S --- N )
 3    BL [ C/SCR 1- ] LITERAL BUF-ADR
 4    BUFPOS SCAN+= >R
 5    BL [ C/SCR 1- ] LITERAL BUF-ADR
 6    BUFPOS R +
 7    SCAN+<> R> + ;
 8
 9 (P RETURNS THE NUMBER OF CHARACTERS THAT MUST BE SKIPPED TO
10 MOVE TO THE BEGINNING OF THE NEXT WORD RELATIVE TO THE CURRENT
11 CURSOR POSITION.  )
12
13 -->
14
15
```

```
     73
 0 \ R-WORD L-WORD                                    25APR81HHL
 1
 2 : R-WORD   (S --- )
 3    MOVE-RIGHT-WORD          \ MOVE FORWARD 1 WORD
 4    +CURPOS ;                \ AND UPDATE CURSOR
 5 (P R-WORD MOVES THE CURSOR RIGHT 1 WORD.  THE CURSOR IS LEFT
 6 AT THE BEGINNING OF THE WORD.  IF THERE ISN'T ANY, THE CURSOR
 7 MOVES TO THE END OF THE SCREEN )
 8
 9 : L-WORD   (S --- )
10    MOVE-LEFT-WORD           \ MOVE BACKWARDS 1 WORD
11    +CURPOS ;                \ AND UPDATE CURSOR
12 (P L-WORD MOVES THE CURSOR LEFT 1 WORD.  THE CURSOR IS LEFT
13 AT THE END OF THE PREVIOUS WORD.  IF THERE ISN'T ANY, THE
14 CURSOR MOVE TO THE BEGINNING OF THE SCREEN )
15 -->
```

```
     74
 0 \ DEL-CHARS                                        25APR81HHL
 1
 2 : DEL-CHARS     (S N POS --- )
 3    2DUP + OVER          \ N P FROM P
 4    DUP CHARS-TO-EOL     \ N P F P L
 5    BUF-MOVE             \ N P
 6    DUP CHARS-TO-EOL +   \ N EOL
 7    OVER - BUF-ADR       \ N EOL-N
 8    SWAP BLANKS ;        \ FILL END WITH BLANKS
 9
10 (P DEL-CHARS DELETES N CHARACTERS STARTING AT POSITION POS.
11 THIS IS USED MAINLY FOR DELETING ENTIRE WORDS AT ONE TIME.
12 IT IS MUCH FASTER THAN CALLING DEL-CHAR N TIMES FOR MOST
13 WORDS )
14
15 -->
```

```
     75
 0 \ D-WORD                                           25APR81HHL
 1
 2 : D-WORD    (S --- )
 3    MOVE-RIGHT-WORD          \ MOVE OVER 1 WORD
 4    CURPOS BUF-ADR           \ BUT LESS THAN LAST BLANK
 5    CURPOS CHARS-TO-EOL      \ ON THE CURRENT LINE
 6    -TRAILING SWAP DROP      \ FOR SPEED
 7    MIN CURPOS DEL-CHARS     \ AND DELETE TEXT
 8    CURPOS DISPLAY-TO-EOL ;  \ AND SHOW RESULT
 9
10 (P D-WORD DELETES THE NEXT WORD IN THE INPUT STREAM.  NOTE
11 THAT CHARACTER AND WORD DELETION AFFECT ONLY THE CURRENT
12 LINE )
13
14 -->
15
```

```
    76
 0 \ U-TAB D-TAB CRL-SCREEN                          25APR81HHL
 1 : U-TAB   (S --- )
 2   4 C/L *             \ MOVE UP 4 LINES
 3   MINUS +CURPOS ;
 4
 5 : D-TAB   (S --- )
 6   4 C/L *             \ MOVE DOWN 4 LINES
 7   +CURPOS ;
 8
 9 : CLR-SCREEN   (S --- )
10   0 &CURSOR !         \ RESET CURSOR
11   CURPOS BUF-ADR      \ GET BUFFER ADDRESS
12   C/SCR BLANKS        \ AND SET ALL TO BLANKS
13   0 DISPLAY-TO-EOS    \ AND RE DISPLAY
14   E-UPDATE ;          \ INDCATE SCREEN CHANGED
15 -->
```

```
    77
 0 \ DISPLAY-STATUS                                  21APR81HHL

 : DISPLAY-STATUS   (S --- )
   &MODE @ &OLD-MODE @ <> IF   \ HAS MODE CHANGED?
    40 0 CRTXY                 \ MOVE CURSOR
    &MODE @ IF                 \ 1=INSERT 0=OVERSTRIKE
     ." Insert ON"             \ DISPLAY MESSAGE ON THE
    ELSE                       \ STATUS LINE
     9 SPACES
    ENDIF
    &MODE @ &OLD-MODE !        \ RESET OLD MODE
   ENDIF
   CURPOS C/L /MOD             \ CHAR POS, LINE#
   35 0 CRTXY 2 .R             \ DISPLAY LINE#
   28 0 CRTXY 2 .R ;           \ DISPLAY CHAR#
 -->
```

```
    78
 0 \ CLR-LINE                                        25APR81HHL
 1
 2 : CLR-LINE   (S --- )
 3   CURPOS DUP                  \ SAVE CURRENT CURSOR POSITION
 4   >LINE# LINE#) &CURSOR !     \ GET TO BEGINNING OF LINE
 5   CURPOS BUF-ADR              \ BUFFER ADDRESS OF BOL
 6   C/L BLANKS                  \ BLANK IT OUT
 7   E-UPDATE                    \ INDICATE TEXT HAS CHANGED
 8   0 MOVE-CURSOR               \ GET TO BEGINNING
 9   CURPOS CLEAR-TO-EOL         \ AND CLEAR THE LINE
10   &CURSOR ! ;                 \ RESTORE THE CURSOR
11
12 (P CLR-LINE SETS THE CURRENT LINE TO BLANKS )
13
14 -->
15
```

```
    79
 0 \ GET-USER-ID                                     25APR81HHL

 : GET-USER-ID   (S --- )
   &E-ID 10 -TRAILING 0= IF     \ IS USER ID BLANK?
    CR ." Enter Your ID: "      \ PROMPT USER
    10 0 DO 46 ( . ) EMIT LOOP \ DISPLAY FIELD LENGTH
    10 0 DO 8 ( BS ) EMIT LOOP \ AND BACK UP
    10 EXPECT                   \ LET THE USER ENTER IT
    &E-ID 10 -TIDY              \ REPLACE CONTROL CHARS WITH BLK
   ELSE                         \ ALREADY ENTERED USER ID
    DROP                        \ IF HERE
   ENDIF ;

 (P GET-USER-ID CHECKS TO SEE IF THE USER'S ID HAS BEEN SET, AND
 IF NOT, PROMPTS HIM FOR IT AND SAVES IT )
 -->
```

```
    80
 0 \ CONTROL CHARACTER DEFINITIONS                   21APR81HHL
 1
 2 CASE: (CONTROL-CHAR)   (S N --- )
 3
 4   BEEP         \  0: ^@ --- ERROR
 5   L-WORD       \  1: ^A --- LEFT WORD
 6   CLR-LINE     \  2: ^B --- CLEAR LINE
 7   D-TAB        \  3: ^C --- DOWN 4 LINES
 8   R-ARROW      \  4: ^D --- RIGHT ARROW
 9   U-ARROW      \  5: ^E --- UP ARROW
10   R-WORD       \  6: ^F --- RIGHT WORD
11   D-CHAR       \  7: ^G --- DELETE CHAR
12   L-ARROW      \  8: ^H --- LEFT ARROW
13   E-TAB        \  9: ^I --- TAB TO NEXT COLUMN
14 -->
15
```

```
    81
 0 \ CONTROL CHARACTER DEFINITIONS                   21APR81HHL

   BEEP         \ 10: ^J --- ERROR
   CLR-SCREEN   \ 11: ^K --- CLEAR SCREEN
   BEEP         \ 12: ^L --- ERROR
   RETURN       \ 13: ^M --- CARRIAGE RETURN
   I-LINE       \ 14: ^N --- INSERT LINE
   BEEP         \ 15: ^O --- ERROR
   BEEP         \ 16: ^P --- ERROR
   BEEP         \ 17: ^Q --- ERROR
   U-TAB        \ 18: ^R --- UP 4 LINES
   L-ARROW      \ 19: ^S --- LEFT ARROW
 -->
```

*(Continued on next page)*

# Screen Oriented Editor (Listing continued, text on page 27)

```
    82
 0 \ CONTROL CHARACTER DEFINITIONS
 1
 2    D-WORD       \ 20: ^T --- DELETE WORD
 3    BEEP         \ 21: ^U --- ERROR
 4    INSERT-MODE  \ 22: ^V --- TOGGLE INSERT MODE
 5    BEEP         \ 23: ^W --- ERROR
 6    D-ARROW      \ 24: ^X --- DOWN ARROW
 7    D-LINE       \ 25: ^Y --- DELETE LINE
 8    EXIT-SCRATCH \ 26: ^Z --- ABANDON SCREEN
 9    EXIT-UPDATE  \ 27: ESC -- EXIT EDITOR NORMALLY
10    ;
11
12 -->
13
14
15
```

```
    83
25APR81HHL \ CONTROL-CHAR                                  25APR81HHL

: CONTROL-CHAR  (S CHAR --- )
   DUP 127 = IF          \ IF ITS A DELETE
     DROP 8              \ THEN TURN IT INTO
   ENDIF                 \ A BACKSPACE
   DUP 28 < IF           \ MIGHT IT BE VALID?
     (CONTROL-CHAR)      \ YES, SO GO DO IT
   ELSE
     DROP BEEP           \ NO, COMPLAIN
   ENDIF ;

(P PROCESS A CONTROL CHARACTER.  IF THE CHARACTER IS A DELETE,
IT IS CHANGED INTO A BACKSPACE.  IF IT IS LESS THAN OR EQUAL TO
AN ESCAPE, IT IS EXECUTED, OTHERWISE WE BEEP. )
-->
```

```
    84
 0 \ E-OVERSTRIKE                                   25APR81HHL
 1
 2 : E-OVERSTRIKE   (S --- )
 3    KEY DUP                   \ GET NEXT KEYSTROKE
 4    ?PRINTABLE IF             \ IF ITS PRINTABLE
 5     DUP EMIT                 \ SHOW IT ON THE SCREEN
 6     BUFPOS C!                \ STICK IT IN THE BUFFER
 7     E-UPDATE                 \ BUFFER HAS CHANGED
 8     1 +CURPOS                \ AND MOVE THE CURSOR
 9    ELSE
10     CONTROL-CHAR             \ ELSE PROCESS IT AS A COMMAND
11    ENDIF ;
12 (P E-OVERSTRIKE IS CALLED WHENEVER THE EDITOR IS IN
13 OVERSTRIKE MODE.  NOTE THAT ONLY A SINGLE CHARACTER IS
14 PROCESSED, AND CONTROL IS ALWAYS RETURNED TO THE MAIN
15 PROCESSING LOOP )  -->
```

```
    85
25APR81HHL \ E-INSERT                                      25APR81HHL

: E-INSERT    (S --- )
   KEY DUP                   \ GET THE NEXT CHARACTER
   ?PRINTABLE IF             \ CHECK IF ITS PRINTABLE
    CURPOS INS-CHAR          \ IF SO, INSERT IT HERE
    CURPOS DISPLAY-TO-EOL    \ RE-DISPLAY THE LINE
    1 +CURPOS                \ AND MOVE OVER 1
   ELSE
    CONTROL-CHAR             \ ELSE PROCESS THE COMMAND
   ENDIF ;

(P E-INSERT IS CALLED WHENEVER THE EDITOR IS IN INSERT MODE.
NOTE THAT ONLY A SINGLE CHARACTER IS PROCESSED AND CONTROL IS
RETURNED IMMEDIATELY TO THE CALLING ROUTINE. )
-->
```

```
    86
 0 \ E-INIT                                         21APR81HHL
 1
 2 : E-INIT   (S [N] --- )
 3    DEPTH IF SCR ! ENDIF      \ EDIT LAST SCREEN IF STACK EMPTY
 4    SCR @ BLOCK &BUF-ADR !    \ SAVE THE BUFFER ADDRESS
 5    GET-USER-ID CRTCLR-SCR    \ GET DATE AND NAME & CLEAR SCREEN
 6    0 &MODE ! 0 &CURSOR !     \ INIT VARIABLES
 7    0 &UPDATE !               \ NOT MODIFIED
 8    0 %Y-OFF CRTXY            \ MOVE CURSOR TO START
 9    L/SCR 0 DO                \ DISPLAY LINE NUMBERS FOR USER
10     I 3 .R CR
11    LOOP
12    10 0 CRTXY                \ MOVE TO STATUS LINE
13    ." Scr: " SCR @ 4 .R  6 SPACES ." X=     Y="
14    0 DISPLAY-TO-EOS ;        \ SHOW THE SCREEN
15 -->
```

```
    87
25APR81HHL \ E                                             25APR81HHL
                FORTH DEFINITIONS
: E    (S [N] --- )
   EDITOR                \ LOOK THROUGH THE EDITOR VOCABULARY
   E-INIT                \ INITIALIZE THE SCREEN
   BEGIN                 \ THIS IS THE ONLY LOOP IN THE EDITOR
    DISPLAY-STATUS       \ DISPLAY THE STATUS ON LINE 0
    0 MOVE-CURSOR        \ MOVE THE CURSOR TO WHERE IT SHOULD BE
    &MODE @ IF           \ CHECK THE MODE, 1=INSERT 0=OVERSTRIKE
     E-INSERT
    ELSE
     E-OVERSTRIKE
    ENDIF
   AGAIN ;
(P USED TO INVOKE THE EDITOR.  SCREEN NUMBER SHOULD BE ON STK )
-->
```

```
    88
 0 \ ASK USER ABOUT HIS TERMINAL                    26APR81HHL
 1
 2 CR ." If your terminal is one of the ones listed below"
 3 CR ." just enter the corresponding number.  If not, you"
 4 CR ." will have to write your own cursor addressing routines"
 5 CR CR
 6 CR ." 0 = None of the below"
 7 CR ." 1 = Perkin Elmer Bantam"
 8 CR ." 2 = Lear Ziegler ADM3"
 9 CR CR
10 CR ." Enter your Terminal number please " RE-FORTH
11
12    EDITOR DEFINITIONS 89  + LOAD FORTH DEFINITIONS
13
14 CR ." Ready to Edit" CR CR
15
```

```
    89
 0 \ NO SUCH TERMINAL                               26APR81HHL
 1
 2 CR
 3 CR
 4 CR ." Well, take a look at SCREENS 42 - 44 "
      ." to see how to proceed"
 5 CR ." If you would like me to include your terminal routines"
 6 CR ." in the next release of this editor, please send me"
 7 CR ." the code you wrote to drive your terminal.  Thanks."
 8 CR ." Good luck, and may the FORTH be with you"
 9 CR CR
10
11
12
13
14
15
```

```
    90
 0 \ PERKIN-ELMER CURSOR ROUTINES                   19SEP80HHL
 1
 2 : PE-CRTXY    (S X Y --- )
 3    32 + 88 27 EMIT EMIT EMIT
 4    32 + 89 27 EMIT EMIT EMIT ;
 5
 6 : PE-CRTCLR-SCR   (S --- )
 7    75 27 EMIT EMIT
 8    10 0 DO 0 EMIT LOOP ;
 9
10 : PE-CRTCLR-EOL   (S N --- )
11    DROP 73 27 EMIT EMIT ;
12
13 ' PE-CRTXY        CFA  'CRTXY !
14 ' PE-CRTCLR-SCR   CFA  'CRTCLR-SCR !
15 ' PE-CRTCLR-EOL   CFA  'CLEAR-TO-EOL !
```

```
    91
 0 \ ADM-3 CURSOR ROUTINES                          19SEP80HHL
 1
 2 : ADM3-CRTXY     (S X Y --- )
 3    27 EMIT 61 EMIT  ( ESC = )
 4    32 + EMIT 32 + EMIT ;
 5
 6 : ADM3-CRTCLR-SCR     (S --- )
 7    26 EMIT    ( ^Z )
 8    10 0 DO 0 EMIT LOOP ;
 9
10 : ADM3-CRTCLR-EOL     (S N --- )
11    C/L MOD C/L SWAP - SPACES ;
12
13 ' ADM3-CRTXY      CFA  'CRTXY !
14 ' ADM3-CRTCLR-SCR CFA  'CRTCLR-SCR !
15 ' ADM3-CRTCLR-EOL CFA  'CLEAR-TO-EOL !
```

```
     0
 0 \ PERKIN-ELMER CURSOR ROUTINES                   19SEP80HHL
 1
 2 : PE-CRTXY    (S X Y --- )
 3    32 + 88 27 EMIT EMIT EMIT
 4    32 + 89 27 EMIT EMIT EMIT ;
 5
 6 : PE-CRTCLR-SCR   (S --- )
 7    75 27 EMIT EMIT
 8    10 0 DO 0 EMIT LOOP ;
 9
10 : PE-CRTCLR-EOL   (S N --- )
11    DROP 73 27 EMIT EMIT ;
12
13 ' PE-CRTXY        CFA  'CRTXY !
14 ' PE-CRTCLR-SCR   CFA  'CRTCLR-SCR !
15 ' PE-CRTCLR-EOL   CFA  'CLEAR-TO-EOL !
```

```
     1
 0 \              The Rest is Silence               25JAN81HHL
 1
 2 ********************************************************
 3 *                                                      *
 4 *    Please direct all questions, comments, and        *
 5 *    misc. personal abuse to:                          *
 6 *                                                      *
 7 *              Henry Laxen                             *
 8 *              1259 Cornell Ave.                       *
 9 *              Berkeley, California                    *
10 *              96706                                   *
11 *                                                      *
12 *              (415) 525-8582                          *
13 *                                                      *
14 ********************************************************
15
```

```
OK
```

**End Listing**

# Elements of a FORTH Data Base Design

*In the spirit of the FORTH Interest Group, this discussion and the FORTH screens are placed in the public domain. However, further distribution must include this notice and an appropriate acknowledgment.*

In this day and age, data base design and manipulation is one of the major activities best accomplished with computers. In practice, FORTH proves to be an ideal language for developing and using custom data bases. By comparison with other languages, high or low level, FORTH is a winner. It meets the requirements of being interactive and providing documentation as identified by Fred Brooks in his *The Mythical Man-Month* as ideal for the development of new systems. The amazing speed and ease with which custom data bases can be developed more than justifies the effort required to learn FORTH.

I have developed a number of small data bases of up to 800 records containing 128 bytes each to serve my specific needs. I have also initialized, with simple formatted input and output routines, a custom data base for inventory control in a few hours one evening. Having used languages other than FORTH for similar work, it is highly unlikely I will ever go back to them.

This discussion presents a group of utility FORTH word definitions which facilitate the development of custom data bases and a sample application using utilities to define a small file. A number of techniques available in FORTH are illustrated.

Some months ago, at a regular monthly meeting of the FORTH Interest Group in Hayward, California, the prime mover of the group distributed and discussed several FORTH screens which provided the foundation for beginning the definition of a data base file. I have modified his screens slightly and expanded them to provide a general framework with which to define custom accounting data bases. I will assume that the reader has some knowledge of the fig-FORTH Model and proceed with the examination of screens developed from it. In this discussion, FORTH words are in all capital letters to set them apart from the English words in the text.

The first two screens provide eight utility FORTH words for developing a data base file. The comments included within parentheses in the screens should, combined with the mnemonic nature of the words, give you a clue to what is happening. The first two words are variables used in manipulating the file, REC# and OPEN. 2@ is a FORTH word, and alias for D@, which fetches the next two values beginning with the address on the top of the stack and places them on the stack. The word, LAYOUT, places two parameters of the new definition of a file on the stack for subsequent use. READ is the first word one will have occasion to use in routine manipulating records in the data base. It takes the number of the desired record from the top of the stack and, after checking to see that it is a valid record, places its value in the variable REC# which is used to identify the record then under consideration. The word RECORD takes the value for a record number from the stack and returns its address to the stack. Finally, ADDRESS takes the record number at the variable REC# and using RECORD leaves the address of that record on the top of the stack. With only these eight FORTH words: two variables, one utility word, and four basic words for file reference, we can proceed to the definition of three defining words in the next screen.

The three words on the next screen are called defining words because they are used to define new FORTH words as the names of fields in our record and to define the name for the file we are defining, each with specific properties. These words utilize the combination of the FORTH primitives <BUILDS and DOES> which are present in the model. It may take some time to fully appreciate what these primitive words accomplish and the way they work. Perhaps an examination of what they are doing in this screen will help you understand their function.

Two types of record fields are distinguished and defined with separate words, a numerical or data field and a text field. The first word, DFIELD, is used to add to a record being defined, a field containing the number of bytes given on the top of the stack and gives that field a name. In subsequent use, that newly defined word (data field name) will cause the address of that field in the record whose value is currently in the variable REC# to be left on the stack. This word is used to identify the location in a record where a numerical value is to be stored in a binary form. I call it a "data field," in contrast with a "text field" in which the length of the field should also be immediately available. Thus TFIELD is used to define a "text field" which will identify a field in the new record with a length equal to the number of bytes given on the top of the stack, and gives that field a name. In subsequent use, that newly defined word (text field name) will cause not only the address of that text field in the record whose value is currently in the variable REC# to be left on the stack, but also the length of that field. The length is convenient when the primitive word TYPE is used to print the character string in that field. Obviously, the length is not needed in a data field. Thus provisions are made for defining two types of fields in a record. As new fields are added to a record in the course of its definition, the current length of the record is maintained on the top of the stack.

Once the definition of the fields in a record is completed, the value of the record length remains on the stack. To this we need to add values for the number of records we wish to include and finally the block number in which the records are to start, before we can use the defining word FILE to give the file a name. Later when the new file name is used, the address of the necessary file parameters is placed in the recently defined variables OPEN as required for access to any given record with the words defined in the first screen.

With these two screens we have the file utilities necessary to define a new file. However, several characteristics of the particular implementation of FORTH which is being used are important. Most systems created under the model have 128 bytes per block although any multiple of 128 can be used. In these systems then, the largest record length can be no longer than 128 bytes, but with a larger block size, larger records can be used. In order to take maximum advantage of the block size, it should be very nearly equal to a multiple of the record length. For example, a record length of 70 bytes would not leave enough room in a 128 byte block for a second record and in this case

**by Glen B. Haydon**

*Glen B. Haydon, Haydon Enterprises, Box 429 Route 2, La Honda CA 94020.*

*Copyright © 1981 by Haydon Enterprises. All Rights Reserved.*

58 bytes of space would be wasted. If need be, such a designed file would work, but at the expense of memory space. Also, the initial block to be used in the definition created by the word FILE must be chosen according to the block size for the particular implementation. For example, block 400 in an implementation with 128 bytes per block would be block 50 in an implementation with 1024 bytes per block. Although I find a block size of 1024 to be more efficient and use it routinely, the screens presented here have been written for and tested on an implementation with 128 bytes per block.

Before starting with a discussion of an example of the application of these file development utilities, several screens of utilities for use in the input and output of numerical data will prove to be most helpful. These include a group of double precision utilities, date compression and expansion routines, and numerical routines for handling dollar amounts and storing them as double precision integers.

The double precision integer utilities are used in date compression and expansion as well as in the double precision integer operations for dollar amounts. These are simple extensions from the limited double precision words found in the model and should require no further explanation. The input on the stack before executing the word and the output left on the stack afterwards are indicated in the format used in the fig-FORTH Glossary. You will note that several of these are mixed double and single precision operations which are sufficient for the requirements of this program.

The date compression routine is really simple. When I find the time I will develop an algorithm to convert the date to a true Julian day and store the least significant value. This would make calculation of the time between two given dates easy. In the meantime, the present routine allows one to enter the date as numerical values separated by slashes, a commonly used format, and reduce the value to a single 16-bit integer requiring only two bytes for storage. The routine provides an example of using a delimiter other than a space to parse WORD and the use of NUMBER to interpret a numerical value without searching the dictionary. After the parsing of the input, three double precision numbers are left on the stack. The word DATEBIT defines a simple algorithm which is applied to reduce these three double precision values to 16 bits. The execution of ?DATE first prompts with the format to be used, then waits for the value to be entered. The value is then converted to the 16 bits and left on the stack for storage. Since . is used to connote "print" in FORTH, .DATE is defined to print a properly formatted date from a 16-bit integer on the stack. This routine is useful as an example of conversion of a binary value to a text string for printing.

Finally, we have a screen to define some FORTH words used to input and output dollar amounts and convert them to and from double precision, 32-bit integers with the necessary scaling for the location of the decimal point. In FORTH, the use of a decimal point forces an input number to a double precision integer which takes four bytes. A convenient FORTH primitive word, DPL for decimal point locator, keeps a count on the number of digits entered following the decimal point. Utilizing this value as an input for a case type word, the numerical value entered can be scaled properly, regardless of how many digits are entered to the right of the decimal, if any. This method of executing a case-like routine is straightforward. First, the action to be taken in each case is defined. 0SCALE means that there were no digits to the right of the decimal, which requires that the entered double precision integer must be multiplied by 100. In a similar manner 1SCALE is used meaning that there was only one digit entered following the decimal point and the entered double precision integer must be multiplied by 10. 2SCALE doe nothing, since no scaling is needed. Finally, if more than 2 digits are entered, an error must have been made and an appropriate error message is given. Once each of these cases is defined, their code filed addresses, CFA, can be stored beginning with the address of a defined variable NSCALE and extending into the allotted space. The word SCALE then finds the value of the variable DPL and counts over to the proper code field address which is then placed on the stack and the selected word is executed.

After this scaling operation, the word to input a dollar amount ?$AMOUNT is defined, which leaves the scaled double precision integer on the stack ready to be stored. Finally, a routine defined by the word .$AMOUNT connoting "print dollar amount," will print the double precision integer on the top of the stack as a dollar value right justified in eight spaces.

There are certainly other and probably better ways to accomplish the work done by these three screens of utilities, but they work. The way they work provides some examples of the beauty of FORTH as it exists in the model.

With these five screens, we can very quickly define a record for a data base with custom selected fields and then the associated file characteristics. In the past, I have several times included in a data base values calculated from other values in the base. On occasion, it has been necessary to change one of the original values. This has always required that the calculated fields be re-done too. I now find that it is more convenient to enter only the basic data. All calculations can be made while the output is being formatted and printed with no significant loss of time. The slowest part of printing the formatted result is the delay in the output device.

As an example of the definition of a new data base, I have chosen one in which each record would be allotted 4 fields for a two byte tag, a 30 byte stock name, a two byte date, and a 4 byte stock price. Though little could be done with this as a data base, it does provide an example of each type of input. Finally, a simple set of routines is given to clear the records, input new records, and print out a list of the records in the file.

As a matter of convention, I give each field a name with no prefix. Thus a data field name will leave an address on the stack and a text field name will leave an address and count on the stack. By using the FORTH connotations of ! for store and . for print, I define some utilities for inputting data and text, and for printing out the respective fields. From these utilities I can assembie an input format and an output format as desired. I have not included routines for error checking, which would be most desirable especially in a hostile environment.

Now to examine the actual example of the definition of a file which we will call DEMO. Each record will begin with zero offset from the record address and a 0 is entered followed by 2 for a two-byte length of a data field to be named TAG. Many occasions in later manipulation of records make it desirable to have such a field for adding flags, etc. Following this definition the length value of 2 is left on

the stack so that for the next field only its length need be entered. In this case a text field of 30 bytes which is given the name NAME which then leaves the value of 32 (the length of the TAG field plus the NAME field) on the stack. Then a two-byte data field, DAY, is reserved for a 16-bit compressed date and then a four-byte data field DOLLAR, for a double precision integer value of a dollar amount. With this, the 4 fields within the record of a new file are defined. Next we will define the file name. According to the utility for generating a new file, we must first add to the value of the record length remaining on the stack, a value for the number of records we plan to include in the file and then the first block number to be used as determined by the FORTH implementation in use. Then, we use the word FILE to create a file with these parameters and give it the name DEMO. The data base file is now defined. For the record number whose value is in the variable REC#, we can place the value of the address of the data fields, the address and count of the text fields on the stack by simply entering the field name. Next a few simple utilities will make accessing these new fields easier.

Remembering the connotations associated with the FORTH words ! and . we will define words to input data or text to the appropriate fields of that record whose value is currently in the variable REC#. These are simple file primitives which will then be available for routines to format input and output as desired. The field TAG is not used at this time and specific routines are not defined. To store a name in the name field we define the word !NAME. This routine first fills the existing field with blanks, ASCII 32 (decimal) and then pauses for input from the keyboard. The input text is truncated to the maximum length of the text field if necessary and then moved to that field. In order to output the name in the field we define the word .NAME. In a similar manner we define !DAY to store a 16-bit integer value of a date which has been compressed into that field. In the earlier utilities we have already defined ?DATE which waits for a date to be input and leaves the compressed value on the stack. All that is necessary is to put the address of the field on the stack with DAY and then store the encoded date there. We then define .DAY to output the date stored in the DAY field. We get the 16-bit value stored there to the top of the stack and use the previously defined word .DATE to output it in the proper format. Finally we define !DOLLAR to parse a dollar value input with a decimal point in any location and scaled to a double precision number which is then stored in the proper field. In a similar manner, .DOLLAR is defined to format the stored double precision integer to a right justified eight-digit number preceded by a dollar sign. With these definitions, we have completed a set of FORTH words to input and output data from records in our data base.

Immediately after putting data into a record, it is often desirable to see what is actually present in that record. The values in each byte of a record can be displayed using a dump routine. Simply place the desired record address on the top of the stack by entering the record number followed by our file utility word READ and ADDRESS followed by the length of the record and the word for your dump routine. But the byte values printed out in hex or decimal are not really all that helpful. It is hard to interpret the numerical value in their byte pattern. A convenient word .REC is defined to print out the current record number followed by the formatted output of the value in each field using the above utilities and an appropriate number of spaces and carriage returns. This is the most rudimentary form of a formatted output. If desired, the output could be presented in reverse video by a slight modification of this routine. It could also be placed anywhere on the screen.

Finally, a few examples of formatting input and output routines are shown on the last screen. First, it is desirable to clear all data in a file with a word CLEAR.DATA before entering new data. This particular definition clears only the first block, all that is necessary in this application. You should be able to modify the definition of this word to meet the requirements of your application and particular implementation of FORTH.

I use the 0 record in a file for a variety of information about the file, which I can address directly from the address of its first byte without using the field definitions, or I can use specific bytes or fields in ways other than I have defined them. In this example I use the value in the integer at the field TAG in the 0 record to keep track of the last record currently in the file. When this record is cleared with CLEAR.DATA, a value of 0 is present in the location of TAG which means that there are no records present. 0 READ places the value of 0 in the variable REC# and then TAG places the address on the top of the stack and @ gets that value, the last record number used in the file. To add a new record, this value is incremented and then duplicated on the stack. The top copy is stored back in the field of TAG in the 0 record which is updated. Then the second copy is placed in the variable REC# and we are ready to fill in the information for the next record.

A series of prompts can be formatted on the screen in any convenient arrangement as in this example. Following the desired prompt for each field, the previously defined word is used to get the information for the field and store it there. After entering a record, it is always nice to see the data you actually put in. This is done with the word .REC followed by the FORTH primitive UPDATE to flag the buffer as altered and FLUSH to save the new record on the disk in the file. This assures that the image of the record which is displayed is the version saved on the disk.

An output format can be developed in a similar manner. In this example I have included a check to see if there are any records in a file because the DO . . . LOOP will always print one loop and peculiar output is generated if the bytes in the fields are all set to zero. This output routine presents a simple list of the record numbers and the formatted content of the fields.

In conclusion, I find this approach to file definition is time saving and hope that you will find it useful. The discussion of the FORTH utilities used to define a new data base field and the example of handling data, provides some elaboration of the information included on the screens. This will be a review for one who already has learned the primitives in the FORTH model and who understands how the language works, but perhaps the discussion of these screens will help those less experienced. There is nothing sacred about the techniques used here. Modify the various words to suit your particular needs. It is easy enough to develop new formats interactively. However, I would encourage you to utilize and build on the standards of the fig-FORTH model. When the 79 Standards become generally available, it should be relatively easy to update your screens

without changing the format of the record file. The importance of utilizing an accepted standard in developing programs for ultimate use in a wide variety of implementations of FORTH cannot be over-emphasized.

I wish to thank Bill Ragsdale for his encouragement to write this discussion based on his presentation to the FORTH Interest Group at one of their monthly meetings last year.

### Application Note:

These FORTH routines have been developed on a FORTH Operating System for the Heathkit H89. This system is available from the Mountain View Press, Box 4656, Mountain View, CA 94040. The compiled FORTH program image can be saved on disk and will be up and running in less than four seconds from a cold boot. The system has 1024 byte blocks which also increases the speed of operation.

However, after development, the screens were loaded on a FORTH implementation derived from the *fig-FORTH for 8080 Assembly Source Listing* which is available from the FORTH Interest Group, Box 1105, San Carlos, CA 94070 in printed form and already on disk also from the Mountain View Press. This version has 128 byte blocks and operates in conjunction with CP/M. To this has been added the fig-EDITOR from the *fig-FORTH Installation Manual* and a single extension, DUMP, used to illustrate the appearance of the records as stored in a block.

The printed session illustrated was made using the CP/M control P to echo the output on the printer. The session starts with CP/M loaded and its usual prompt. The CP/M file, FORTH60.COM, is the object module of the fig-FORTH model. The warning messages are not on screens 4 and 5 and the warning flag is turned off. Then the screens for the fig-EDITOR and a good dump routine are loaded. Finally, the screens discussed are loaded. The file DEMO is called and the application of some of the file utilities is illustrated. This presentation will hopefully assure that there are no errors in the printed screens. ▶▶J

LISTING ON PAGE 47

### Bibliography

Brooks, F. P., Jr., *The Mythical Man-Month*, Addison-Wesley Publishing Company, 1975.

----, *fig-FORTH Installation Manual, Glossary, Model*, Forth Interest Group, Box 1105, San Carlos, CA 94070.

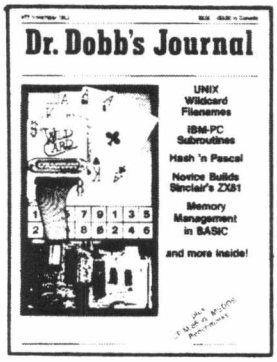
----, *fig-FORTH for 8080 Assembly Source Listing*, Forth Interest Group, Box 1105, San Carlos, CA 94070.

**Dr. Dobb's Journal**
For Users of Small Computer Systems

**Each issue includes:**
- valuable software tools
- algorithms & problem solving
- industry news
- important product reviews

**With in-depth coverage of:**
- telecommunications
- systems programming
- language development
- machine independent programs

**and much, much more!**




**Yes! Please enter my subscription for**
☐ 1 yr. (12 issues) $25 ☐ 2 yrs. $47 (save $13 off newsstand)
☐ Please bill me ☐ I enclose check/money order

Name_____

Address _____

City_____ State_____ Zip_____

**Dr. Dobb's Journal, A Publication of People's Computer Company**
**1263 El Camino Real, P.O. Box E, Menlo Park, CA 94025**

**Are you ready?**

*DDJ*, the world's foremost microcomputer publication, has been working for years to prepare its readers to be innovators, to lead the wave of breakthroughs in our changing technology.

Every issue of *Dr. Dobb's Journal* helps one to understand the nuts and bolts of small computer systems. We offer entire listings of valuable software: our pages have included compilers, cross-assemblers, editors, new languages, hardware interfaces and more – usually before anyone else thinks of them!

**Even more important!**

Our subscribers share insights, correspond, and contribute to one another's work, more than any other group we know. They treat *Dr. Dobb's Journal* as a "hands-on" publication.

This warm cooperation has done more to refine software products, and generally to advance the state of microcomputer technology, than perhaps any other resource. And it is available to you through our pages!

**For the straight facts . . . . . .**

If you are a serious computing professional or enthusiast, then you should take a very close look at what *DDJ* offers you. We've been on the cutting edge since 1976.

# FORTH Data Base Design

```
SCR # 21
  0 ( SCREEN 21: COMMENTS AND LOAD FOR DEMO DATA FILE )
  1 27 EMIT 69 EMIT CR CR CR CR CR
  2 ." This demonstration data system provides a pattern for the"
  3 CR ." further development of any type of data base.  The basic"
  4 CR ." file formating definitions are on Screens 22 and 23. Some"
  5 CR ." utilities are on 24, 25, and 26.  The demo file "
  6 CR ." definitions are on 27. Elementary file manipulation "
  7 CR ." utilities are on 28.  This model should get you started."
  8 : PROCEDE CR CR ." ENTER 'Y' TO LOAD SCREENS " KEY 89 = IF
  9         29 22 DO I LOAD LOOP ENDIF ;
 10 PROCEDE
 11 ;S
 12
 13
 14
 15

SCR # 22
  0  CR ." SCREEN 22: FILE DEVELOPMENT "
  1 0 VARIABLE REC#  ( holds the current record number )
  2 0 VARIABLE OPEN  ( points to current file descriptor )
  3
  4 : 2@  DUP 2+ @ SWAP @ ;    ( double fetch )
  5
  6 : LAYOUT  ( leave bytes/record-2, and bytes/block-1 )
  7   OPEN @ 4 + 2@ ;
  8 : READ    ( n-th record, on stack, is made current )
  9   0 MAX DUP OPEN @ 2+ @ < 0=
 10   IF ." FILE ERROR " QUIT THEN REC# ! ;
 11 : RECORD   ( leave address of n-th record )
 12    LAYOUT */MOD OPEN @ @ + BLOCK + ;
 13 : ADDRESS   ( leave address of the current record )
 14    REC# @ RECORD ;
 15 ;S

SCR # 23
  0  CR ." SCREEN 23: FILE DEVELOPMENT - 2 "
  1 : TFIELD  <BUILDS  ( create a text field   )
  2    OVER , DUP , +  ( leaves file count for this definition )
  3             DOES>  ( leaves: addr, count  )
  4   2@ ADDRESS + SWAP ;
  5 : DFIELD  <BUILDS  ( create a data field  )
  6   OVER , +         ( leaves file count for this definition )
  7            DOES>   ( leaves address  )
  8   @ ADDRESS + ;
  9 : FILE             ( create a named storage allocation )
 10    <BUILDS ,       ( origin block )
 11      1+ ,          ( number of records in file )
 12      DUP B/BUF OVER / * ,  ( # bytes per block )
 13      ,             ( # bytes per record )
 14     DOES> OPEN ! ; ( when file name used, point to )
 15 ;S                 ( its descriptor parameters.    )

MSG # 15
OK

SCR # 24
  0  CR ." SCREEN 24: DOUBLE PRECISION ARITHMETIC "
  1 : D/      ( d, u --- d    )
  2     SWAP OVER /MOD >R SWAP U/ SWAP DROP R> ;
  3 : D*      ( d, u --- d    )
  4     DUP ROT * ROT ROT U* ROT + ;
  5 : D/MOD   ( d, u --- r, q )      U/ ;
  6 : D@      ( addr --- d    )      DUP 2 + @ SWAP @ ;
  7 : D!      ( d, addr ---   )    DUP ROT SWAP ! 2 + ! ;
  8 : DDUP    ( d --- d, d    )     2DUP ;
  9 : DSWAP   ( d1, d2, --- d2, d1 )    ROT >R ROT R> ;
 10 : DDROP   ( d1, d2 ---    )      DROP DROP ;
 11 ;S
 12
 13
 14
 15

SCR # 25
  0  CR ." SCREEN 25: DATE COMPRESSION AND EXPANSION "
  1 : DATEBIT ( converts date input to 2 bytes  )
  2     32 D* D+ 13 D* D+ DROP   ;
  3 : ?DATE    ."  ( MM/DD/YY ) "
  4     QUERY 47 WORD HERE NUMBER 47 WORD HERE NUMBER BL WORD
  5     HERE NUMBER DATEBIT ;
  6 : .DATE      ( formats 2 byte date code and prints )
  7     0 13 U/ 32 /MOD ROT 100 * ROT + 0 100 D* ROT 0 D+
  8      <# # # 47 HOLD # # 47 HOLD # # #> TYPE ;
  9   ;S
 10
 11
 12
 13
 14
 15

SCR # 26
  0  CR ." SCREEN 26: ?$AMOUNT AND .$AMOUNT  "
  1  ( define action for each scale case  )
  2 : 0SCALE 100 D* ;  : 1SCALE 10 D* ;  : 2SCALE ;
  3 : 3SCALE ." INPUT ERROR " CR ;
  4  ( define scale case and extend for each with 'CFA' )
  5 ' 0SCALE CFA VARIABLE NSCALE ' 1SCALE CFA , ' 2SCALE CFA ,
  6   ' 3SCALE CFA ,
  7  ( scale double precision value according to 'DPL' )
  8 : SCALE DPL @ 3 MIN 2 * NSCALE + @ EXECUTE ;
  9  ( wait for decimal value and scale it - leave value on stack )
 10 : ?$AMOUNT QUERY BL WORD HERE NUMBER SCALE ;
 11  ( print d from stack as $ and right justify in 8 spaces )
 12 : .$AMOUNT
 13   DUP ROT ROT DABS <# # # 46 HOLD #S SIGN #>
 14   36 EMIT DUP 8 SWAP - SPACES TYPE ;
 15 ;S

MSG # 15
OK

SCR # 27
  0  CR ." SCREEN 27: DEMO FILE - RECORD GENERATION "
  1 0  2  DFIELD  TAG     ( a tag  )
  2   30  TFIELD  NAME    ( item name  )
  3    2  DFIELD  DAY     ( the date  )
  4    4  DFIELD  DOLLAR  ( a dollar amount )
  5      200 ( number of records ) 400 ( starting block )
  6      FILE  DEMO
  7 : !NAME  ( wait for name then store it in record )
  8      NAME DROP 30 32 FILL QUERY 1 TEXT PAD COUNT
  9      NAME ROT MIN CMOVE UPDATE ;
 10 : .NAME  ( print name field )   NAME TYPE ;
 11          ( the rest follow in the same way  )
 12 : !DAY ?DATE DAY ! UPDATE ;  : .DAY  DAY @ .DATE ;
 13 : !DOLLAR ?$AMOUNT DOLLAR D! UPDATE ;
 14 : .DOLLAR DOLLAR D@ .$AMOUNT ;
 15 : .REC CR REC# @ 3 .R 2 SPACES .NAME .DAY 2 SPACES .DOLLAR ;

SCR # 28
  0 CR ." SCREEN 28: DEMO FILE - CLEAR.DATA, INPUT, OUTPUT "
  1  ( clear especially tag in the 0 record in file )
  2 : CLEAR.DATA 0 READ TAG 128 0 FILL UPDATE ;
  3  ( example of formatting for input  )
  4 : INPUT 0 READ TAG @ 1+ UPDATE DUP TAG ! READ
  5    CR CR ." ENTER NAME                  --> " !NAME
  6       CR ." ENTER DATE     --> " !DAY ( has a format prompt )
  7       CR ." ENTER AMOUNT                --> " !DOLLAR
  8    .REC FLUSH ;      ( save this record on disk )
  9  ( list files 1 through the number in TAG of 0 record )
 10 : OUTPUT 0 READ TAG @ DUP 0= IF CR CR ." EMPTY FILE "
 11      DROP ELSE 1+ 1 DO FORTH I READ .REC LOOP ENDIF CR CR ;
 12 ;S
 13
 14
 15

SCR # 29
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15

MSG # 15
OK

A>
A>FORTH60       ( fig-FORTH Model )

8080 fig-FORTH 1.1
 OK
 OK
0 WARNING !     ( Warning messages not on Screens 4 & 5 )
 OK
 OK
 OK
47 LOAD         ( fig-EDITOR )
 R MSG # 4  I MSG # 4
FLUSH MSG # 4  OK
 OK
 OK
 OK
49 LOAD         ( My version of a good dump )
SCREEN 49: GOOD DUMP
OK
 OK
 OK
21 LOAD         ( Loads Screens discussed )
```

(LISTING CONTINUED ON PAGE 56)

# Forth Decompiler

*Dr. Dobb's* readers
have been aiding
the growth and development
of the micro field
for years.
If you would like
to make a contribution,
contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

## by Ray Duncan

*Ray Duncan, Laboratory Microsystems, 4147 Beethoven Street, Los Angeles, CA 90066.*

The FORTH Decompiler utility program will decode and list the "top layer" of any "high level defined" word in the glossary. This capability is useful when developing applications interactively at the keyboard; the final version of a definition can be listed on the terminal or line printer for later editing into the screen file. It is also instructive for a new FORTH programmer to decompile the various FORTH interpreter/compiler system words, both to see how they work and to learn programming style by the implementor's example.

The Decompiler is written in standard fig-FORTH and has been tested on 8080 FORTH and Z-80 FORTH. It should be directly portable to any fig-FORTH system with a "post-incrementing" inner interpreter. The branch address decoder assumes a compiled in-line offset, this word will have to be modified for other systems that compile an absolute target address.

Screen #30 contains the definition of a CASE control structure that was written by Charles Eaker. This structure essentially allows the programmer to write a convenient multiway branch without a lot of confusing nested IF's and ELSE's. The operation of the CASE construct is beyond the scope of this article, but a detailed explanation by its author may be found in FORTH Dimensions, Volume II, Number 3, page 37.

Screen #31 finds the addresses of the run-time code for each of the defining words such as VARIABLE, CONSTANT, and USER, as well as various interpreter controlling routines. Each address is turned into a constant, which is used later in the main program to trap decompilation of these definitions and to print special identifying messages.

Screens #32-34 contain a number of colon-definitions of the form xxxx.DSP, which are called by the main decompilation program to display each of the vocabulary word types and update the pseudo-code pointer appropriately.

Screens #34-38 are source text for the main control routine of the Decompiler. First the dictionary is searched for the input word, and if no match is found an error message is printed. If a match is found, the value of the Code Field Address is checked for equality with the Parameter Field Address; if true this implies that the definition is a machine code primitive and no further decompilation is attempted. If the Code Field Address points to the run-time routine for a defining word, an identifying message and the value of the parameter field are displayed. Otherwise, the program enters a loop which successively decodes each pseudo-code address in the parameter field and prints its name or value until a ";" or one of its derivatives is encountered. Execution of the Decompiler can be aborted by pushing any key on the terminal. Examples of decompilation of various vocabulary word types are shown.

Abbreviations used: NFA = name field address, CFA = code field address, PFA = parameter field address, fig-FORTH = implementation of FORTH following the Forth Interest Group model.

DDJ

**LISTING ON PAGE 50.**

### Sallying FORTH to Battle

*(Continued from page 26)*

something other than numbers and arithmetic operators are involved that the stack contents can become somewhat unclear.

Am I an opponent of FORTH? Absolutely not. FORTH or any other threaded language is both usable and fun on a small machine. I use my own version (ZIPD) fairly often. I don't use it for everything, only when it makes sense. Would I recommend FORTH on a personal computer? Again, yes. I would also recommend other langauges, dependent on the application and the environment. DDJ

**References**

[1] *FORTH-79 STANDARD*, FORTH Interest Group, P. O. Box 1105, San Carlos, CA 94070, Oct. 1980.

[2] Kernighan, B. W. and Ritchie, D. M., *The C Programming Language*, Prentice Hall, Inc., Englewood Cliffs, NJ 07632, 1978.

# FORTH Decompiler (Text on page 48)

```
Screen # 30
  0 ( fig-FORTH Decompiler )
  1 ( CASE control statement by Charles E. Eaker )
  2 ( published in FORTH Dimensions II/3 page 37 )
  3 FORTH DEFINITIONS DECIMAL
  4 : CASE            ?COMP CSP @ !CSP 4 ; IMMEDIATE
  5 : OF              4 ?PAIRS
  6                   COMPILE OVER COMPILE =
  7                   COMPILE OBRANCH HERE 0 ,
  8                   COMPILE DROP 5 ; IMMEDIATE
  9 : ENDOF           5 ?PAIRS
 10                   COMPILE BRANCH HERE 0 ,
 11                   SWAP 2 [COMPILE] ENDIF 4 ; IMMEDIATE
 12 : ENDCASE         4 ?PAIRS COMPILE DROP
 13                   BEGIN SP@ CSP @ = 0=
 14                   WHILE 2 [COMPILE] ENDIF REPEAT
 15                   CSP ! ; IMMEDIATE                -->


Screen # 31
  0 ( fig-FORTH Decompiler )
  1 0 VARIABLE QUIT.FLAG     0 VARIABLE WORD.PTR
  2 ( find run-time addresses of each vocabulary word type )
  3 ' (LOOP)          2 - CONSTANT     LOOP.ADR
  4 ' LIT             2 - CONSTANT     LIT.ADR
  5 ' :               2 - @ CONSTANT   DOCOL.ADR
  6 ' OBRANCH         2 - CONSTANT     OBRANCH.ADR
  7 ' BRANCH          2 - CONSTANT     BRANCH.ADR
  8 ' (+LOOP)         2 - CONSTANT     PLOOP.ADR
  9 ' (.")            2 - CONSTANT     PDOTQ.ADR
 10 ' C/L             2 - @ CONSTANT   CONST.ADR
 11 ' BASE            2 - @ CONSTANT   USERV.ADR
 12 ' USE             2 - @ CONSTANT   VAR.ADR
 13 ' (;CODE)         2 - CONSTANT     PSCODE.ADR
 14 -->
 15


Screen # 32
  0 ( fig-FORTH Decompiler )
  1
  2 : N.              ( print a number in decimal and hex )
  3                   DUP DECIMAL . SPACE
  4                   HEX 0  ." ( "  D.  ." H ) "   DECIMAL ;
  5
  6 : PDOTQ.DSP       ( display a compiled text string )
  7                   WORD.PTR @ 2+ DUP >R DUP
  8                   C@ + 1 - WORD.PTR ! ( update PFA pointer )
  9                   R> COUNT TYPE ;
 10
 11 : WORD.DSP        ( given CFA, display the glossary name )
 12                   3 - -1 TRAVERSE DUP 1+ C@ 59 =
 13                   IF 1 QUIT.FLAG ! ENDIF ( if ";" we are done )
 14                   DUP C@ 160 AND 128 =  ( make sure legal NFA )
 15                   IF ID. ELSE 1 QUIT.FLAG ! ENDIF ;        -->
```

```
Screen # 33
  0 ( fig-FORTH Decompiler )
  1
  2 : BRANCH.DSP      ( get branch offset, calculate the )
  3                   ( actual branch address, and display it )
  4                   ." to "
  5                   WORD.PTR @ 2+ DUP WORD.PTR ! ( update PFA ptr )
  6                   DUP @ +    ( offset + PFA = actual target addr )
  7                   0 HEX D.  DECIMAL ( print it )   ;
  8
  9 : USERV.DSP       ( display a user variable )
 10                   ." User variable, current value = "
 11                   WORD.PTR @ 2+  ( calculate PFA )
 12                   C@ 38 +ORIGIN @ +  ( then user area address )
 13                   @ N.   ( fetch and print contents )
 14                   1 QUIT.FLAG ! ( done, set flag )  ;
 15 -->


Screen # 34
  0 ( fig-FORTH Decompiler )
  1
  2 : VAR.DSP         ( display a variable )
  3                   ." Variable, current value = "
  4                   WORD.PTR @ 2+  ( calculate PFA )
  5                   @ N.     ( fetch and print contents )
  6                   1 QUIT.FLAG ! ( all done, set flag )  ;
  7
  8 : CONST.DSP       ( display a compiled constant )
  9                   ." Constant, value = "
 10                   WORD.PTR @ 2+ ( calculate PFA )
 11                   @ N.    ( fetch and print contents )
 12                   1 QUIT.FLAG ! ( all done, set flag )  ;
 13 -->
 14
 15


Screen # 35
  0 ( fig-FORTH,Decompiler )
  1 : DIS
  2   -FIND 0=            ( is input word in dictionary? )
  3   IF 3 SPACES ." ? not in glossary " CR      ( no, quit )
  4   ELSE DROP DUP DUP 2 -    ( yes, calculate CFA )
  5   @ =      ( if contents of CFA = PFA then its a primitive )
  6   IF ." <primitive> " CR      ( so print message and quit )
  7   ELSE     ( otherwise it's high level FORTH so decode it )
  8   0 QUIT.FLAG !       ( initialize done flag )
  9   2 - WORD.PTR !      ( initialize pseudocode pointer )
 10   CR CR               ( print some blank lines )
 11   BEGIN               ( now list the compiled pseudocode )
 12   WORD.PTR @ DUP      ( fetch current pseudocode pointer )
 13   0 HEX D.  SPACE DECIMAL ( print value of pointer )
 14   @   ,               ( fetch current pseudocode word )
 15 -->
```

*(Continued on next page)*

# FORTH Decompiler (Listing continued, text on page 48)

```
Screen # 36
  0 ( fig-FORTH Decompiler )
  1 CASE                ( now decode any special word types )
  2 LIT.ADR OF          ( compiled literal, print its value )
  3                     WORD.PTR @ 2+ DUP WORD.PTR ! @ N. ENDOF
  4 DOCOL.ADR OF        ( : points to the nesting routine )
  5                     ." : "     ENDOF     ( so just print a colon )
  6 OBRANCH.ADR OF      ( conditional branch with in-line offset )
  7                     ." Branch if zero "     BRANCH.DSP ENDOF
  8 BRANCH.ADR OF       ( unconditional branch with in-line offset )
  9                     ." Branch "     BRANCH.DSP ENDOF
 10 LOOP.ADR OF         ( end of a DO...LOOP structure )
 11                     ." Loop "     BRANCH.DSP ENDOF
 12 PLOOP.ADR OF        ( end of a DO...+LOOP structure )
 13                     ." +Loop "     BRANCH.DSP ENDOF
 14 -->
 15


Screen # 37
  0 ( fig-FORTH Decompiler )
  1 PDOTQ.ADR OF        ( display compiled text string )
  2                     ." Print text: "     PDOTQ.DSP ENDOF
  3 USERV.ADR OF        ( display a user variable )
  4                     USERV.DSP ENDOF
  5 VAR.ADR OF          ( display a global variable )
  6                     VAR.DSP ENDOF
  7 CONST.ADR OF        ( display a compiled constant )
  8                     CONST.DSP ENDOF
  9 PSCODE.ADR OF       ( display ;CODE and quit )
 10                     WORD.PTR @ @ WORD.DSP
 11                     1 QUIT.FLAG ! ENDOF
 12 -->
 13
 14
 15


Screen # 38
  0 ( fig-FORTH Decompiler )
  1
  2                          ( all special word types checked, )
  3 DUP WORD.DSP             ( if word did not match any cases )
  4                          ( just print its name )
  5 ENDCASE CR               ( done decoding word type )
  6 2 WORD.PTR +!            ( update pseudocode pointer )
  7 QUIT.FLAG @              ( check if finished flag set or if )
  8 ?TERMINAL OR             ( interruption from terminal )
  9 UNTIL                    ( otherwise display another word )
 10 ENDIF ENDIF CR ;         ( all done now )
 11
 12
 13
 14
 15
```

End Listing

## Sample Run

```
       OK
30 LOAD OK
DIS XXX    ? not in glossary

OK
DIS DUP <primitive>

OK
DIS C/L

704  Constant, value = 64   ( 40 H )

OK
DIS BASE

800  User variable, current value = 10   ( A H )

OK
DIS WORD.PTR

1F89  Variable, current value = 8073   ( 1F89 H )

OK
DIS TRIAD

1DD9  :
1DDB  12   ( C H )
1DDF  EMIT
1DE1  3   ( 3 H )
1DE5  /
1DE7  3   ( 3 H )
1DEB  *
1DED  3   ( 3 H )
1DF1  OVER
1DF3  +
1DF5  SWAP
1DF7  (DO)
1DF9  CR
1DFB  I
1DFD  LIST
1DFF  ?TERMINAL
1E01  Branch if zero to 1E07
1E05  LEAVE
1E07  Loop to 1DF9
1E0B  CR
1E0D  15   ( F H )
1E11  MESSAGE
1E13  CR
1E15  ;S

OK
```

End

# OF INTEREST

by Dave Cortesi

## Two Products Step FORTH

We found two FORTH announcements to open the column in this special issue. **MicroMotion** have produced what they claim to be "the first FORTH for the Apple that meets the international standards established by the FORTH standards team in October 80." The package is said to include a "professionally written" manual. That might mean that they hired a technical writer to do the manual, a wise move in the light of some of the comments we have heard about FORTH manuals. FORTH-79 can handle disks with either 13 or 16 sectors, although its files are not compatible with those of APPLE DOS. The price is $90.

**Timin Engineering** have enhanced their FIG-FORTH implementation for CP/M with a floating point arithmetic feature. The feature extends a standard Timin FORTH so that it can handle arithmetic on, and comparison of, single precision (6+ decimal digits) floating point numbers. The feature costs $100.

Note: Just before press time we received a call informing us that **FORTH, Inc.** has been acquired and/or merged with **Technology Industries, Inc.** Said caller stated the staff will remain the same, but T.I. resources will be boosting more language development – possibly looking for the El Dorado of "programmer-less programming"?

## North Star, Meet CP/M

Is North Star BASIC superior to the usual CP/M ones? **Allen Ashley** thinks so, but then he makes COMSTAR, a compiler for North Star BASIC. He wrote to tell us that owners of COMSTAR may buy a CP/M run-time library for the compiler so that their programs can run under CP/M. The feature costs $75, and includes a North Star-to-CP/M transfer utility.

Those who'll continue to use their Microsoft compilers might like to know that **Westico** is selling what they think is a greatly improved linking loader for Microsoft relocatable files. The main features claimed for LYNX are its friendly commands and its ability to construct overlay segments. We remember overlay segments; if you don't know what an overlay is, you're better off.

## Getting Serious About Atari

The PCC staff is trying to get serious about the house Atari 800. The Star Raiders cartridge has been locked in the publisher's desk and the typesetter is writing a dull old invoice program. **Optimized Systems Software** is another serious group; they're dedicated, they say, to promoting "total access" to the 800's abilities. To that end they've created an enhanced BASIC and a better disk operating system. The BASIC contains Atari BASIC as a subset but adds more powerful graphics commands, PRINT USING, and better string handling. The operating system, OS/A+, contains disk utilities and an assembler; it supports batched commands as well as interactive ones. Each package costs $80.

**Mosaic Electronics** has a serious suggestion for those who'd like to use either Atari machine for hardware experiments. Their $18 I/O Package consists of connectors and ribbon cables with which the experimenter can lead the four game jacks on the front of the console out to a breadboard where they effectively form a parallel port. Anything at all can happen from there on out; the documentation is said to contain a number of examples.

## Minority Computerists, Unite!

An Atari isn't really meant to be a hardware hacker's machine, but the Heath ET-3400 was designed for exactly that. Now, at last, there is an **ET-3400 User's Group**. We received the Group's first newsletter. It consists mostly of reprints, but the long-neglected owners of this fine little machine could turn that around by joining. There's no set membership fee; send a reasonable donation.

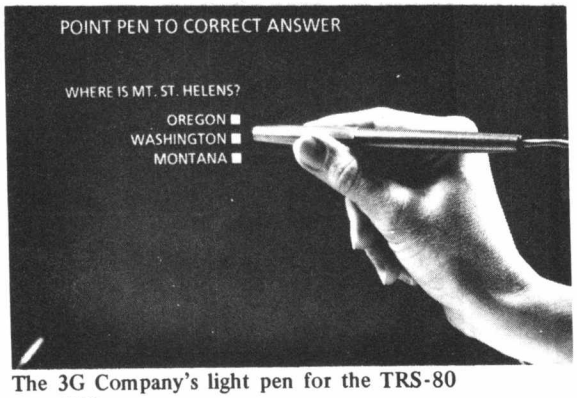
Another set of owners who must feel a bit neglected are those who bought Processor Technology SOL machines. **Proteus** is a group of SOL owners working to keep their machines alive and supplied with software. The club now has access to the assembly source code for all the Processor Technology systems software. Source code, object code, and reprinted manuals are available, as well as 60 diskettes of public domain software.

## An Abundance of Information

Ever feel as if there's just too much information flooding in for you to handle? Well, you aren't alone; anyone who tries to keep up with the computer field is drinking from a firehose. Here are a few new compendia that might help organize your knowledge.

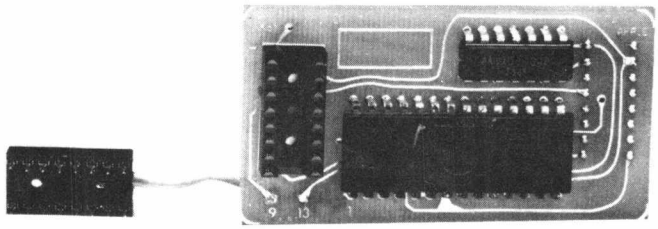
**Elliam Associates** have indexed a couple of hundred articles about CP/M on diskette, to be retrieved with the IRS retrieval system from Island Cybernetics. The articles were published in 1979 and 1980; the index fills two disks. The index alone costs $35; IRS is another $38.

Rather than indexing articles, **Imprint Software** indexes programs. Their $29 catalog lists "thousands" of programs, organizing them three different ways: by subject, by required system, and by publisher. They claim to have listings for all major computer systems, and for programs for all countries, not just the USA.




The 3G Company's light pen for the TRS-80 costs $40.

The Vista Type-Ahead buffer lets you get 40 keystrokes ahead of your Apple.



**INMAC** is a retailer of computer supplies and support equipment. They're the first such, they claim, to publish a catalog specifically for personal machines. Their *Microcomputer User's Idea Book* is organized by machine type and lists diskettes, ribbons, cables, furniture, and accessories.

We received the *Robotics Industry Directory* prepared by **Robotics Age** magazine. It costs $25; it contains 60-odd pages of spec sheets for industrial robots with profiles of their makers, and a directory of robotics consultants.

## Lowering the Price of Words

The flood of words rolls on, and somehow it all has to be set in type. Jim Warren published a diatribe against the makers of typesetting equipment in the *Silicon Gulch Gazette* distributed at the Faire. Shape up, he warned them, or the micros will take over your front-end sales. **Type Share** was already ahead of him. They've built a computerized typesetting system, based on the Commodore VIC, that they're ready to sell for "less than $500." The user of the system prepares the copy to be typeset on the VIC and records it on cassette tape. The cassette is sent to a Type Share center to set on a phototypesetter; galleys are returned to the user, ready for paste-up, at "a fraction" of the conventional price. The company also sell a more elaborate system based on the Commodore 8002; it can deliver its copy on diskette or over a phone line.

Many micro users prepare documents with a daisy-wheel printer. The cost of ribbons is the heaviest operating expense of these printers, and the best print quality comes from the expensive film ribbons. **Torres Ribbon Service** aims to cut the cost for you; they reload film ribbon cartridges for almost any daisy printer. They will refill your cartridges for $2.50 each, well under half the price of a new ribbon. It was only at the recent Computer Faire that we learned that it was possible to reload a ribbon cartridge; we tried the services of **American Ink Products** with good results. We suspect most major cities harbor such a service.

## Lower the Price of Hardware

**Wintek** wrote to tell us that they have cut 30% from the price of all items in their line of 6800-based modules. If you buy 100 of them, you can have a single-board computer for $82.

Several other bits of useful, inexpensive hardware emerged from this month's heap. The neatest one is the Apple Type-Ahead Buffer from **Vista Computer Co.** It's an elegant little circuit board that buffers up to 40 keystrokes. With it installed – and installation requires only a phillips screwdriver – you can type the next command while your Apple's still working on the last one. Anyone with more than a few hours of computer use will appreciate the pleasure of that.

Light pens aren't new, but the **3G Company** has what looks like a nice one for the TRS-80. It comes with a demonstration cassette and a "complete programming booklet" for $40.

Radio Shack notified us that they now carry part number 271-1705, a complete joystick assembly, for just $5. The stick swings in a 30-degree arc in either dimension to control two, 100K ohm linear pots. The applications to which really cheap joysticks could be put are endless.

## 3M Cartridge Control for CP/M

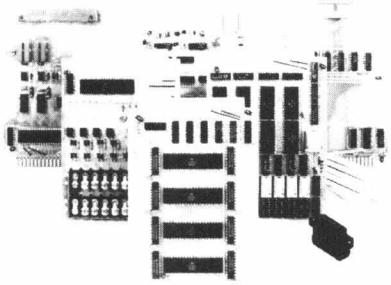
You may have seen the handsome little 16-megabyte tape cartridges; we first met them in the IBM 5110 desktop machine and they are often used now for hard disk backup. **MCS** has put together a comprehensive software and hardware package to control a cartridge drive under CP/M. It will keep track of 21,000 files per cartridge by name, with date and time of recording, and will move files between cartridge and disk. If you already have a cartridge drive, the software costs $300. Adding an S-100 controller and a drive brings the price to $5000.

## IBM Dips Toe in Market

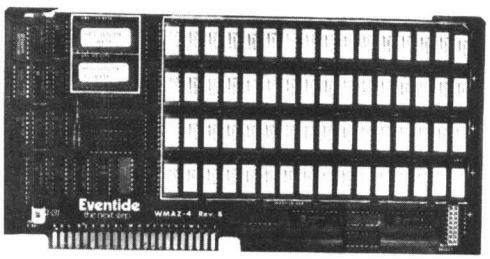
There has been a wave of speculation about IBM's expected entry into the desktop computer market. Just as this issue went to press we received an announcement from the IBM General Systems Division, describing in general terms the IBM System 23 Datamaster, "a low-cost, desktop computer that can combine word processing with data processing in a single system designed for first-time users."

The only surprise in the announcement was that "Datamaster can include two computer workstations which permit two operators to use it simultaneously." We couldn't figure out what that might mean: multiprogramming? Or dual processors? So we called our local IBM office. "The Datamaster has an optional second processor; the two processors can share files," an IBM rep told us. Is it available now? "We have stock in the office, ready to go," he said.

Datamaster can have up to 128KB of storage (split 64-64 between two CPUs, we assume) and 1-4 diskette drives. A set of business accounting programs, written in BASIC, is available now; word process-

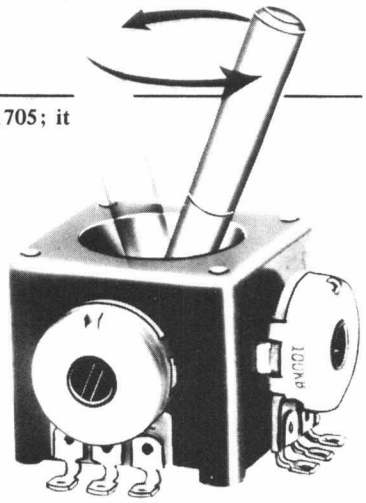


**The whole Wintek line of micro modules has been reduced in price 30%.**




**This is what half a megabyte of RAM looks like; it costs $12/kilobyte.**

**Radio Shack's joystick is part 271-1705; it costs just $5.**



ing software will be available in September. And the price? Well, the price is genuine IBM: "A complete system is just under 10 thousand," we were told. "If you can do without a printer you can get it into the 7 thousand range." That rushing sound you hear is an entire industry sighing in relief. ▶▶J

## Contact Points

**3G Company**, Rt. 3 Box 28a, Gaston, OR 97119; (503) 662-4492.

**Allen Ashley**, 395 Sierra Madre Villa, Pasadena, CA 91107; (213) 793-5748.

**American Ink Products**, 527 Howard Street, San Francisco, CA 94105.

**Elliam Associates**, 24000 Bessemer St., Woodland Hills, CA 91367; (213) 348-4278.

**ET-3400 User's Group**, c/o Charles Van Dyke, 11231 Oak St., El Monte, CA 91731.

**Eventide Clockworks**, 265 West 54th Street, New York, NY 10019; (212) 581-9290.

**Imprint Software** USA: 420 South Howes, Ft. Collins, CO 80521; (303) 482-5574. UK: 16 Milton Ave., Highgate, London N6; 01-348-3998.

**INMAC**, 2465 Augustine Drive, POB 4780, Santa Clara, CA 95051; (408) 727-1970.

**MCS** 8308 Juniper, Ft. Worth, TX 76180; (817) 498-6390.

**MicroMotion**, 12077 Wilshire Boulevard #506, Los Angeles, CA 90025; (213) 821-4340.

**Mosaic Electronics**, POB 748, Oregon City, OR 97045.

**Optimized Systems Software**, 10379 Landsdale Ave., Cupertino, CA 95014; (408) 446-3099.

**Proteus**, 1690 Woodside Road, Suite 219, Redwood City, CA 94061.

**Robotics Age**, POB 725, La Canada, CA 91011; (213) 352-7937.

**SIG/M**, POB 97, Iselin, NJ 08830.

**Timin Engineering**, 9575 Genesee Avenue, Suite E-2, San Diego, CA 92121; (714) 455-9008.

**Torres Ribbon Service**, 11154 South Mt. Vernon Avenue, Colton, CA 92324; (714) 792-0831.

**Type Share**, 8315 Firestone Blvd., Downey, CA 90241; (213) 923-9361.

**Vista Computer Co.**, 1317 E. Edinger Avenue, Santa Ana, CA 92705; (714) 953-0523.

**Westico**, 25 Van Zant, Norwalk, CT 06855; (203) 853-6880.

**Wintek**, 1801 South Street, Lafayette, IN 47904; (317) 742-8428.

# A FORTH Data Base Design

*(continued from page 47)*

```
This demonstration data system provides a pattern for the
further development of any type of data base.  The basic
file formating definitions are on Screens 22 and 23. Some
utilities are on 24, 25, and 26.  The demo file
definitions are on 27. Elementary file manipulation
utilities are on 28.  This model should get you started.

ENTER 'Y' TO LOAD SCREENS
SCREEN 22: FILE DEVELOPMENT 2@ MSG # 4
SCREEN 23: FILE DEVELOPMENT - 2
SCREEN 24: DOUBLE PRECISION ARITHMETIC
SCREEN 25: DATE COMPRESSION AND EXPANSION
SCREEN 26: ?$AMOUNT AND .$AMOUNT
SCREEN 27: DEMO FILE - RECORD GENERATION
SCREEN 28: DEMO FILE - CLEAR.DATA, INPUT, OUTPUT OK
 OK
 OK
 OK

DEMO OK
CLEAR.DATA OK
OUTPUT

EMPTY FILE

OK
INPUT

ENTER NAME                  --> ZENITH
ENTER DATE     -->  ( MM/DD/YY ) 4/21/81
ENTER AMOUNT                --> 18.50
  1  ZENITH                       04/21/81  $   18.50OK
 OK
INPUT

ENTER NAME                  --> IBM
ENTER DATE     -->  ( MM/DD/YY ) 4/21/81
ENTER AMOUNT                --> 60.
  2  IBM                          04/21/81  $   60.00OK
 OK
INPUT

ENTER NAME                  --> DEC
ENTER DATE     -->  ( MM/DD/YY ) 4/21/81
ENTER AMOUNT                --> 103.5
  3  DEC                          04/21/81  $  103.50OK
 OK
 OK
0 READ ADDRESS HEX 80 DUMP DECIMAL
  58CA    3  0  0  0  0  0  0  0   0  0  0  0  0  0  0  0  ................
  58DA    0  0  0  0  0  0  0  0   0  0  0  0  0  0  0  0  ................
  58EA    0  0  0  0  0  0  0  0  5A 45 4E 49 54 48 20 20  ........ZENITH
  58FA   20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
  590A   20 20 20 20 20 20 B5 84   0  0 3A  7  0  0 49 42        ....:...IB
  591A   4D 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  M
  592A   20 20 20 20 20 20 20 20  20 20 20 20 B5 84  0  0              ....
  593A   70 17  0  0  0  0  0  0   0  0  0  0  0  0  0  0  p...............
OK
 OK
 OK
OUTPUT
  1  ZENITH                       04/21/81  $   18.50
  2  IBM                          04/21/81  $   60.00
  3  DEC                          04/21/81  $  103.50

OK
 OK

                                        OK
 OK
 OK
: STATEMENT CR CR 20 SPACES ." STATEMENT " CR CR OUTPUT
     CR CR ." TOTAL VALUE "  33 SPACES  0 0 0 READ TAG @ 1+ 1
     DO I READ DOLLAR D@ D+ LOOP .$AMOUNT CR CR CR ; OK
 OK
 OK
STATEMENT

                  STATEMENT


  1  ZENITH                       04/21/81  $   18.50
  2  IBM                          04/21/81  $   60.00
  3  DEC                          04/21/81  $  103.50



TOTAL VALUE                             $  182.00


OK
```

**End Listing**

# The FORTH Alternative

by Mitchell E. Timin

There is a new way of using a computer that is still unfamiliar to most computer people. It is known as FORTH. Although many have heard of FORTH, relatively few understand just what it is. The purpose of this article is to describe FORTH.

## A Computer Utilization System

FORTH is a *computer utilization system*, not just a language. FORTH is a language, but it also includes the capabilities of an operating system, a text editor, an assembler, and a monitor. It is a complete system for making use of a computer. With FORTH all of these facilities are memory resident, and hence available for immediate use. This is in contrast to conventional systems, where the editor, language compiler, etc. are separate utilities needing to be loaded into memory from disk each time they are used.

The customary way of using a computer is by means of an operating system and various program files. The operating system maintains a dialogue with the user, servicing the user's requrests. When the execution of a program is requested, then this dialogue is suspended until the program execution is finished. With FORTH there is no separation of the software into operating system and utilities. The operating system commands are replaced by a complete language which eliminates the need for separate programming languages. The FORTH language is therefore an operating system and a programming language at one and the same time.

BASIC is another language that is often implemented in an interactive manner; this is one of the reasons for its popularity. This form of BASIC also has a memory-resident editor and can edit and run without loading from disk. Unfortunately, interactive BASIC has many shortcomings which cause considerable difficulty when it is used for medium size and larger projects. Chief among these is its lack of a satisfactory subroutining mechanism and severe restrictions on the naming of variables. Furthermore, BASIC is unsatisfactory for real-time systems programming due to its slow execution speed and lack of bit and byte manipulation primitives. In contrast, FORTH gives the user immediate access to all capabilities of his computer.

*Dr. Mitchell E. Timin, c/o Timin Engineering Co., 9575 Genesee Rd., Suite E2, San Diego, CA 92121.*

## FORTH Words

FORTH consists primarily of a dictionary of *words*. These FORTH words are conceptually similar to English language words in that they each have a certain spelling, a definition, and rules for usage. An example of a FORTH word is TYPE, which, when given a starting adress and a count, will find some characters in memory and display them on the console. A second example is MAX, which will select the largest of a group of numbers. The common numerical operators, + - * and /, are FORTH words whose "spelling" is just one character. Several of the most commonly used FORTH words are also spelled with one character.

FORTH words may be used as direct commands to the computer from the operator. One or more FORTH words, when typed at the console, will cause their particular actions to occur. For example, typing VLIST (a FORTH word) followed by the return key, will result in the display of all of the FORTH words, since that is the function of VLIST. FORTH comes with a vocabulary of over one hundred words that are likely to be used by the user. This does not count the assembler vocabulary nor many internal system words. The total vocabulary is usually several hundred words.

## Programming in FORTH

The FORTH user is not limited to the built-in FORTH words! Programming in FORTH consists of *defining new words* in terms of existing words. In this manner the vocabulary of available FORTH words can grow indefinitely. FORTH is very similar in this respect to a natural language. English is continually adding new words; each of them is defined in terms of the old ones.

Thus we see that FORTH words function both as programming elements and as direct commands to the computer. A FORTH word is conceptually similar to the subroutine or subprogram in other languages, however the usage rules and internal implementation are unique to FORTH.

A FORTH user creating new definitions may use the EDITOR commands to write these onto disk, thus creating a library of application software. Note that EDITOR commands are FORTH words. The EDITOR is not a separate utility, it is just a group of FORTH words, all of which are available at any time. The word LOAD is used to add these newly created definitions to the dictionary, in memory. The user also has the option of entering new definitions directly into the dictionary, with no disk record. This is done when experimenting or for one time use of a definition.

After the dictionary has been enlarged, by either of the above methods, the user may create another version of FORTH which includes this enlarged dictionary. Any number of special purpose versions may thus be created, if desired. These "customized" FORTH versions may restrict the user to a limited vocabulary, if desired. This permits the use of the special application software, while denying direct use of other parts of the FORTH system.

## The Stack

Most FORTH words require some data to operate on; most also produce some data that may in turn be used by another FORTH word. The primary method used by FORTH to pass data to and from FORTH words is the *data stack*. The data stack is a region in memory where the user can easily store and retrieve data. For instance, if the user types 1234 at his terminal, followed by carriage return, the number 1234 will be put on the data stack. In general, FORTH words take whatever data they need from the top of the stack, and leave whatever results they produce on the stack. Named variables, arrays, direct memory addressing and CPU registers are available as alternate data handling mechanisms.

Using the stack in this manner leads to the requirement that operands precede operators when writing FORTH. To illustrate, if it is desired to add 45 and 54, and then print the result, type:

```
45 54 + .
```

When the user hits RETURN after the above line the following sequence of events will occur:

- the number 45 is put on the data stack.
- the number 54 is put on the data stack.
- the FORTH word "+" begins to execute, doing the following:

- the number 54 is removed from the stack.
- the number 45 is removed from the stack.
- these two numbers are added, producing the sum, 99.

- the number 99 is put on the stack..

– the FORTH word "." begins to execute, doing the following:
  - the number 99 is removed from the stack.
  - it is tranlated from binary form to a sequence of ASCII codes.
  - these are sent to the terminal, causing 99 to appear.
– control returns to the FORTH executive which types OK.

The practice of entering operands prior to operators is also used on the highly successful Hewlett-Packard calculators. (Mathematicians call this reverse Polish notation.) The H-P calculators also use a data stack; reverse Polish notation is the natural way to communicate with a stack machine. Use of a data stack is the most efficient way of passing data between routines, which is the reason for its use in FORTH.

## Defining Words

Programming in FORTH makes use of special FORTH words called *defining words*. Here is an example; we will define a new FORTH word called ADD3 which will compute the sum of three numbers from the stack and display the result:

```
: ADD3 + + . ;
```

The defining word is ":". Its use causes ADD3 to be created and compiled into the dictionary. ";" ends the definition. Later, if ADD3 is used it will execute + + . which will add the top three stack items and display the result. The first will replace the top two stack items with their sum. The second + will add that sum to the last stack item, leaving only the sum of the original three. Finally the . prints the result.

The only other frequently used defining words are VARIABLE and CONSTANT. The defining word CODE is used with the Assembler. The advanced FORTH programmer may easily create new defining words, but this is unnecessary for the vast majority of applications.

## Hardware Access

The FORTH user has complete access to all I/O ports and memory locations. Data can be read from or written to any device or memory location, either directly by the user's command, or during the execution of FORTH routines written by the user. This makes FORTH ideal for control or testing of hardware. FORTH can use any number base also, so binary, hex, or octal may be used when appropriate. For example, suppose bit 0 of output port A0 controls a motor. To turn on a motor the user types:

```
HEX 1 A0 P!
```

The word HEX is used to switch to hexadecimal for terminal I/O. It need only be used once, until another number base is desired. P! is the word which sends data to a port, in this case the port is A0 and the data is 1.

Now suppose we wish to have the computer turn the motor on and off at periodic intervals, so that we may test its temperature rise. We will define the word MOTOR-ON-OFF to do this, as follows:

```
: MOTOR-ON-OFF BEGIN
1 A0 P! WAIT  0 A0 P! WAIT
?TERMINAL UNTIL ;
```

Here, MOTOR-ON-OFF is the new FORTH word which we are creating. BEGIN and ?TERMINAL UNTIL create a loop which repeats indefinitely until a key is hit on the console. WAIT is a word which we must have defined previously; it is a software delay of an appropriate interval. Now, whenever we type MOTOR-ON-OFF and return, the motor will begin to turn on and off until a key is hit at the terminal. The definition of WAIT may be as simple as:

```
: WAIT 7FFF 0 DO LOOP ;
```

which causes a brief delay by repeating an empty loop 32,000 times. (7FFF hex = 32,767 decimal)

## Using Disk Storage

FORTH makes use of disk storage by considering the disk to consist of *blocks*. Each disk block has 1024 bytes. The blocks are referred to by number, starting with number 1. Disk blocks are used when FORTH software is created using the editor. They are also used for any data storage operation. Individual blocks may always be accessed by number, but the programmer may create a naming system to suit specific needs.

An important feature of FORTH disk utilization is *virtual memory*. By this term we mean that images of some disk blocks may be in memory at any time, and that these are available without disk access. The number of blocks which may be in memory simultaneously depends on the implementation, but eight is a typical value. Typically, these will be the eight most recently referenced blocks. Any of these may be read or written to without the delay of disk I/O. When another block is called for it will automatically replace the oldest of those in memory. That one, if it has been modified, will be updated by writing onto the disk the copy in memory.

## Assembly Language Modules

The FORTH Assembler is used to create a FORTH word that consists of executable machine code. The more common FORTH words, as in the examples above, are compiled into lists of addresses of other FORTH words. Use of the Assembler, and of the defining word CODE, allows writing instructions in assembly language which are then assembled into machine code within the dictionary. This is done when maximum execution speed is needed. FORTH written without the assembler does not execute as fast as assembly code, nor as fast as fully compiled languages such as FORTRAN. FORTH programmers usually complete an application program without using the assembler. Then, if there is a speed problem, one or more "bottleneck" words will be redefined using the assembler. In most instances 90% of the execution time is used by only 10% of the code, so most code may remain in high level FORTH.

## Structured Programming

The concepts of *top down and structured programming*, which have gained wide acceptance in recent years, are fully realizable with FORTH. FORTH programming is structured programming. FORTH does not use a "GO TO" statement; instead there is a small group of FORTH words that implement control structures. IF . . . ELSE . . . ENDIF is the two-path test and branch, while CASE: implements a multi-path test and branch. DO . . . LOOP, BEGIN . . . UNTIL, and BEGIN . . . WHILE . . . REPEAT implement looping structures. All FORTH control structures have one entry path and one exit path.

The top down approach to program creation is facilitated because each FORTH word is a clearly defined module which is constructed from other FORTH words. With FORTH there is no limit to

the depth of this nesting. Hence the programmer begins his efforts by choosing one name to represent the final result which he plans to achieve. Next he would analyze the task and break it down into a small number of sub-tasks; for each of these he chooses the name of a FORTH word. He proceeds in this manner until reaching the level of the original FORTH vocabulary. The input and output data for each word must be specified as part of this design process. For example, a programmer creating an automatic testing program might name the program TEST. This would become a FORTH word which performs a complex sequence of operations. He then might conclude that the basic components of the task are to initialize the hardware, exercise the hardware, and deactivate the hardware. He then chooses the names INIT, EXER, and OFF for those modules. The EXER module is then broken down into TESTA, TESTB, TESTC, and so on.

After the initial design procedure is done, coding and testing of modules proceeds from the bottom up. The top down design procedure results in specifications for the many FORTH words which will build the final result. Each of these is created using the appropriate defining word (usually ":") and is tested immediately upon creation. Testing a FORTH word consists of typing a command line which puts appropriate data on the stack and invokes the new word. If necessary, a loop or other structure can be used to invoke the new FORTH word many times under varying conditions. It is also a common practice to define a dummy version of a middle-level word so that the top-level definitions may be tested prior to completion of the bottom-level words.

## Advantages of FORTH

There are two important benefits resulting from the use of FORTH instead of other computer languages:

1) Total software development time is greatly reduced.
2) Memory requirements are greatly reduced.

This has been demonstrated by the experience of those who have used FORTH for major projects. The reduction in programming time is due partly to the extreme modularity of FORTH. On a large project the programmers accumulate a special vocabulary of FORTH words which are very appropriate to their application. These very-high-level routines are just as easy to use as the simplest FORTH primitives. FORTH projects do not experience the bogged-down situation, which is common on large projects using other languages. This is directly due to the fact that the very-high-level FORTH construct becomes a primitive as soon as it has been defined. Hence, the primitives of the language become more and more powerful as the project progresses. The second reason for the high production rate of FORTH is its interactive nature. With conventional programming systems the user experiences frequent waiting while the editor, compiler, linker, and application load and execute. This is especially true on floppy disk systems, but is also true on multitasking minis when there are many users. With FORTH all of this is memory resident so these waits do not occur.

The unique internal structure of FORTH produces remarkably compact code; our version of FORTH is about 11 kilobytes. These 11K bytes include the language compiler and interpreter, operating system, editor, assembler, and debugger. Application software developed by the user may either be disk resident, loaded by FORTH as needed, or it may be added to the 11K package and become memory resident. Also, if memory is limited, the application software may replace the editor, assembler and other parts of FORTH. In this manner a custom FORTH implementation may be created occupying as little as 8K bytes. (The above byte counts refer to code space. Full Timin FORTH also requires 8K bytes for disk buffers and another K for stack space and the BIOS portion of CP/M. The custom application may or may not need this.)

## A Little History

FORTH was invented by Charles Moore in 1969. He was then with the National Radio Astronomy Laboratory, which resulted in FORTH coming into wide use within the astronomical community for observatory automation.

In 1978 a group of experienced FORTH programmers formed the FORTH Interest Group (FIG) in order to make the language more widely available. There is also a European FORTH user's group.

There is now a FORTH standard known as FORTH-79. This was created by the FORTH Standards Team on which FIG and the Europeans are well represented. Although an exact implementation of FORTH-79 is not yet available, FIG FORTH is very close. **DDJ**

### FORTH by Timin Engineering

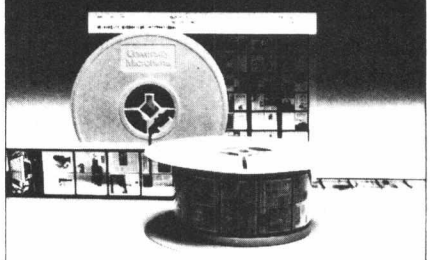
*Our version of FORTH is based upon FIG FORTH. Timin Engineering Co. has added the interleaved disk format, the Z-80/8080 assembler vocabulary, and several FORTH words including SAVE, DUMP and CASE:. The Editor is based upon the FIG model, but a string search module was implemented using assembly language for quick response.*

*In order to achieve the fastest possible disk I/O, the physical sectors that make up a FORTH disk block are always on the same track. They are distributed on the track so that all can be read in one revolution of the disk.*

*The FORTH Assembler supplied with Timin FORTH will assemble all 8080 or Z-80 machine instructions. The mnemonic system used follows INTEL's 8080 assembly language, extended to include the special Z-80 instructions.*

*Timin FORTH will run on any 8080/8085/Z-80 system with CP/M and at least 24K of memory. It is available on most diskette formats.*


This publication is available in microform.



University Microfilms International

300 North Zeeb Road
Dept. P.R.
Ann Arbor, Mi. 48106
U.S.A.

30-32 Mortimer Street
Dept. P.R.
London W1N 7RA
England

# DR. DOBB'S CLINIC

by Dave Cortesi, Resident Intern

## DUP DUP NEG AND =

That's one solution, expressed in FORTH by Mitchell Stanek and Dean Vieau of Detroit, MI, to the "Lost Chord" problem we presented in June. The problem was to re-invent a simple test that would tell if a binary integer contained only a single 1-bit. The NEG (two's complement) word isn't in fig-FORTH, but it's easily created.

Readers provided two different solutions. The test used by Stanek and Vieau can be expressed more conventionally (but less succinctly) as:

( N and (-N) ) = N

Thomas E. Carr explained it this way: "The only bit that is set in both a number and its two's complement is the least significant one. Therefore, if a number is ANDed with its complement, the result is only the least significant bit of the original number. If that bit also happens to be the only bit set in the original, then the two numbers (the original and the result of the AND) are equal. Voila!" Both this test and the next one will pass a quantity of N=0; all respondents noted that that case must be filtered out first.

Dan Bates of Johnson City, TN commented on how the test on a 16-bit integer could be folded into an 8-bit register. He wrote, "If X=2**n then the high byte of X, XHI, and the low byte of X, XLO, cannot be equal. This test also eliminates X=0. If X<>2**n, and XHI<>XLO, then the result of XHI OR XLO will have 2 bits set. If X=2**n then XHI OR XLO will have one bit set. So to turn the 16-bit problem into 8 bits:

If XHI=XLO Then false, return.
Let T := XHI OR XLO
If (T AND -T)=T, Then true, return
Else false, return

A larger number of readers favored a test that can be expressed this way:

( N and (N-1) ) = 0

Georges A. Bécus of the University of Cincinatti said of this test: "It is based on Peter Wegner's method for counting ones in a binary number (*Comm. ACM*, May 1960, p. 322; see also Lewis and Doerr, *Minicomputers: Structure and Programming*, Hayden, 1976, pp. 156-157). The method consists of subtracting one from the binary number N and ANDing the result with N. The process is repeated until a zero result is obtained. The number of times the process is repeated is the number of ones in N. Since we are trying to determine if the number N has a single one we need only go through the process once. If the result is zero then N has a single one."

Most writers presented their tests in pseudo-code or in C, providing assembly language routines only after the algorithm had been justified at a higher level. That's a healthy attitude, and we'll avoid asking for assembly language solutions in the future. The elements of a problem might be bits or binary integers, but C, Pascal, even BASIC can be used to manipulate them.

## Shifty Coding

There are still things to be learned about bit manipulation and binary arithmetic. Eveyone knows that it's good to implement multiplication and division by a power of 2 as a series of shifts, right? That was the implicit point behind "The Lost Chord." In his response, Norman Crowfoot of Flagstaff, AZ, pointed out that it isn't always a good idea. "One thing that many people don't know is the fact that a right shift of an odd negative number is *not* dividing by two. If you work out the math with the bits you find that -5 shifted right one place, with sign propogation, is -3. The fix is that whenever you shift a bit out the right of a negative number, one must be added to the value." Multiplication of a two's complement negative number by shifting it left isn't reliable, either. Perhaps someone will analyze these things?

## The Switch

William A. McWorter of Columbus, OH, is our kind of reader: he sent us two "sponses" just when we needed them. Most everyone knows the old trick for swapping two binary values A and B without the use of a temporary variable:

A := A xor B
B := A xor B
A := A xor B

Reader McWorter writes, "First 'sponse' for the Clinic. What if xor is not feasible? Are there other ways? I submit

A := A - B
B := A + B
A := B - A "

The trick works, and it's fascinating. It may be a little easier to follow if you recast it to eliminate the mid-stream assignments, thus:

$B' := (A - B) + B$
$A' := ((A - B) + B) - (A - B)$

By one of those eerie little coincidences that make magazine work interesting, it was just 24 hours earlier that we had received a note from Michael Driscoll of Sterling, VA, in which he claimed that this sequence would swap two variables:

A := A + B
B := A - B
A := A - B

That can be rewritten in an assignment-free way:

$B' := (A + B) - B$
$A' := (A + B) - ((A + B) - B)$

Both methods clearly work. What's more, while the xor-trick works only for binary values, these arithmetic tricks should work with numbers stored in any base. Are they safe with floating point values? Or rather, when are they dangerous?

## Separate But Equal

Reader McWorter's letter continues, " 'Sponse' two is a computing problem that made me think hard. How do you implement coroutines? I offer the 8080 code in Listing I.

We managed to dredge up a memory of what coroutines are: independent modules that, from one point of view, call each other as subroutines. From another point of view coroutines are equal programs that exchange control between themselves in an orderly way (as opposed to unrelated programs time-sliced by a higher mechanism). Algorithms for some problems fall out very neatly when organized as coroutines. With further thought, we worked out what the LINK

**Listing I.**

```
LINK:   LXI   H,0          ; Get caller's
        DAD   SP           ; . .stack pointer
        XCHG               ; . .and save it.
        LHLD  SAVEDSP      ; Swap coroutine's
        SPHL               ; . .stack pointer
        XCHG               ; . .with that of
        SHLD  SAVEDSP      ; . .the caller.
        RET                ; Resume coroutine
```

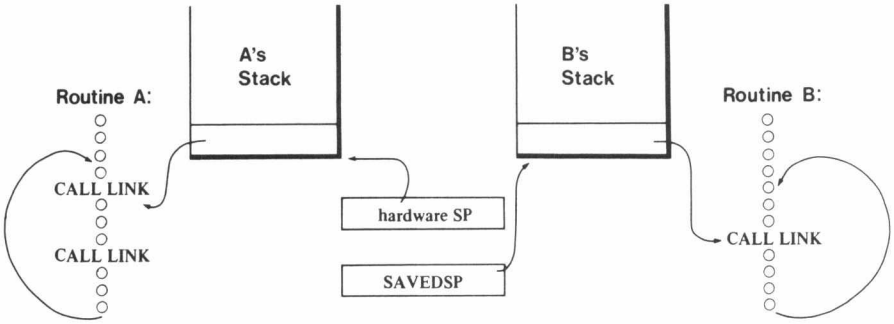



**Figure 1. Illustrating the operation of co-routines: the independent routines hand-off control between them through the logic of LINK (see text).**

fragment accomplishes. It's easier to express as a picture (figure 1). Routines A and B have their own stacks; they may or may not have variables in common. The diagram shows the state of the program just after Routine A has called LINK.

McWorter goes on, "[Coroutines] simplified the writing of a printer driver for my digital plotter. LINK connects only two coroutines. How can the linkage be written for three coroutines interacting in all possible ways? Can versions of BASIC or Pascal support coroutine structures or is SIMULA the only lucky one?"

## Submitting to CP/M

Working with CP/M is *such* an adventure. One just never knows what sort of undocumented bonus feature will turn up around the next corner. We stumbled onto the fact that the CCP treated colons as delimiters (May *Clinic*). Then John Pierce of Digital Research told us that the semicolon was the proper delimiter (July).

Now we've received a letter from Steve Pritchard of Willowdale, Ontario, Canada. He says "I too had stumbled into the use of semicolons to comment out statements. I use them to kill statements in the middle of a submit file. For example if we have a SUB file called X:

```
LIB OBJ=OBJ<$1> ;LIB combines
                                RELs
$2 ERA $1.REL       ;optional erase
```

Thus 'SUBMIT X ABC' will add ABC.REL to OBJ.REL and then erase ABC.REL, while 'SUBMIT X ABC ;' will add ABC.REL to OBJ.REL only. As you can see, this gives a very primitive statement selection capability, but since it is unadvertised we cannot complain. [*Yes we can! DEC*] I was also toying with the idea of writing an IF processor which would not be too hard to implement."

Pritchard has found an effect we would never have anticipated. After all, if the semicolon is a delimiter, why wouldn't it delimit the parameter list of a SUBMIT command? We experimented. SUBMIT treates a semicolon just like any other character; one can appear at any point in the parameter list. Parameters following a semicolon are still visible and are handled normally. So a semicolon is a delimiter only to a program that wants to treat it as one; the CCP does, and SUBMIT doesn't.

## Delayed Again

We have several nice division and decimal-output routines to display, but the listings are fairly bulky. In order to do them justice, we will devote the next Clinic to that problem. In the following (November) Clinic we will turn to the many sponses and responses that will have arrived in the meantime. Won't they?

DDJ

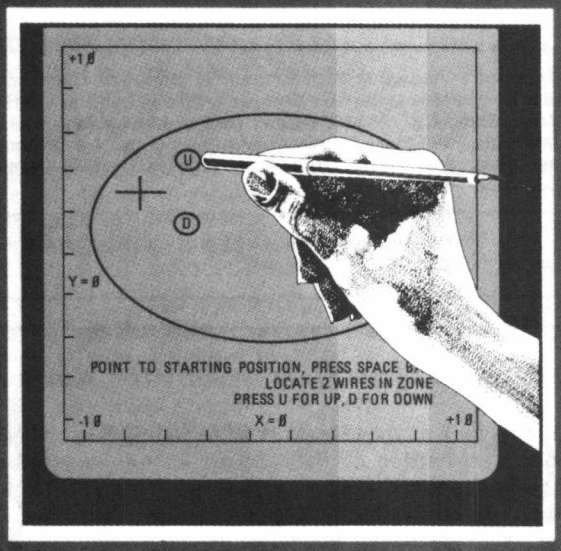
Our high-quality editorial
and
your high-quality products
are made for each other!
Call (415) 323-3111 to reserve your space
in the next *Dr. Dobb's Journal*.

$2.50
($3.00 in Canada)

# DR. DOBB'S JOURNAL

For Users of Small Computer Systems

Number 60 October 1981 Volume 6, Issue 10




**TEK4010: A Tektronix Simulator**
*Anthony B. Skjellum*

**BASIC—FORTH**
*C.H. Ting*

**The Conference Tree**
*John S. James*

**and more!**

**How to Run Tektronix Software on Micros . . .**

**. . . a Poor Man's FORTH Computer . . .**

**and Telecom-munications News!**

# DR. DOBB'S JOURNAL
For Users of Small Computer Systems

October 1981 Volume 6, Issue 10, Number 60


# CONTENTS


**TEK4010: A Tektronix 4010 Simulator** ..... 8
*by Anthony Skjellum*

**BASIC-FORTH** ..... 12
*by C. H. Ting*

**The Conference Tree** ..... 14
*by John S. James*

Letters ..... 4

Dr. Dobb's Clinic ..... 7

Of Interest ..... 44


**Cover Illustration: Al McCahon**


**Publisher** Michael Madaj
**Editor** Marlin Ouverson
**Assistant Editor** Julie Anton
**Contributing Editors** Dave Caulkins, Dave Cortesi
**Art Director** Clifford West
**Artist** Barbara Ruzgerian
**Typesetter** Renny Wiggins
**Business Manager** Maureen Christine
**Marketing Director** Craig S. Harper
**Advertising Sales** Kent Safford
**Circulation Manager** Peter Clark
**Circulation Assistant** Leah Dansby



*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025.



© 1981 by People's Computer Company unless otherwise noted on specific articles. All rights reserved.


# EDITORIAL

We are pleased to present BASIC-FORTH in this month's menu of articles. For those who were fascinated by September's all-FORTH issue, but do not yet have the language, this is, in the author's words, "a poor man's FORTH machine." We were tempted to call it *hors d'forth* because it is just enough to whet the appetite and raise some interesting possibilities.

There is much important work to be done in the pages of *DDJ*. New machines are in production, and more software is being written than you can shake a stick at. They deserve the strong, objective reviews only this journal can provide. And our loyal authors and contributors are making exciting advances which could fill three of our magazines! We are looking forward to publishing soon a mass of groundbreaking work in the fields of telecommunications, artificial intelligence, small-c, and other crucial topics which are begging for the kind of coverage *DDJ* readers like to get – and give.

In other words, *Dr. Dobb's Journal* must expand to provide you with more pages of solid, useful software and systems tools. But we need your help to do this. Ours is a small, friendly company that depends on your subscription to pay the bills. (We don't have a big corporate purse string to pull, or to influence us, either.) Please take the time to look at the categories of Donating Subscribers here. Your contribution of fifty dollars, or one hundred or one thousand dollars will go a long way toward helping *Dr. Dobb's* finest to expand its practice.

**Donating Subscribers**

**Contributing Subscriber**: $50/year ($25 tax deductible). **Retaining Subscriber**: $75/year ($50 tax deductible). **Sustaining Subscriber**: $100/year ($75 tax deductible). **Lifetime Subscriber**: $1000 ($800 tax deductible). **Corporate Subscriber**: $500/ year ($400 tax deductible, receives *five* one-year subscriptions).

*DDJ* does a lot of things very well, but needs to do even more. We list the names of our Donating Subscribers in each issue because they are as important as the people who put this journal together. Won't you join us?

Marlin Ouverson
Editor


**Subscription Rates:** $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. dollars, drawn on a U.S. bank.

**Writer's Guidelines:** All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

**Contributing Subscribers:** Eric H. Johnson, Jerry Horn, Edward W. Carr, Steven Fisher, Thomas H. Grohne, G. V. Elkins, Robert M. Connors, William J. McEown, Robert C. Luckey, Kim Leary, Robert N. Stabler, DeWitt S. Brown, John B. Palmer, E. Glion Curtis, Jack Miller, Burks A. Smith, Transdata Corporation, Mark Ketter, John W. Campbell, Friden Mailing Equipment, Howard E. Decker, Jim Wilson, Frank Lawyer. **Retaining Subscribers:** Osamu Shimizu, Scott B. Guthery, Robert L. Edens, Titus Purdin. **Sustaining Subscriber:** Gordon Eubanks, Jr. **Lifetime:** Michael S. Zick.

**Foreign Distributors UK & Europe:** LP Enterprises, 8/11 Cambridge House, Cambridge Road, Barking, Essex, 1G11 8NT, England (Tel. 01-591-6511, TLX 892395). Hofacker-Verlag, Tegernseer Strass 18, D-8150 Holzkirchen, West Germany. **Asia & Australia:** Computer Store, POB 31-261, 22B Milford Road, Milford, Auckland 9, New Zealand. ASCII Publishing Co., Ltd., Aoyama Building 5F, 5-16-1 Minami Aoyama, Minato-ku, Tokyo 107, Japan.

# LETTERS




## Bugs Found in Santa Barbara

Dear PCC:

In issue 55, May '81 . . . there is a (untested?) listing of SBTB for the 6809 processor. There are many bugs in the code. Several of them have been fixed, but it requires reassembly. Reassembly is no problem, since I have thrown out the macros and built the keyword tables. If anyone would like the current version that I have get in touch with me. It has been modified somewhat to fit my system.

The bugs (were) are as follows:

'#' did not work . . . a patch here is an ok fix.

patch $0007 with $23 (#)

patch $0FD with $26 (BNE ONE)

If the program following is used the errors can be found:

```
10 FOR A=-1 TO 1
20 FOR B=-1 TO 1
30 PRINT #1, A, '<', B, ; IF A<B
   PRINT 'TRUE',
40 PRINT
50 NEXT
60 NEXT
```

You will see that -1 cannot occur with 'Unary' the way it is. Nor will '<' test properly, the instruction at '<' is BLS which is an >>>UNSIGNED<<< test. The possible condition codes at this point can be zero, minus, overflow and carry (borrow).

'Unary' was fixed thus:

```
UNARY  JSR    <EXPR
       COMA
       COMB
       ADDB   #1
       ADCA   #2
       JMP    <TVHOW    ; but of
```

course this requires reassembly. . .

If the area between $00A0 and $00FF is empty, then at 'Unary' a JUMP or LBRA can be patched in. JUMP to $00A0 and patch in:

```
00A0    43
00A1    53
00A2    C801
00A4    8900
00A6    0E38    * JMP <TVHOW
```

Paul T. Barton
1412 N. E. 46th Ave.
Portland, OR 97213

## Micro AI?

Dear Sir:

Re your forthcoming issue on artificial intelligence, the problem of explaining AI ideas to a general audience has been a primary interest of mine for some time. It's important.

Some caveats, however. AI and small home computers are mostly incompatible. AI requires a certain critical mass of hardware and software facilities (at least 1000K main storage for one thing) that are likely to be well beyond the home system. Programs that seem clever *can* be written for small computers – it's just that they can't demonstrate sufficient flexibility to deserve the title of "intelligent."

However, there's a certain creative and educational value in building "toy" systems that use ideas extracted from more complicated AI systems. For instance, representing structures of facts in the computer and answering questions about them, as per my article in *Creative Computing*, October 1980. I'll think about writing a short article for you on this, more on implementational considerations than the *C. C.* article.

Another interesting area is production (or rule-based) systems. Sequential-flow programming languages have so biased the way programmers tend to program that just seeing a few simple examples of a control structure in which a global ordering is not specified, yet it's clear what is going on from strict constraints on productions, might be very exciting for your readers. Having been at Stanford, medical diagnosis programs spring immediately to mind (perhaps someone could write an interactive program for common home ailments), but some of the complex Adventure-style games also use these production systems ideas, and they might be an easy way to convey sophisticated ideas to a general audience.

Language understanding is another possible area. Certain kinds of activities have very limited vocabulary and syntax in their description, and it might not be hard to write a program to converse about them. However, such programs are not AI unless they embody some general underlying principles, and are not just *ad hoc*.

Language *production* (to either writing or speech) is not really an AI issue, since there's a rather straightforward algorithm to do it. Thus articles about it are probably not appropriate for your issue. Similarly with robots. Actually being able to move around in a human-like way to some extent is just a gimmick, and it is almost never necessary for computers to be able to do intelligent things. Thus with the exception of a few narrow kinds of jobs, AI researchers have been quite uninterested in robots.

Sincerely,
Neil C. Rowe
787 E. Illinois
Lake Forest, IL 60045

## Malpractice?

To the Editor:

I am a longtime reader and find the contents of *Dr. Dobb's* superior in quality. It makes a bright spot in the month's mail. I wish to make two comments.

Issue fifty six (56) contained Jonathan Mill's first installment of the Z80 to 8086 Cross Assembler. I would enjoy further segments of this useful series.

Two letters in issue fifty seven (57) concerning Small-C bug fix included statements concerning the availability of the "improved" compiler at substantial cost (approximately $60.00). This is an objectionable practice and should not be condoned by *DDJ*. Mr. Cain was kind enough to publish Small-C for anyone to use and Mr. Woods provided corrections and enhancements. These actions have done substantial service to the use of high level languages for systems programming. The availability of the Compiler for user modification encourages use and improvement. I do not object to the sale of an improved version if Mr. Cain chose to allow it. I do object to the practice of "Letter Advertising." If you plan on selling a product through *DDJ* at least pay the minimal cost of buying space. Additionally I have not seen a single article pub-

lished which used the C-language, though the algorithm section has been utilizing it for some time. Could we encourage development of the language through use?

I could foresee an issue devoted to the program developed with the Small-C compiler (hopefully not entirely product reviews). Remember that *DDJ*'s main reason for existence is to publish useful programs.

I would appreciate publication of this letter, the editor's and any reader's response to it.

Sincerely,
Herb Calhoun
414 Norman
Euless, TX 76039

*Dear Herb:*

*Thanks for bringing the subject up. The gray line between providing information versus "letter advertising" is a topic which recurs frequently at our offices. Whether it is in a letter or an article, each commercial reference is weighed carefully against the value of the information which goes along with it. We want readers to have the information; we don't want to subject them to sales pitches disguised as helpful hints. If we slip too far out of that gray area – into the white* or *the black – I know our readers will send along a friendly reminder like yours to let us know.*

*About development of the Small-C language through use . . . yes, yes! If enough of* DDJ*'s faithful authors produce useful and de-bugged programs, Doc says he will even consider a special issue. And getting Doc to approve a theme issue on just one subject is like – well, you'd have to be there to see it!* – MDO

Dear Mr. Ouverson:

Herewith my note for *continuation* of J. Mills' article pursuant to a "Cross-Assembler for the 8080 or Z80."

For me, these types of articles – along with those of Dr. Dobb's Clinic – are of the greater interest. Good substantial programs along with the tools needed in software creation, i.e., proven algorithms, are a joy to find – especially when I perceive my subscription cost has returned to me something of benefit. I may immediately use or save for use in the future more of your articles than others.

In another vein: I'm *not* perturbed with your policy of paid advertisement. If reviews can be objective along with advertisers' pressure then do it.

J. Sloane
RD#1 Box HF-17
New Freedom, PA 17349

## Stay Tuned . . .

Dear Sir,

I read with interest Jonathan Mills' article on his Z88 Cross-Assember in Volume 6, Issue 6.

I am a keen reader of your magazine but it is not often that code of the size and usefulness of XASM-86 is published in the public domain. I for one would appreciate your publishing the entire package in installments.

Yours faithfully,
W. Dickson
19 Grove Road
Mt. Albert,
Auckland,
NEW ZEALAND

## I'm OK, Heath OK

Dear Mr. Ouverson:

I would like to respond to the letter of complaint by Mr. Hoffman in issue 56. He made the comment that Heath would not sell the hardware required to install a 5¼" disk drive in a Heath H17, and that there was no record of the part numbers needed for such an installation.

I can only offer personal experiences to refute his statements, but I have dealt on a routine basis with the Heathkit Centers in Seattle, Kansas City and Alexandria (Washington, DC) and feel that Mr. Hoffman must have stumbled across an uncooperative salesman in his efforts. The Alexandria store stocks the required brackets in large numbers, as this has proved to be a popular modification. For his, and other readers' benefit, the required parts are:

| Heath # | Description |
|---|---|
| 206-1331 | Mu-Metal Shield |
| 204-2420 | Top Floppy Bracket |
| 204-2421 | Bottom Floppy Bracket |
| 205-1804 | Support Bracket |
| 255-47 | 3/8" Spacer |

I have a great many friends in the Blacksburg area who own other than Heath computers, and I unhesitatingly match Heath's customer support against that of any other maker. Some of the tales they could tell of poor or non-existent support would stand your hair on end.

Parenthetically, I should add that when I spoke to the folks at the Alexandria store (and there are few more helpful, understanding computer dealers around) I learned that Heath is rewarehousing their parts activity, with a resulting slow-up of parts shipments. I was told that this was nearly complete, and that parts are again flowing. Case in point: Heath now has a very quiet muffin fan you can use to replace the older, noisy unit in the H89 and H17-3. Made in France, I understand. Must be a moral there somewhere.

Sincerely,
D. C. Shoemaker
2000 A. Foxridge
Blacksburg, VA 24060

## Ill-Timed

Dear Marlin:

I have noticed a small error in Mr. Knippenberg's program CAD for Time Delay Circuits (Volume 6, Issue 5). According to the Signetics integrated circuits handbook, the formula for calculating output pulse width for the 74123 is:

Tw = 0.32 Rt Cext (1 + 0.7/Rt)

Where: Tw is in nanoseconds
Rt is in K ohms
Cext is in picofarads

In line 915 of his program, Mr. Knippenberg has changed the formula to read:

T = 0.32 * R * C * (1 + 0.7/R)

Where: T is in microseconds
R is in ohms
C is in microfarads

The obvious error is that he is multiplying the product of the RC constant by 1 plus .7/(R*1000) instead of 1 plus .7/R. This can be corrected by changing line 915 to read:

```
T = 0.32 * R * C * (1 +
    0.7/(R/1000))
```

Likewise, line 920 should be changed to:

```
T = 0.31 * R * C * (1 +
    0.7/(R/1000))
```

Sincerely,
Andrew P. Beck
P. O. Box 571
Jackson, NJ 08527

## Any Takers?

Dear *DDJ*,

Almost three years ago, I submitted two short pieces of code along with a challenge to your readers, and the letter was published in *DDJ* #29. The challenge concerned code for saving and restoring 8080 registers. Since then, fourteen responses have appeared in your Letters department (at last count, which was up to and including *DDJ* #56), and an unknown number of other letters have been sent in by readers but not published. The thread connecting all of these letters has obviously become tangled and confused, so here's an attempt to review and consolidate everything that has appeared.

It all started because an office mate and I began discussing the idiosyncrasies of various instruction sets. I mentioned puzzles I had seen from time to time for various computers that were designed to test a programmer's ability to fathom the tricks and subtleties possible in machine and assembly language. For example, how can a program for the 6800 clear ALL of memory, including the program itself? (Again see *DDJ* #29, page 43.)

In an attempt to create a puzzle for the 8080, I thought for a few minutes and invented the problem of devising reentrant subroutines for pushing and pulling all registers, using as few bytes of code as possible. In a little while, I had a proposed solution (for an admittedly poorly stated puzzle, since I neglected to mention all registers except SP were supposed to be stacked and yet left unmodified, although it's obvious my proposed solution was trying hard to do just that). I typed it into a system, assembled it, listed it, and sent it to *DDJ*.

The first published response, from William Moss in *DDJ* #33, is in fact NOT reentrant because of his use of SHLD and LHLD. The next response, from Robert Minnihan in *DDJ* #35, destroys HL.

Charles Marvin's response appears on the same page, and he points out both the reentrancy problem in Moss' code and a problem with the carry bit in my original solution. The DAD instruction does not always "destroy" (as in "complement the previous state of") the carry bit as Marvin states, but it does always either set or clear it (contrary to the claims made in the comments of my solution), so the longer 18-byte alternate code I proposed in comment form in my solution does appear necessary. But Marvin goes on to give a better 16-byte version for the push routine and a byte-saving 6-byte pull routine, thereby supplying the best known solution.

Harry Stewart then sent in a 6800 solution for a similar problem (*DDJ* #37), but I have not inspected it closely, and *DDJ* #33 contained three letters about the puzzle. John Bockelmann sent in, "as a further example of saving machine status," non-reentrant code written by Alex Caemerer that saves and restores all registers and the stack pointer. Unfortunately, there was no reference to any of the previous letters, so many subsequent responses began to address the problem of saving the state of an 8080 without regard to the intentions of the original puzzle. On the next page, Bruce Smith has a version of Moss' code that is not only non-reentrant, but also self-modifying.

The last letter in *DDJ* #38 about the puzzle, from D. C. Sessions, "is hopefully the last echo of the 8080 register storage/recovery question," but Sessions' proposed solution is virtually identical to Minnihan's from *DDJ* #35 (no doubt independently arrived at).

In *DDJ* #40, we find Jon Bokelman (from California) responding to *DDJ* #38's John Bockelman (from New Jersey) with a one-byte routine! Jon reviews the infamous DAD side effects, and then goes on to propose a completely new save-the-state problem, with the promise of a solution. On the next page, James Monagan writes that Caemerer's routine falls victim to the DAD effect.

In *DDJ* #41, Orv Balcom ("to keep the ol' pusher-popper controversy going") and Chester Quinn both point out Bockelmann/Caemerer's DAD bug, and both submit code to fix it. The followng page contains Bokelman's promised solution to his earlier stated puzzle.

That brings us to the last two letters on the subject, in *DDJ* #44. First, L. Barker supplies a 37-byte solution to Bokelman's problem, but the code is not straightforward. It is followed by a letter from Jim Howell, who, thinking that Balcom and Quinn were writing about Bokelman (they were actually writing about Bockelmann), points out that they use the stack even though Bokelman's problem disallowed it. Howell then goes on to carefully explain his 21 bytes of code for saving registers. He leaves HL and SP changed, but restoring them should only add four bytes to his code.

Howell's letter ends with "where was the original statement of the problem and what was its motivation?". I think this recap answers his question, but I'll be surprised if it can be published without causing another response.

Mike Gabrielson
Box 2692
Stanford, CA 94305

## Moment of Truth

Gentlebeings:

This is a belated reply to Ray Duncan's article "Using (and misusing) the Z-80 Microprocessor," in *DDJ* #55. There is an enthymene in Duncan's article that it is *always* appropriate to optimize time in preference to space.

That isn't, in my experience, universally true.

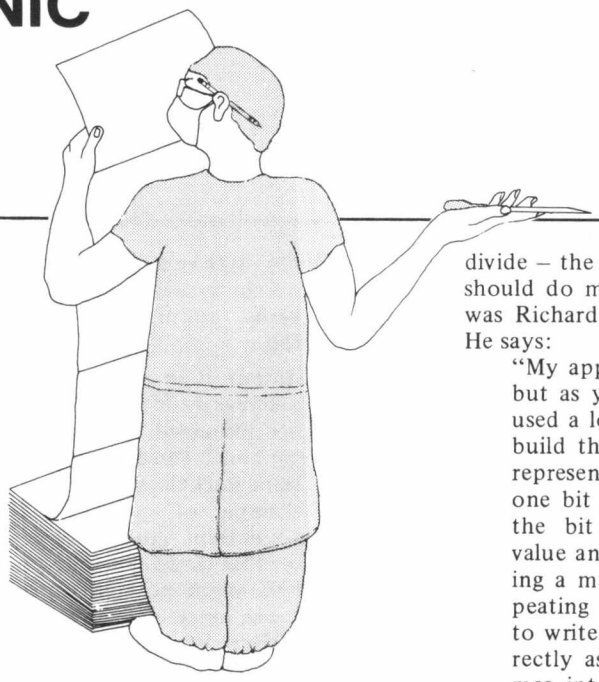
It probably makes sense to be concerned with time in a piece of code that gets executed a google or so times, but it may make much more sense to optimize space in a piece of code that gets executed only once or twice. Somewhere in the middle there's a "moment of truth" where a decision really does have to be made.

There's another place, however, where Duncan's second enthymene – that no one uses interrupts – is probably more serious. The use of the EXX instruction is not to be undertaken lightly in an environment where some part of the system is interrupt-driven, since one of the purposes of the EXX instruction is to provide a set of working registers to a process that may go visit a great-aunt in Peoria if not serviced rapidly. If you, as a programmer, have used the EXX instruction to "make a hole" for a subroutine, you may be in for a most peculiar shock when you get back from the subroutine and discover that the operating system has sliced you thin and warmed you over – if you notice anything at all. (I have this image of trying to do a JP (HL) when (HL) was stuck in HL' during a subroutine and there was an interrupt. . . .)

Sincerely yours,
Peter Zilahy Ingerman
40 Needlepoint Lane
Willingboro, NJ 08046

DDJ

# DR. DOBB'S CLINIC



by D. E. Cortesi, Resident Intern

## Binary to Decimal

In the June "Clinic" (*DDJ* #56) we presented FASTDIV10, an 8080 assembler routine to divide by 10. It was specified thus:

Input: HL = n, $0 \leqslant n \leqslant 32767$
Output: HL = quotient(n/10)
A = remainder(n/10).

Such a routine, we supposed, had to be at the center of any binary-to-decimal conversion routine. We moaned a bit about how there didn't seem to be any faster way to do it. And, of course, readers responded with a variety of improvements. Some took on the overall task of converting a binary integer to decimal. Others concentrated on the problem of dividing a 16-bit number by 10 or any other small integer. We'll examine the latter first.

One warning: only a few of these routines have been tested by us, and all of them are subject to typesetting errors. Don't use one until you understand it well enough to verify it, or else test it thoroughly.

## Shifts and Adds

Shortly after our wish for a divide routine that used shifts and subtracts went to the printer, a review copy of Donald Knuth's *Seminumerical Algorithms* (Addison-Wesley, 2nd ed. 1981, highly recommended) came into the office. In it we found Knuth's exercise 4.4.9, which revealed just what we'd been asking for: division by ten using shifts and adds. With some labor we worked it up as Z80 code (Listing 1, p. 36).

No sooner had we done that, than Dennis Allison wrote, pointing out that he'd described that solution in a prior article ("Converting Binary to Decimal," *DDJ* #42). Dennis presented the algorithm of Listing 1 in C with a clear explanation. Then reader Robert Martin of Northbrook, IL sent us a similar routine in C. Charles Rockwell of Guildford, CT knew of the same idea; his explanation serves nicely to describe Listing 1:

> "Get an approximation for HL/10 by doing an in-line multiply by 0.1 (=0.1999 hex). Save this approximation. Multiply it by 10 and subtract from the original to find a trial remainder. If the remainder is 10 or larger the approximation must be corrected (by incrementing) and the remainder must be corrected (by subtracting 10)."

## Faster, Smaller Division

Two readers, John Morgan of Bellingham, WA and L. Barker, sent near-identical algorithms for doing an 8-bit-into-16-bit divide. Listing 2 is Barker's. Note that it can be adapted to any divisor 3 . . 255 by changing two equates. DIVISR is equated to the divisor. UNDR16 is set true if the divisor occupies four or fewer bits; Farmer takes advantage of that knowledge to "unroll" some unproductive shifts from the main loop.

## Even Faster, But Only Smaller

Allan Ashley of Pasadena, CA sent in what is certainly the most unique division algorithm we've seen. It can handle only numbers less than 2560, for reasons that will become clear as you study the code. It's very quick, and could be useful as in-line code in certain special applications. Ashley's analysis runs thus: given $0 \leqslant HL < 2560$,

$$256*(HL/10) = 25*HL + (6*HL)/10$$
$$= 25*HL + HL/2 + HL/10$$

and his code to generate the latter expression is an education (Listing 3, p. 36).

The code sent by Morgan and Barker will perform division by 10 in just under 700, Z80 clock periods, less than 75% of the time taken by FASTDIV10. The shift-and-add routine of Listing 1 takes about 725 clocks and is quite a bit larger; as noted in the code it could go faster at the cost of more storage. Ashley's routine takes only 226, 8080 clock periods, but its input range is limited.

## Bulky Conversion to Decimal (BCD)

Several readers undertook to convert binary to decimal without the bother of a divide – the kind of lateral thinking we should do more of. The first to respond was Richard Greenlaw of Gahanna, OH. He says:

> "My approach uses lots of code, but as you said, this function is used a lot. The general idea is to build the resulting ASCII string representing the decimal value one bit at a time, starting with the bit representing the most value and working down. By using a macro to generate the repeating code groups I was able to write a program that ran correctly as soon as I got the commas into the macro parameters correctly."

Greenlaw's routine (Listing 4, p. 37), although bulky, isn't much slower than other methods. His approach is original, and there's a good lesson in his use of macros to get the routine up quickly.

## Decimals, Always Adding (DAA)

Charles Rockwell described a good method of converting binary to decimal, given the usual architecture of micro CPUs:

> "Tabb and Roginsky[1] suggest an even faster binary to BCD routine. A binary number can be considered as:
>
> (...(d15)*2+d14)*2+d13)...+d0
>
> and an inner loop to convert binary to BCD using the DAA instruction looks like:
>
> ```
> L: DAD H  ; next digit to carry
>    ADC A  ; A gets 2A+Carry
>    DAA    ; convert to decimal
>    JNC L  ; loop if < 99
> ```
>
> A carry indicates that A is larger than 99, so that this loop can be considered a divide by 100."

Wayne Farmer, Robert Lurie, Martin Manay and Matt Asay all sent conversion routines based on this idea. Farmer's was the most complete; it appears in Listing 5, p. 39. He estimates its execution time as less than 1900 clocks for most input values. A conversion routine based on the best of the divide routines would consume about 1000 clocks for each non-zero digit in the result.

In an example of the kind of synchoronicity that plagues all periodicals, a

*(Continued on page 43)*

**(LISTING ON PAGE 36)**

# TEK4010

## A Tektronix 4010 Simulator

**by Anthony Skjellum**

*Copyright © 1981 by Anthony Skjellum. All rights reserved.*

*Editor's note:*

*We liked Mr. Skjellum's article when we first received it. However, its length and subject matter prompted us to ask the author about the significance of the piece. This is, in part, his reply:*

Why do users want to have a Tektronix simulator? First, there is a tremendous base of software written to Tektronix terminals on large computers. Some of this software (written in BASIC, FORTRAN or perhaps C) could be directly on small computers with the appropriate graphics hardware and simulator software. Inexpensive educational software (e.g. for physics and chemistry) is readily available for use on large systems, and there is no reason why this software could not be downloaded to microcomputers. However, we are not limited to this. As configured, TEK4010 turns the target system (i.e. the microcomputer) into a terminal which can communicate with larger computers.

"The above is certainly a justification for industrial, scientific or educational use. But I also believe that personal computer users can benefit from my program. For example, home users could run Tektronix software on a remote computer with the use of a modem. Software nets (and CBBS systems) could provide Tektronix-compatible software for users to download as well. This would allow home users to run quality software even if they just use their systems as a remote graphics terminal.

"I also believe that it would be very worthwhile for graphics software developed in the microcomputer community to follow the Tektronix graphics standard. This would afford portability of graphics application software written in high level languages and further encourage the use of high-density graphics on micros. As of now, application software written for any high-density hardware tends to be quite system dependent and is generally difficult to transport from system to system.

Allow me to summarize. First, a significant number of machines support high-density graphics and can run TEK4010. Second, there is a large base of Tektronix software which could be used with microcomputers which could not be used without the simulator. Third, cost-effective systems which use my software could readily be configured once the software is altered to meet hardware requirements. This would be useful in industry, education and to the hobbyist. Fourth, the promulgation of the Tektronix standard would only help software portability in the microcomputer community."

---

This article presents a program which simulates a Tektronix 4010 graphics terminal on a Z80 [footnote 1] microcomputer system. This simulator was written to allow applications programs written on a VAX-11/780 [footnote 2] (for use with a standard 4010 terminal) to run unmodified with the Z80 computer system acting as the graphics terminal.

While software of this type is usually system dependent, this implementation is likely to be useful to others planning to write Tektronix simulators for their own hardware configuration. Even though the software depends on the setup of the development system, it could well be modified for other graphics environments. The code was designed for use in a read-only memory and is therefore ROMable.

Included in this article is information about the use of the system including a discussion of its capabilities. A section is devoted to the explanation of the external subroutine package used by this program. The development machine is outlined and motivations behind this implementation are discussed.

### Introduction

The predecessor of this program, TEK4006, was written by Mark Bartelt at the California Institute of Technology for use in conjunction with the Z80 microcomputer systems developed by Professors Dr. R. Gomez and Dr. J. Pine (for whom TEK4010 was written) of the Physics Department. TEK4010 is a major revision of TEK4006 and includes the additional cross-hair cursor capability and associated features. The cross-hair cursor does not require analog inputs to the Z80 system since it uses control characters to move the cursor.

### Motivations

A set of working physics programs was obtained and modified for use on Caltech's Computer Center Timesharing VAX-11/780. However, there are only two Tektronix 4010 terminals on which the programs could be run. This presented a problem since the ultimate goal is to have wide usage of this and other instructional software. Fortunately, seven microcomputer setups are available in the physics department for use in the sophomore physics laboratory and lecture hall. This software was created to allow the use of these computers with the instructional software.

### Operation

TEK4010 handles all communication with the remote computer. Via a buffered, interrupt-driven input scheme, data is accepted from the remote line. The program recognizes Tektronix graphics commands and escape sequences and keyboard input is routed to the remote computer. It is well to note that TEK4010 does not support the Tektronix margin 1 (starting at column 40). Instead, scrolling is supported since this was considered more useful.

When an ESC SUB (1BH 1AH) is sent by the remote computer, a Tektronix 4010 terminal enters the graphic input mode where a cross-hair cursor is used to return a value to the computer on-line. Normally, the user positions the cursor with "thumbwheels" (potentiometers) on the terminal and then presses one of several keys. Depressing any key causes a multi-byte cursor code to be transmitted by the terminal (see Table I). This sequence always includes the current cursor address and input character and may include a postamble depending on how a stappable option is set for the terminal. Most applications programs which use this feature only require the use of a few characters to return the cursor address. It was felt that little generality would be lost if certain codes were intercepted and used as command indicators by TEK4010. The "/" key is the only non-control character used as a command and this can be changed if necessary. Also, note that ∧ C and ∧ Y are transmitted without the cursor address by TEK4010. This permits the user to terminate the program without getting spurious plotting on the display (at least for the VAX/VMS operating system).

The cross-hair cursor is displayed as a large plus sign by TEK4010. It is moved around by depressing a set of control keys at the keyboard. (See Table II.) For example, typing a ∧ D moves the cursor down XOFF points where XOFF is the

## Table I
### Multi-byte Cursor Code

```
byte 1:              char input
byte 2:              <high order of X>
byte 3:              <low  order of X>       (two five bit coordinates)

byte 4:              <high order of Y>
byte 5:              <low  order of Y>

(optional: three cases 1) none, 2) CR, 3) CR, EOT )

byte 6:              none|CR
byte 7:              none|EOT
```

## Table II
### Cursor Controls in Cross-hair Mode

```
D:          Cursor down  [xoff] points.

L:          Cursor left  [xoff] points.

^R:         Cursor right [xoff] points.

^U:         Cursor up    [xoff] points.

^H:         Home cursor.
```

## Table III
### Other Cross-hair Functions

```
^A<x>,<y>:  Address cursor to <x>,<y>.  Enter ^A followed by two
            three digit decimal numbers separated by any delimiter.
            Leading zeroes required.

^B<offset>: Set [xoff] to input value.  Enter one three-digit number
            with leading zeroes as required.

^C:         Sent as is.

LF:         Move cursor near start of current line.

^N:         Toggle display of current cursor location.  Display is
            in lower left of screen.

^Y:         Sent as is.

^Z:         Reset [xoff] to the default value of fifty.

/<digit>:   Enter a <digit> ::= 0|1...|9.  0 represents a division
            by ten.  [xoff] will not get smaller than one.
```

current step size. The default size for XOFF is the current step size. The default size for XOFF is fifty and its value ( ∧B) or by division with the '/' operator. Division is by a single digit 0-9 where 0 denotes a division by 10. The ∧Z command resets XOFF to its default value. The cursor may also be addressed absolutely via decimal input (∧A) and a dynamic display of the cursor coordinates may also be selected ( ∧N). The LF function moves the cursor near the beginning of the current line. This is useful for moving to menus located on the left side of the display. Functions such as ∧N, ∧A and LF were added for the benefit of lecturers who would prepare demonstrations with the simulator. See Table III for a description of these commands.

Since TEK4010 has occasion to do its own output, a rarely used portion of the screen was selected. This is the lower left corner. Commands like ∧V (scaling) and ∧N (display address toggle) use this region. Again, this was possible since the software we use does not display anything on this part of the screen.

Several other commands are available. ∧E dumps the contents of the display to the Axiom 820 microplotter-printer. ∧S does a temporary transmission halt and ∧Q resumes transmission while ∧F provides a local screen clear.

The current hardware provides graphics resolution which is 5/8 of the standard Tektronix terminal (resolution is 640x480; a 4010 has 768 vertical points for plotting and it seems possible to write additional text above this). Some of our applications programs do use more than 768 vertical dots (actual Tek coordinate). Therefore, both ½ and 5/8 vertical scaling is provided. This feature is operator selectable but scaling is always 5/8 in the X-direction. This feature is selected with ∧V . (See Table IV.)

## Hardware

As configured, the computer systems provide 64K of user, random-access memory. It provides display resolutions of 320x240 and 640x480 as noted above. 640x480 is the high density mode which consumes about two-thirds of the random-access memory. This is no problem since the Z80 is used only as a terminal in this mode.

A lower-density mode is also supported by the system (320x240) and this is the standard display state. This resolution is adequate for most programs

running directly on the Z80 system. Consequently, TEK4010 is presently the only program on the system which utilizes the high-density mode graphics. The current system uses 6K of read-only memory. TEK4010 resides in a 2K PROM at E800H and the monitor and graphics drivers reside in two 2K PROMs starting at 0F00H. Conveniently, the PROMs may be switched out to allow new versions of TEK4010 and the system software to be tested in RAM before they are committed to read-only memory.

## Runtime Options

TEK4010 has certain initial conditions which are set by a six-bit dip-switch input. These functions are described in Table V. Systems which do not have a dip-switch input port could read the parameters off a command line to get the proper options. Of course, options available will also be a function of the graphics hardware.

Also, the system will run in the low-density mode (240x320) but the cross-hair cursor capabilities will not scale themselves properly. There is no intention to use low-density Tektronix mode for the present application, so providing the option of low-density, cross-hair addressing was considered unnecessary.

The system software also does software character generation.This makes it possible to have more than one character size. A switchable option is available to choose between standard and big characters. However, this option cannot be used when using the simulator with standard software as we do.

## Interfacing

The development system provides a series of graphics routines which serve as the primitive functions for all graphics-bound programs written for these systems. Several routines are used by TEK-4010 in order to emulate the 4010 terminal. For example, GDRAW draws a line between two points. Other functions manipulate point graphics. These functions will have to be developed for your specific hardware environment. Table VI (page 16) lists the requirements of the actual external routines as well as their functions and attributes (i.e. how they affect registers, etc.). The routines RESCALE, HSCALE, COORDS and HOORDS inside TEK4010 will also have to be changed to reflect the resolution of the graphics hardware available.

**Table IV**
**Other Functions**

| | |
|---|---|
| ^E: | Dump screen to printer (if selected at assembly-time) |
| ^F: | Local clear-screen. |
| ^Q: | Resume transmission. |
| ^S: | Halt transmission. |
| ^V: | Set vertical scaling. Valid entries are H (half) and F (full). |

**Table V**
**Switch-Selectable Options**

The following options are switch selected before execution of TEK4010:

1. black-on-white/white-on-black
2. low-resolution/high-resolution
3. big characters/standard characters

In implementing this software for other systems, I recommend that a comprehensive package be developed which allows TEK4010 to remain relatively unchanged. Certain routines that write directly into the display memory (e.g. BLOCK) will still have to be altered. However, the overall task will be simpler if routines are added which align the conventions of TEK4010 to your system. While this will be slightly less efficient, it will save a major programming task and still result in useful graphics software.

## Assembly-time Options

On a Tektronix terminal, there are three possible straps for selecting the postamble for graphic input mode information. They are : no postamble, CR, or CR and EOT. Three conditional-assembly options permit TEK4010 to provide these three conditions as well (see Table I).

Another assembly-time option is the printer dump command (^ E). Most of the Z80 systems we use have Axiom 820 microplotters for displaying graphics (see illustration). This software will need alteration for whatever graphics output device may be available on your system. If none is available, the command should be disabled by conditional assembly (as in Listing I).

## Conclusion

The complete TEK4010 package is included in Listing I (page 18). Listing II (page 32) contains SUPPORT.LIB which is a macro/equate library used by the simulator and the symbol table is presented in Listing III (page 35).

This package provides a cost-effective way to run Tektronix-compatible software on hardware which already exists in the Physics department at Caltech. Dedicated systems using TEK4010 could provide relatively inexpensive graphics terminals for all sorts of applications. ■■J

**Note**

This software is provided for non-commercial use only. Any commercial use, without the author's written permission, is prohibited.

**Reference**

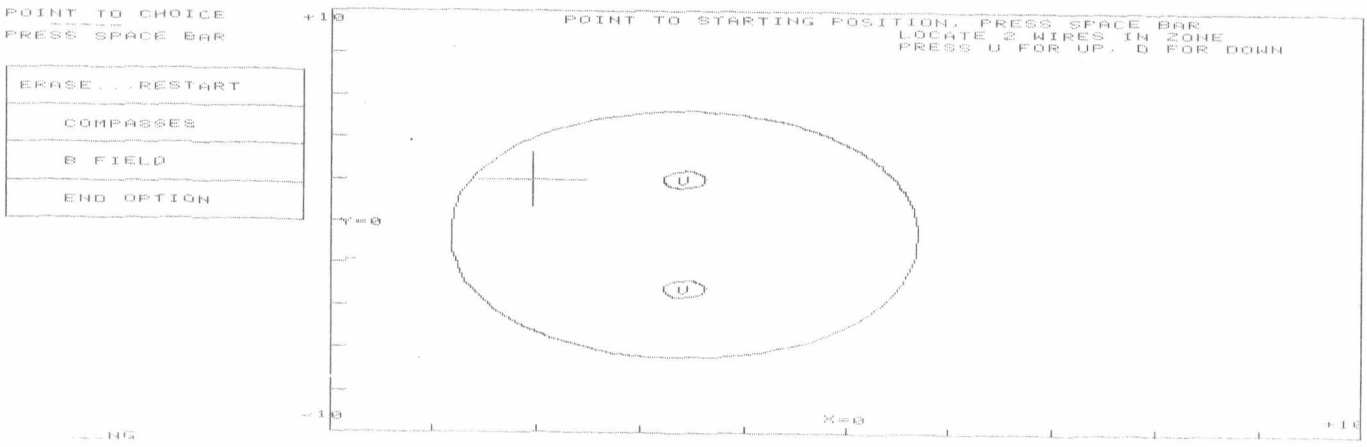
1. Tektronix 4010 – 4010-1 Users' Manual.

**Footnotes**

[1] Z80 is a trademark of Zilog.

[2] VAX is a trademark of Digital Equipment Corporation (DEC).

[3] VAX/VMS is a trademark of DEC.




This output was produced by the program BWIRES.BAS by R. M. Grant and S. C. Wheeler, Dept. of Physics, Denison University, Granville, OH 43023. The program runs under PDP-11 BASIC+2 on the VAX-11/780.

nb: PDP, VAX, BASIC+2 are trademarks of DEC.


# Influence the Future! Join ComputerTown™

*The future in microcomputer technology is very exciting. The potential is enormous. We want to ensure that everyone has an opportunity to learn about the options and share in the benefits that personal computing offers. Please join us in a partnership to make this possible.*

## WHAT IS COMPUTERTOWN*?

A ComputerTown is a place where people of all ages have access to computers. Its goal is to offer everyone an opportunity to become "computer literate" in an informal educational setting. The present organization serves as a link between more than 120 ComputerTown sites worldwide and as an information network between people with an interest in the growth of computer literacy on an international scale.

## *DDJ* READERS MAKE A DIFFERENCE!

Always on the cutting edge, *DDJ* readers have the knowledge and influence to blaze a trail for computer literacy. Share your expertise or spread the word to your associates. As a member you become part of our internationally distributed resource list.

ComputerTown is entering a new stage of development, gearing up, not only for increased membership, but for increased products and services you can take part in. The first books in the ComputerTown series will be available this winter and spring. In addition, work has already begun on a series of informative booklets covering a wide range of topics such as selecting hardware and software, exemplary educational software, exemplary educational software by age group, staff training, funding information, micros in special education and more. We are also developing courseware, tutorials, and teaching materials for a variety of computer literacy classes.

## MEMBERSHIP BENEFITS

**All members receive:**

- **ComputerTown News Bulletin.** As a member, you get a free subscription to this newsletter which covers the growth of computer use in schools, libraries and homes around the country.
- **ComputerTown Publication Discounts.** Members get a special price on all ComputerTown publications including: ComputerTown Book Series, Special Report Booklets, *Bulletin* Back Issues and Bound Volumes. In the future we will offer discounts on various computer magazine subscriptions and courseware materials.
- **Seminar Discounts.** You are entitled to discounts on selected ComputerTown-sponsored conferences and seminars in several cities.
- **Support Services.** Our staff provides services such as resource referral and site-development troubleshooting (depending on membership category). These services are designed to adapt to your needs.
- **Membership Certificate.** This certificate displays your affiliation with a worldwide network promoting computer literacy.

**Leader, Group and Commercial Memberships receive:**

- **Free Implementation Package.** This manuscript details over two years of research and development on public microcomputer access and serves as a guide to developing your own computer literacy activities.

**For more information, circle reader service no. 46, or write: Fritzi Lareau, Membership Services.**

***A Project of People's Computer Company 1263 El Camino Real, P.O. Box E, Menlo Park, California 94025 (415) 323-3111**

# BASIC-FORTH

## A poor man's FORTH computer, and an invitation to FORTH language

by C.H. Ting

My favorite definition of the FORTH language is "a programming language by which a programmer creates a set of instructions as the solution of his programming problem." The fact that FORTH allows and encourages the programmer to extend and modify its instruction set makes this language unique and exciting, and consequently attracts a host of religiously devoted followers. However, there are still two factors impeding FORTH's even wider growth and dissemination; the peculiar syntax using postfix (reverse Polish) notation, and the expense of a computer which can host a FORTH operating system (above $2000). Such a computer must have at least one diskette drive for bootstrapping and program storage.

There are more than 300,000 small BASIC computers in this country with about 8K bytes of RAM memory and cassette recorder as the only means for program and data storage. Obviously, a FORTH system written entirely in the BASIC language would have definite appeal to owners of these BASIC computers. BASIC-FORTH is designed for these computers with minimum resources. It fully utilizes the facilities already resident in the BASIC operating system to convert the BASIC computer into a dictionary-based FORTH stack computer. BASIC-FORTH's vocabulary includes 12 regular FORTH stack instructions, and 17 BASIC functions converted into FORTH instructions. In addition, the control structure instructions such as IF, ELSE, THEN, BEGIN, UNTIL, DO, and LOOP are also included. These instructions and their stack effects are listed in Table 1.

The bulkiest part of this program is the NUCLEUS which contains the DICTIONARY, a linked list of primitive FORTH instructions coded in BASIC (lines 900 to 1522). At the end of the dictionary is the number conversion routine (lines 1600 to 1608). The operating system of this FORTH system is the TEXT INTERPRETER (lines 30 to 138), which parses lines of input as instructions, and executes the instructions in sequence by first searching the dictionary. An instruction found in the dictionary is immediately executed. If the instruction cannot be located in the dictionary, an attempt is made to convert it into a number and the resulting number is pushed onto the data stack S. If the instruction is neither a member of the dictionary nor a number, an error condition is generated, forcing the execution of line 29. This action aborts the current line of

*C. H. Ting, c/o NDT Technology Laboratory, P. O. Box 504, Sunnyvale, CA 94086.*

**Table 1**
**BASIC-FORTH Instructions**

**Stack Instructions**

| | | |
|---|---|---|
| R? | | Type out all stack data. |
| | (n ---) | Type out the top number on stack and remove it. |
| DUP | (n--n n) | Duplicate top of stack. |
| DROP | (n--) | Discard top of stack. |
| SWAP | (n1 n2--n2 n1) | Exchange top two stack items. |
| OVER | (n1 n2--n1 n2 n1) | Make copy of second item on top. |
| ROT | (n1 n2 n3--n2 n3 n1) | Rotate third item to top. |
| PICK | (n1--n2) | Copy n1th item to top. 1 PICK=DUP, 2 PICK=OVER. |
| ROLL | (n--) | Rotate nth item to top. 2 ROLL=SWAP |
| R@ | (--n) | Copy top of return stack to data stack. |
| >R | (n--) | Move top item to return stack. |
| R> | (--n) | Retrieve item from return stack. |

**Arithmetic, Logical, and Functional Instructions**

| | | |
|---|---|---|
| * | (n1 n2--prod) | Multiply. |
| / | (n1 n2--quot) | Divide. |
| + | (n1 n2--sum) | Add. |
| - | (n1 n2--diff) | Subtract. |
| ABS | (n --\|n\|) | Absolute value of n. |
| ATN | (n--arctan n) | Arctangent of n. |
| COS | (n--cos n) | Cosine of n. |
| EXP | (n--exp n) | e to the nth power. |
| INT | (n--integer n) | Integer part of n. |
| LOG | (n--ln n) | Natural logarithm of n. |
| RND | (--n) | Leave a random number between 0 and 1. |
| SGN | (n--sign) | Leave 1, 0, or -1 depending on sign of n. |
| SIN | (n--sin n) | Sine of n. |
| SQR | (n--√n) | Square root of n. |
| TAN | (n--tan n) | Tangent of n. |
| ↑ | (n1 n2--n3) | Exponentiation. |
| = | (n1 n2--n3) | True if n1 is equal to n2. |
| > | (n1 n2--n3) | True if n1 is greater than n2. |
| < | (n1 n2--n3) | True if n1 is less than n2. |

**Control Instructions**

| | | |
|---|---|---|
| IF | (n--) | If n is true, continue execution. Otherwise, skip to ELSE or THEN. |
| ELSE | | Skip to THEN. |
| THEN | | Continue execution. |
| BEGIN | | Save instruction pointer on return stack. |
| UNTIL | (n--) | Branch to BEGIN if n is false; otherwise continue. |
| DO | (n1 n2--) | Save instruction pointer, loop limit n1, and loop counter n2 on return stack. |
| LOOP | | Increment loop counter, if equal or greater than loop limit, branch back to DO. Otherwise, discard three items on return stack and continue. |

instructions, prints out the offending instruction, and restarts the text interpreter from line 30.

It is difficult to implement the colon definitions or other high-level FORTH instructions, because the text interpreter cannot easily extend the dictionary. Between lines 300 and 900, some simulated high-level FORTH instructions are coded as examples. The equivalent FORTH instructions are:

```
: SQUARE    DUP *  ;
: CUBE      DUP SQUARE *  ;
```

The content of a colon instruction, that is, the names of other predefined instructions in the dictionary, is implemented as a string B$, which is concatenated to the calling string I$ currently under interpretation. The nesting level pointer K is incremented to save the current interpretative pointer L(K) in the calling string. L(K) will be the point of return when the called instruction is completely executed. Since L(n) is defined as an array of dimension 10, this program allows ten levels of nesting within high-level FORTH instructions. The depth of nesting can be increased by re-dimensioning L and L0.

The data stack S(n) can hold 40 numbers, and the return stack R(n) can hold 20 numbers. The dimensions of these stacks can also be modified. The return stack is used only by the following instructions: >R, R>, R@, BEGIN, UNTIL, DO, and LOOP. The loop instructions use the return stack to hold pointers for backward branchings. The functions served by the return stack in these loop instructions are different from that of regular FORTH. The user must exercise caution in using the return stack.

All instructions are interpreted and immediately executed without exception. The IF-ELSE-THEN construct is of some interest in its implementation. IF examines the top of the data stack. If the number is true, execution continues to ELSE. ELSE skips all instructions up to THEN, and continues execution after THEN. If the number is false, IF continues execution after the next ELSE or THEN – whichever is encountered first. With this scheme, unlike regular FORTH, IF-ELSE-THEN structures cannot be nested into each other. However, IF-ELSE-THEN structures can be nested into other loop structures.

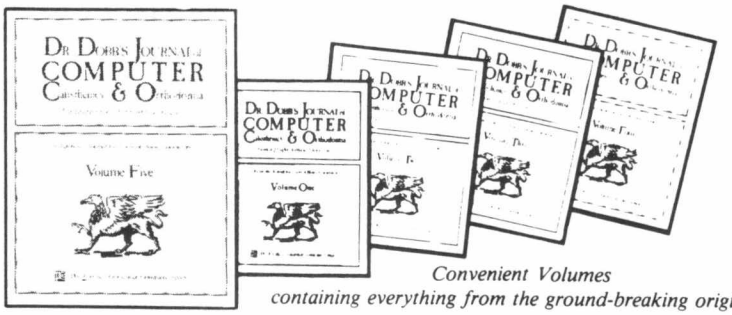
*(Continued on page 43)*

(LISTING ON PAGE 40)


For Your Personal

# REFERENCE COLLECTION

## Thousands of dollars' worth of microcomputer information and listings!




*Convenient Volumes containing everything from the ground-breaking originals*

**Dr. Dobb's Journal Vol. V** – All the ground-breaking issues from 1980 in one volume! Systems software reached a new level with the advent of CP/M, chronicled herein by Gary Kildall and others (*DDJ*'s all-CP/M issue sold out within weeks of publication). Software portability became a topic of greater import, and *DDJ* published Ron Cain's immediately famous Small-C compiler – reprinted here in full!

Contents include: The Evolution of CP/M, a CP/M-Flavored C Interpreter, Ron Cain's C Compiler for the 8080, Further with Tiny BASIC, a Syntax-Oriented Compiler Writing Language, CP/M to UCSD Pascal File Conversion, Runtime Library for the Small-C Compiler, and, as always, even more!

**Dr. Dobb's Journal Vol. I** – The material brought together in this volume chronicles the development in 1976 of Tiny BASIC as an alternative to the "finger blistering" front-panel machine-language programming which was then the only way to do things. This is always pertinent for bit crunching and byte saving, language design theory, homebrew computer construction and the technical history of personal computing.

Topics include: Tiny BASIC, the (very) first word on CP/M, Speech Synthesis, Floating Point Routines, Timer Routines, Building an IMSAI, and more.

**Dr. Dobb's Journal Vol. II** – 1977 found *DDJ* still on the forefront. These issues offer refinements of Tiny BASIC, plus then state-of-the-art utilities, the advent of PILOT for microcomputers and a great deal of material centering around the Intel 8080, including a complete operating system. Products just becoming available for reviews were the H-8, KIM-1, MITS BASIC, Poly Basic and NIBL.

Articles about Lawrence Livermore Lab's BASIC, Alpha-Micro, String Handling, Cyphers, High Speed Interaction, I/O, Tiny Pilot & Turtle Graphics, many utilities, and even more.

**Dr. Dobb's Journal Vol. III** – The microcomputer industry entered its adolescence in 1978. This volume brings together the issues which began dealing with the 6502, with mass-market machines and languages to match. The authors began speaking more in terms of technique, rather than of specific implementations; because of this they were able to continue laying the groundwork industry would follow. These articles relate very closely to what is generally available today.

Languages covered in depth were SAM76, Pilot, Pascal and Lisp, in addition to RAM Testers, S-100 Bus Standard Proposal, Disassemblers, Editors and much, much more.

**Dr. Dobb's Journal Vol. IV** – This volume heralds a wider interest in telecommunications, in algorithms, and in faster, more powerful utilities and languages. Innovation is still present in every page, and more attention is paid to the best ways to use processors which have proven longevity – primarily the 8080/Z80, 6502, and 6800. The subject matter is invaluable both as a learning tool and as a frequent source of reference.

Main subjects include: Programming Problems/Solutions, Pascal, Information Network Proposal, Floating Point Arithmetic, 8-bit to 16-bit conversion, Pseudorandom Sequences, and Interfacing a Micro to a Mainframe – more than ever!

**YES!** ☐ Please send me the following Volumes of *Dr. Dobb's Journal* for $20.75 each.
☐ ALL 5 for ONLY $88, a savings of over 15%!

Please charge my: ☐ Visa ☐ MasterCard ☐ American Express
I enclose ☐ Check/money order

Card # _____ Expiration Date _____

Signature _____

Name _____ Address _____

City _____ State _____ Zip _____

*Prices subject to change.*

Mail to: Dr. Dobb's Journal, P.O. Box E, Menlo Park, CA 94025
Allow 6-9 weeks for delivery.

Vol. I ____ x $20.75 = ____
Vol II ____ x $20.75 = ____
Vol. III ____ x $20.75 = ____
Vol IV ____ x $20.75 = ____
Vol. V ____ x $20.75 = ____
ALL 5 ____ x $88.00 = ____

Sub-total $____

**Postage & Handling ____**
**Must be included with order.**
*Please add $1.25 per book in U.S.*
*($1.75 each outside U.S.)*

**TOTAL $____**

13

# The Conference Tree

## Computer Conferencing on Personal Computers


### Abstract

Computer Conferencing allows group meetings which are non-simultaneous in both time and place. A typical conference may run 24 hours a day and last for months, with individuals interacting at their convenience, using a terminal and the telephone. The Conference Tree, a system now available on the Apple, allows such meetings on a small computer.


The Conference Tree program was originally designed for the personal use of the author, who was swamped by questions about FORTH. Most of the queries needed a search through books and papers, for information the requestor had no other practical way to get. It would be easier to give a phone number to a FORTH computer conference (e.g., 415-538-3580), with instructions to type two carriage returns, and suggest leaving a query on the computer if the information wasn't there already.

The existing computerized bulletin boards would not have handled this job very well. They store messages linearly with all subjects together, not by related groups. It can be hard to find comments, answers or other references to a given message. Their numbering of messages and use of many single-letter commands make the systems harder to learn for those who have never used any computer.

The Conference Tree is a hierarchical (tree-structured) database of messages, designed to be accessed by telephone, with categories and sub-categories determined entirely by the users. After it was made available to the public as a test system (February 1981), many new uses were suggested, some not imagined by the designer. The same software could be an on-line newsletter, a technical meeting or journal with immediate feedback, a movie or restaurant guide, a tool for decentralized political action or for a private business team, or a way for retired executives to keep in touch. In short, it would be a general-purpose tool. The system was productized and documented, and released in July 1981.

### Overview

All data in The Conference Tree consists of MESSAGES. Each message has a name of up to 20 characters, and a text of up to fifty 80-character lines.

Every message except one is attached to one other message, sometimes called its parent. (Similarly we can speak of child and sibling messages). A message may have any number of siblings, or children and grandchildren etc. to any practical depth.

The single message which does not have a parent is the first one in the database; it is the origin of the tree, called the "root." In the Conference Tree, the name of the root is usually 'CONFERENCES'. Naturally 'CONFERENCES' (and several other messages, such as a 'HELP' subtree) are normally added before the system is put on-line for the public.

There are nine commands in The Conference Tree, but only four of them are commonly used, and a single command ('READ') is enough for beginners. 'READ <name>', where <name> is the name of any message, will print its text, and then list the names of any submessages (children). These of course may be read in turn if the user is interested.

This simple mechanism allows great flexibility. 'CONFERENCES' itself is usually a very short message, e.g., "The conferences now going are:". To start a new conference, the user can simply add a message describing the subject to 'CONFERENCES' – although any message in the database can in effect become the start of a new conference.

'HELP' is a one-screen summary of commands and options available to the user. Its children and their descendents are the on-line user documentation. A first-time user need only type two carriage returns to start, and then the welcome message will suggest typing 'READ CONFERENCES' or 'READ HELP'. From this point, the system is self-instructing.

Often a 'USERS' message is also attached to 'CONFERENCES'; it suggests that users may attach a description of themselves and their interests to it. Then mail may be attached to these user description messages – although The Conference Tree is primarily designed for public conferencing, not electronic messages to individuals.

In practice, most messages are comments on, or criticisms or continuations of, their parent message, or invitations to organize new subconferences and break up a discussion topic in certain ways.

There are four special message names in the system: 'CONFERENCES' and 'HELP', which have been described, and 'PRIVATE' and 'GUEST' ('HELP' is special only because the software suggests that the user read it at any sign of difficulty). 'PRIVATE' is optional; if any message with that name exists, it will not allow 'READ' or any other command to report the names of its submessages (children). This facility allows private conferencing without use of a password mechanism, because subtrees under 'PRIVATE' cannot be discovered by any user who does not already know at least one of the names they use.

'GUEST' is only needed for special conferences which are set up as read-only, i.e., which require the system password to add new messages. If 'GUEST' exists in such a conference, it allows an exception, as messages may be added to it without the password.

Besides these four special messages, all names are freely available to the users. Names need not be unique, although they may not be duplicated among a parent and its (immediate) children. 'READ' can reach any message in the conference, not only children of the last-accessed message (although these are checked first).

### Other Commands, and Options

Besides 'READ', other commands will print the first line only of messages ('BROWSE'), or an indented index showing the names of all descendents of a given message ('INDEX'). 'ADDTO' invokes an editor for entering new messages. These four commands are the ones frequently used.

Less-common commands will set 'TERMINAL' options, 'EXILE' or 'UNEXILE' messages (a kind of soft deletion or "soft censorship" which lets the reader make the ultimate decision whether to see the material in question), and print the system's 'SERIAL' number (mainly for protection against software piracy). '+ADDTO' allows moving and changing of messages. All command names except the latter may be abbreviated to their first letters.

Various options will modify the retrieval commands ('READ', 'BROWSE', and 'INDEX'). 'COMPLETE' will cause examination of an entire subtree, not just the message names; e.g., 'READ CONFERENCES COMPLETE' will print everything in the database (except for private conferences). Any subtree can be

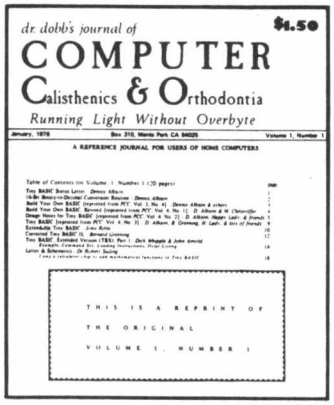

**by John S. James**

*John S. James, c/o CommuniTree Group, P. O. Box 348, Berkeley, CA 94701.*

Since 1976, the most innovative magazine for the technically-sophisticated microcomputer user.

# Dr. Dobb's Journal

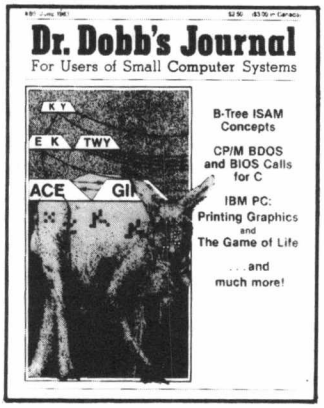
## For Users of Small Computer Systems


dr. dobb's journal of
$1.50
COMPUTER
Calisthenics & Orthodontia
Running Light Without Overbyte
January, 1976 Box 310, Menlo Park CA 94025 Volume 1, Number 1
A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS
THIS IS A REPRINT OF
THE ORIGINAL
VOLUME 1, NUMBER 1

1976



1983




Have you been looking for a magazine that will provide you with serious articles on programming? *Dr. Dobb's Journal* can give you the technical depth you need to understand the latest software advances. We cover every microprocessor with articles on C, Forth, Unix, CP/M and much more. Our reputation for editorial quality is well known. Subscribe now, and discover what industry insiders have enjoyed since 1976.

Coming articles will cover:

- new language development
- 8- and 16-bit technology
- operating systems
- compilers, editors
- assemblers/cross-assemblers
- clever algorithms

**$25**

one year subscription
12 issues
saves $5 off the newsstand price

**YES!** Enter my subscription to *Dr. Dobb's Journal* and bill me later.

name

address

city state zip

**Mail to: DDJ, P.O. Box E., Menlo Park, CA 94025**

searched 'STARTING' at a given creation data, and/or 'BEYOND' a message name (in case a previous printout was interrupted), and/or to 'FIND' all occurrences of a given string. 'EXILED' and 'ONLYEXILED' allow readers to see the soft-censored messages if they want to.

There is no logon or logoff procedure (users just hang up). There is no per-user or per-message password, but there is a master system password. It is usually given to conference editors (called "fair-witnesses"), allowing them to exile, unexile and modify messages.

## Robustness

The Conference Tree is designed for unattended operation, even for weeks at a time, 24 hours a day. Naturally the program needs good recovery from data transmission errors, disk errors, and malicious users.

We know of no way for data errors or a malicious user to crash the system or break through its security. Disk error recovery is good enough that even if all the disks are removed while the program is running, and the drives left empty or filled with unreadable or garbage-data disks, the program keeps running properly although its usefulness is of course impaired; and if the correct disks are reinserted later, the system recovers and starts working normally. This error recovery needs a paper by itself; we will outline it.

The program treats the user as two files (one input, the other output). Every command line is parsed, and any unrecognized input is echoed as an error; the line is not processed further. Only the legal commands and options can be executed, and with a small number of them, a well-designed system can allow high confidence that no possible combination can cause problems.

On the disks, the message texts are stored continguously; any single error will make only one or at most a few messages inaccessible. An index to the messages is stored separately, and it is critical. However, no disk error to the index can stop the program once it is running, because the entire index is kept in memory and never read except at program start-up, when an operator is present. It is completely rewritten to disk whenever it is changed. Therefore any write error will be harmless unless it happens on the last write before the system is taken down. In this worst possible case the operator will have to use a backup next time, but still the system has not failed while unattended.

The data on the disk is always coherent except while a write is actually occurring, perhaps five minutes out of a day of normal use. At all other times the computer can be stopped abruptly with no damage to the data.

If a disk is damaged, it can often be repaired automatically with a special backup which is supplied. The backup does a straight disk copy, except that any erroneous blocks on the disk are not written at all, leaving "holes" where the previous data shows through. Since messages once written to disk are never moved except at compaction (an off-line procedure), the backup disk will be largely identical to the defective one. The backup can be copied first onto a new disk, then the running disk copied on top of it, letting the usually-identical backup data show through the holes. An alternative procedure is to copy the defective disk over and over again onto the same new disk; if every point can be read without error at some time or other, the data is fixed. If these steps fail, the backup disk can of course be used conventionally.

## Turing Test

To protect against the possibility of advertising junk messages being automatically transmitted to a large number of Conference Tree systems, an optional "Turing test" can filter out automatic dial-ins, making sure a warm body is at the originating terminal. This test is merely an operator-set question and answer. A simple, unambiguous question like "What day of the week comes after Monday?" will filter quite well. The test could also be used to admit only those knowledgeable on the subject of a conference – or concerned enough to look up an answer.

It is also possible to require the master password for any access, to run an entirely private conference.

## Limitations

On a 48K Apple, the Conference Tree only allows 321 messages at any one time, regardless of total length, since the entire message index is kept in memory. Each additional 12K of RAM could provide for another 321 messages, however.

The current program is single-user. Multi-user would be difficult with Apples because the minifloppy I/O is not interruptable.

Eventually we may develop larger systems which overcome these limitations, but that is a low priority because our philosophy is decentralized communication. When a conference disk or disks get full, it can divide like an amoeba, with certain subjects or subtopics going off to another computer. This is how we want our system to grow.

Another kind of limitation is that after a message has been saved as permanent, if it is re-edited it must be saved as a new copy without its descendents, and if it is permanently deleted, all its descendents will be removed too. Such operations usually require the system password; there is no per-message password, so users may not come back later and change or delete their messages without the system operator's cooperation. These limitations were deliberately designed, since changing or removing a parent message would change the context of descendents later added by others, an editorial unfairness which sometimes might not be obvious to those making the changes. Users are given every opportunity to rework or scratch their messages before making them permanent, and they can make them anonymously. Sometimes new users are queasy, but in practice the system has been well accepted. It only highlights the fact that use of a public message system is a form of publication, and there is no certainty that any communication can be entirely taken back.

## Proprietary vs. Public Domain

Part of the source code will be published and made public domain – probably enough to run conferences with the real data disks. Other software such as disk compaction and backup will be retained as proprietary, and the object program disk will be protected as these are integral parts of it. We hope this middle ground will enable us to sell and profit from our work, and at the same time encourage development of related systems. We believe that The Conference Tree has applications well beyond what we ourselves will ever be able to develop.

We have no proprietary claim to the command and option name sets, or to the name "The Conference Tree," which we want to be generic for systems of this type.

*(Continued on page 45)*

# TEK4010 (Reference on page 10)

## Table VI
### Description of External Routines and Variables

Routines

XYCHK - Checks X & Y coordinates and finds the corresponding byte address and bit mask for setting or clearing the bit.

Expects:
BC with x-coordinate
DE with y-coordinate

Returns:
Carry set if invalid coordinate.

If valid,
HL with byte address
A with bit mask

Consumes:
A,F,HL

KBCHEK(CONSTAT) - Check for keyboard input

Expects:
none.

Returns:
A is zero if no char available.
A is 0FFH if char available.

Consumes:
A.

KBREAD(CONIN) - Get keyboard input

Expects:
none.

Returns:
A with character.

Consumes:
A.

SUBENT - Save registers on subroutine entry.

Expects:
none.

(BC,DE) with point to draw to.

Returns:
Line drawn or reset.
The cursor is moved to the point specified by (BC,DE) on entry.

Consumes:
GCURXP, GCURYP.

GCHSTR - Write a null terminated string.

Expects:
HL points to string.
String is null terminated.

Returns:
Function complete.

Consumes:
none.
(Of course, the cursor is moved if the string is of non-zero length.)

GCHAR - Write character to display.

Expects:
Character in A.

Returns:
Function complete.

Consumes:
none.

GERASE - Erase screen.

Expects:
none.

Returns:
Function complete.

Consumes:
none.

TVSLWB - Set screen to low density white on black.
TVSLBW - Set screen to low density black on white.
TVSHWB - Set screen to high density white on black.
TVSHBW - Set screen to high density black on white.

Expect:
none.

Return:

Returns:
Function complete. Push all 6 sixteen bit registers (IX, HL, DE, BC, PSW, IY in this order) onto the stack along with a false return address so that when a return instruction occurs, a routine is executed which restores the registers. SUBENT uses IY to do some manipulations so that it is corrupted by the call. However, it is restored upon exit from the subroutine which called SUBENT.

Consumes:
IY.

INTENT - Save registers on interrupt routine entry.

Expects:
none.

Returns:
Same as SUBENT but for interrupt service routines. On exit, this routine executes a subprogram called INTEXT which unsaves all the registers and then executes a return from interrupt instruction (RETI).

Consumes:
IY.

GMOVE - move cursor.

Expects:
BC with X address
DE with Y address

Returns:
GCURXP = BC.
GCURYP = DE.

No error checking is done.

Consumes:
GCURXP, GCURYP.
(These variables form the cursor address)

GDRAW - Draw or undraw a line from current cursor loc. to new position given by (BC,DE).

Expects:
A with toggle: non-zero --> clear.
zero --> set.

*(Continued on top of page 16)*

Function complete.

Consume:
none.

SIOSET - This routine initializes both system SIO's.

Expects:
none.

Returns:
Function complete.

Consumes:
none.

KBSET - Initialize Keyboard.

Expects:
none.

Returns:
none.

Consumes:
none.

Variables

CONST - Input byte char

GMDE - Resolution flag

GCHSIZ - Character size

GCURXP - Current X position of cursor

GCURYP - Current Y position of cursor

**End Table**

# TEK4010 (Text begins on page 8)

## Listing I

```
                ;************************************
                ;************************************
                ;****                            ****
                ;**** TEK4010 II.9               ****
                ;**** BY A. SKJELLUM             ****
                ;****                            ****
                ;**** NOVEMBER 10, 1980          ****
                ;****                            ****
                ;**** EMULATES 4010 W/           ****
                ;**** CROSS-HAIR CURSOR          ****
                ;****                            ****
                ;****                            ****
                ;**** TEK4006 BY M. BARTELT      ****
                ;****                            ****
                ;**** TEK4006 REVISED AND        ****
                ;**** EXPANDED TO TEK4010        ****
                ;**** BY A.SKJELLUM.             ****
                ;****                            ****
                ;************************************
                ;************************************
                ;************************************
                ;
                ;
                ;
                ;
                        ;
                        MACLIB  Z80     ; Z80 LIBRARY
                        MACLIB  SUPPRT  ; SUPPORT LIBRARY
                        ;
                        ;
                ;**********************************
                ;*                                *
                ;*      TEK VAR                   *
                ;*                                *
                ;**********************************
                        ;
0100 =          TEKVAR  EQU     0100H   ;BEGINNING OF
                                        ;SCRATCH AREA
                        ;
                        ;
                        ;
0100 #          @ADDR   SET     TEKVAR  ;VALUE FOR
                                        ;SETTING UP
                                        ;VAR TABLE
                        ;
                        LOC     BUFFER,BUFSIZ ;200 BYTE
0100+=                                  ;INPUT BUFFER
                        LOC     BUFEND,0 ;END OF BUFFER
01C8+=                                  ;LOCATION
                        LOC     SIOCTL,1 ;PORT CTRL VAL
01C8+=
                        LOC     SIODAT,1 ;PORT DATA VAL
01C9+=
                        ;
                        LOC     NXTGET,2 ;BUFFER GET PTR
01CA+=
                        LOC     HALF,1  ;VERTICAL
01E1+=                                  ;ADJUSTMENT
                                        ;FLAG
                        ;
                        LOC     DSPFLG,1        ;COORD
01E2+=                                  ;DISP FLAG
                        ;
                        ;
                        LOC     SAVPTR,2 ;
01E3+=
                        ;
                        ; POINTER FOR STACK USED FOR
                        ; CURSOR SAVE - UNSAVE
                        ;
                        LOC     STKST,0 ;
01E5+=
                        ;
0225 #          @ADDR   SET     @ADDR+64
                        ;
                        LOC     STKND,0
0225+=
                        ;
                        ;
                        ;
                        ;
                        LOC     CTABL,(CCNT*2)-1
0225+=
                                        ;COPY OF WHAT
                                        ;IS UNDER CURSOR.
                        ;
                        ;
                        ;
E800 =          ROM     EQU     0E800H  ;START OF PROG
                        ;
E800                    ORG     ROM     ;TEK4010 II
                        ;
                        ;
                        ;
                ;**********************************
                ;*                                *
                ;* CONDITIONAL ASSEMBLY           *
                ;* TOGGLES                        *
                ;*                                *
                ;**********************************
                        ;
                        ;
FFFF =          STRP1   EQU     ON      ;GRAPHIC INPUT
                                        ;MODE END CHARS
                                        ;ENABLED
FFFF =          STRP2   EQU     ON      ;SEND CR
0000 =          STRP3   EQU     OFF     ;NO EOT
                                        ;1/2 TEK INSTEAD
                                        ;OF 5/8
                                        ;FOR ADDRESS
FFFF =          USED    EQU     ON      ;MNEMONIC FOR
                                        ;LEAVING OUT
                                        ;UNUSED ROUTINES
                                        ;BY CONDITIONAL
                                        ;ASSEMBLY.
0000 =          PRNTR   EQU     OFF     ;AXIOM 820 USE
                        ;
                                        ;W/ BIG CHARS
                        JRZ     SETPIO  ;NO...
E830+2803
E832 CD72EB             CALL    BCHARS  ;BIG CHARACTERS
E835 CDDBED     SETPIO  CALL    PIOSET  ;INIT
                                        ;THE PIO FOR SIO
                                        ;INTERRUPT
E838 CD75F8             CALL    KBSET   ;INIT KEYBOARD
                        ;
                        ;
                        LDAI            ;GET HIGH BYTE
E83B+ED57
                                        ;OF INTERRUPT
E83D 67                 MOV     H,A     ;FORM ADDRESS
E83E 2E42               MVI     L,SIOVEC ; LOW ORDER OF
                                        ;ADDR
E840 3604               MVI     M,(SIOISR AND 00FFH)
                                        ;LOW ORDER
E842 23                 INX     H       ;++HL
E843 36EE               MVI     M,(SIOISR SHR 8)
                                        ;HIGH ORDER
                        ;
                        ;
                        ;
E845 210001             LXI     H,BUFFER ;POINT
                                        ;TO START OF
                                        ;INPUT BUFFER
E848 22CA01             SHLD    NXTGET  ;INIT BOTH
E84B 22CC01             SHLD    NXTPUT  ;BUFFER PTRS
E84E AF                 XRA     A       ;ZERO ACC;
E84F 47                 MOV     B,A     ;B=0;
E850 32CE01             STA     NCHIBF  ;INIT
                                        ;CHAR COUNT
E853 32D001             STA     AUSTPD  ;RESET THE
                                        ;"TRANSMISSION
                                        ;STOPPED" FLAG
E856 32D101             STA     MNSTPD  ;RESET THE
                                        ;LOCAL "STOPPED"
                                        ;FLAG
E859 32E001             STA     XMODE   ;TURN OF CURS
                                        ;MODE
E85C 32E201             STA     DSPFLG  ;OFF W/ COORD
                                        ;DISPLAY
E85F 32E101             STA     HALF    ;5/8 SCALING
E862 3E11               MVI     A,CTRLQ ;SEND A
                                        ;^Q AT STARTUP
E864 32CF01             STA     CTRLQS
E867 213200             LXI     H,SOFF  ;SET UP JMP
                                        ;SIZE FOR
                                        ;CROSS-HAIR
E86A 22D601             SHLD    XOFF
                        ;
E86D 214001             LXI     H,XMID  ;MID VALUE ON
                                        ;DISPLAY X
E870 22D801             SHLD    CURSX   ;X ADDR
E873 22DC01             SHLD    PCURSX  ;PREV. X ADDR
E876 21F000             LXI     H,YMID  ;MID VALUE ON
                                        ;DISPLAY Y
E879 22DA01             SHLD    CURSY   ;Y ADDR
E87C 22DE01             SHLD    PCURSY  ;PREV. Y ADDR
                        ;
E87F 21E501             LXI     H,STKST ;START OF
                                        ;SOFTWARE XY
```

```
                        LOC     NXTPUT,2 ;BUFFER PUT PTR
01CC+=
                        LOC     NCHIBF,1 ;# OF CHARS IN
01CE+=
                                 ;BUFFER
                        LOC     CTRLQS,1 ;^Q-^S SND FLAG
01CF+=
                        LOC     AUSTPD,1 ;TRANS STOPPED
01D0+=
                                 ;FLAG
                        LOC     MNSTPD,1 ;"STOP OUTPUT"
01D1+=
                                 ;FLAG
                        LOC     LOX,1    ;TEMPS FOR
01D2+=
                                 ;ADDRESS
                                 ;ASSEMBLY
                                 ;FROM TEK INPUT
                        LOC     HIX,1
01D3+=
                        LOC     LOY,1
01D4+=
                        LOC     HIY,1
01D5+=
                ;
                ;
                        ;
                        ;
1000 =          STAK    EQU     1000H    ;LEAVE PLENTY
                                 ;OF ROOM
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      VARIABLES FOR XHLOOP     *
                ;*                               *
                ;*********************************
                        ;
                        ;
                        LOC     XOFF,2   ;OFFSET FOR
01D6+=
                                 ;MANEUVERING
                        LOC     CURSX,2  ;X ADDR
01D8+=
                        LOC     CURSY,2  ;Y ADDR
01DA+=
                        ;
                        LOC     PCURSX,2 ;OLD X ADDR
01DC+=
                        LOC     PCURSY,2 ;OLD Y ADDR
01DE+=
                        ;
                        ;
                        ;
                        ;
                        LOC     XMODE,1  ;XMODE TOGGLE
01E0+=
```

*(Continued on page 18, column 2)*

```
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      TEK4010 II               *
                ;*                               *
                ;* INITIALIZATION SECTION        *
                ;*                               *
                ;*********************************
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
E800 F3         TEK     DI               ;DISABLE INTERR
E801 310010             LXI     SP,STAK  ;INIT SP
E804 DB78               IN      SWREG    ;GET PARMS
E806 3C                 INR     A        ;ALL 1'S
                        JRZ     NOSWRG   ;IF SO,
E807+2801
                                         ;USE DEFAULTS
E809 3D                 DCR     A        ;RESTORE
E80A F5         NOSWRG  PUSH    PSW      ;SAVE SWITCH VAL
E80B 2123E8             LXI     H,ERASE  ;GET RET ADDR
E80E E5                 PUSH    H        ;SAVE TO FAKE
                                         ;A SUBRTN CALL
                        BIT     BWFLG,A  ;BLACK ON
E80F+CB5F
                                         ;WHITE?
                        JRNZ    BONW     ;YES...
E811+2008
                        BIT     LRFLG,A  ;LOW RES?
E813+CB4F
E815 CA6CF8             JZ      TVSHWB   ;NO...
E818 C366F8             JMP     TVSLWB   ;YES...
                        ;
                BONW    BIT     LRFLG,A  ;LOW RES?
E81B+CB4F
E81D CA6FF8             JZ      TVSHBW   ;NO...
E820 C369F8             JMP     TVSLBW   ;YES...
                        ;
E823 CD60F8     ERASE   CALL    GERASE   ;ERASE DISPLAY
E826 CD72E8             CALL    SIOSET   ;INIT SIO'S
E829 F1                 POP     PSW      ;GET SWITCH
                                         ;VALUES AGAIN
                        BIT     LRFLG,A  ;LOW RES?
E82A+CB4F
                        JRNZ    SETPIO   ;YES...
E82C+2007
                        BIT     BIGCHR,A ;HIGH RES
E82E+CB57
```

*(Continued on page 18, column 3)*

```
                                         ;STACK.
E882 22E301             SHLD    SAVPTR   ;INIT PTR
                        ;
                        IM2              ;SET INTERR MODE
E885+ED5E
                        ;
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      MAIN LOOP                *
                ;*                               *
                ;*********************************
                        ;
                        ;
                        ;
                        ;
                        ;
E887 FB         LOOPX   EI               ;ENABLE INTERR
                LOOP
E888 3AE001             LDA     XMODE    ;CURSOR MODE?
E88B B7                 ORA     A        ;SET FLAGS
E88C C4FCEB             CNZ     UPDATC   ;UPDATE CURSOR
                        ;
                        ;
                        ;
E88F 3AC801             LDA     SIOCTL   ;GET SIO
                                         ;CTRL PORT #
E892 4F                 MOV     C,A      ;
                        INP     A        ;IS CHAR BEING
E893+ED78
E895 E601               ANI     SIOXMT   ;SENT?
E897 CA39E9             JZ      TSTRCV   ;YES...
E89A F3                 DI               ;DISABLE INTERR
E89B 3ACF01             LDA     CTRLQS   ;IS THERE A ^Q
                                         ;OR ^S WAITING
                                         ;TO BE SENT?
E89E B7                 ORA     A        ;
                        JRZ     TSTKB    ;NO...
E89F+280F
                        ;
E8A1 57                 MOV     D,A      ;SAVE CHAR
E8A2 3AC901             LDA     SIODAT   ;GET PORT
E8A5 4F                 MOV     C,A      ;PUT IT WHERE
                                         ;NEEDED
                        OUTP    D        ;SEND IT
E8A6+ED51
E8A8 AF                 XRA     A        ;ZERO ACC
E8A9 32CF01             STA     CTRLQS   ;RESET FLAG
E8AC FB                 EI               ;ENABLE INTERR
E8AD C339E9             JMP     TSTRCV   ;GO PROC
                                         ;RECEIVED DATA
                        ;
E8B0 FB         TSTKB   EI               ;ENABLE INTERR
E8B1 CD18F8             CALL    KBCHEK   ;KBD CHR
                                         ;WAITING?
```

*(Continued on next page)*

# TEK4010

(Listing continued, text begins on page 8)

```
E8B4 DA39E9             JC      TSTRCV  ;IF NOT...
E8B7 CD2AF8             CALL    KBREAD  ;GET IT
E8BA E67F               ANI     7FH     ;STRIP PARITY
E8BC FE16               CPI     CTRLV   ;V-SCALE?
                        JRNZ    KB0     ;NO...

E8BE+2007
E8C0 C5                 PUSH    B       ;SAVE FLAG IN B
E8C1 CDE2EC             CALL    XHVERT  ;PERFORM SCALING
E8C4 C1                 POP     B       ;RECOVER FLAG
                        JR      TSTRCV  ;CONTINUE THERE

E8C5+1872
             KB0
                        IF      PRNTR
                        CPI     CTRLE   ;WRITE SCREEN
                                        ;ON AXIOM 820
                        JRNZ    KB1     ;NO...
                        PUSH    B       ;SAVE FLAG IN B
                        CALL    AXIOM   ;DO WORK
                        POP     B       ;RECOVER FLAG
                        JR      TSTRCV  ;CONTINUE THERE
                        ENDIF           ;PRINTER
                        ;
E8C7 FE06       KB1     CPI     CTRLF   ;CLEAR CODE?
                        JRNZ    TSTKBO  ;NO...

E8C9+2008
E8CB CD2AEC             CALL    XERASE  ;CLR SCRN AND
                                        ;MAKE SURE NO
                                        ;UPDATE IS DONE
                                        ;DIRECTLY AFTER
                                        ;A CLEAR SCREEN
                                        ;WE MUST ALSO
                                        ;ZERO THE CTABL
E8CE CD8FED             CALL    ZCTABL  ;ZERO IT...
                        JR      TSTRCV  ;LOOK FOR INPUT

E8D1+1866
E8D3 FE13       TSTKBO  CPI     CTRLS   ; ^S?
                        JRNZ    TSTKB1  ;NO...

E8D5+2005
E8D7 32D101             STA     MNSTPD  ;STOP LOCAL
                                        ;OUTPUT
                        JR      LOOP    ;BACK TO MAIN

E8DA+18AC
                                        ;PROGRAM LOOP
                        ;
E8DC FE11       TSTKB1  CPI     CTRLQ   ; ^Q?
                        JRNZ    SNDCHR  ;NO...

E8DE+2006
E8E0 AF                 XRA     A       ;ZERO ACC
E8E1 32D101             STA     MNSTPD  ;RESET THE
                                        ;STOP LOCAL
                                        ;OUTPUT FLAG
                        JR      TSTRCV  ;GO CHECK FOR

E8E4+1853
                                        ;INPUT
                        ;
E8E6 57         SNDCHR  MOV     D,A     ;SAVE INPUT CHAR
E8E7 3AE001             LDA     XMODE   ;CURS. STAT
```

```
E921 19                 DAD     D       ;LOOK AGAIN
                        JR      XHLOOP3

E922+18F5
                        ;
             XHFND      ;
                        ; LOOK UP ADDRESS AND CALL IT
                        ;
E924 5E                 MOV     E,M     ;LOW ORDER
E925 23                 INX     H
E926 56                 MOV     D,M     ;HIGH ORDER
E927 2138E9             LXI     H,XHERE ;STD RET ADR
E92A E5                 PUSH    H
                        ;
E92B EB                 XCHG            ;GET EXEC ADDR
E92C E9                 PCHL            ;GO THERE
                        ;
                        ;
E92D E9                 PCHL
                        ;
                        ;
                        ;
                        ;
                        ;
E92E 57      XHEOT      MOV     D,A     ;SAVE CHAR
E92F 3AC901             LDA     SIODAT  ;GET PORT #
E932 4F                 MOV     C,A     ;PUT IT WHERE
                                        ;NEEDED
                        OUTP    D       ;SEND CHAR TO
E933+ED51
                                        ;T.S.C.
E935 CD5CEC             CALL    XHSEND  ;SEND INFO
                        ;
             XHERE      EXX             ;GET REG. SET
E938+D9
                                        ;BACK
E939 3AD101  TSTRCV     LDA     MNSTPD  ;HAS OUTPUT
E93C B7                 ORA     A       ;BEEN STOPPED?
E93D C288E8             JNZ     LOOP    ;YES...
                        ;
E940 F3                 DI              ;DISABLE INTERR
E941 3ACE01             LDA     NCHIBF  ;ANYTHING
E944 B7                 ORA     A       ;IN INPUT BUFFER
E945 CA87E8             JZ      LOOPX   ;NO...
                        ;
                        ;
                        ;
                        ;
                        ;
E948 3D                 DCR     A       ;--NCHIBF
E949 32CE01             STA     NCHIBF
E94C FE32               CPI     SNDCTQ  ;TIME TO
                                        ;SEND A ^Q
                        JRNZ    GETCHR  ;NO...
E94E+200F
E950 3AD001             LDA     AUSTPD  ;DO WE REALLY
                                        ;NEED TO SEND
                                        ;ONE?
E953 B7                 ORA     A
                        JRZ     GETCHR  ;NO...
E954+2809
E956 AF                 XRA     A       ;RESET THE
E957 32D001             STA     AUSTPD  ;FLAG
E95A 3E11               MVI     A,CTRLQ ;REMEMBER
```

```
E98B+2820
E98D 23                 INX     H       ;++HL;
E98E 91                 SUB     C       ;FOUND?
                        JRZ     CTLDSP  ;YES...
E98F+2804
E991 23                 INX     H       ;++HL;
E992 23                 INX     H       ;++HL;
                        JR      CTLLP   ;REPEAT...
E993+18F4
                        ;
E995 5E      CTLDSP     MOV     E,M     ;GET LOW ORDER
                                        ;OF ADDR TO
                                        ;EXECUTE
E996 23                 INX     H       ;++HL;
E997 56                 MOV     D,M     ;GET HIGH ORDER
E998 EB                 XCHG            ;DE=HL;
E999 E9                 PCHL            ;PC=HL;
                        ;
             SETALF     RES     GMODE,B ;CLR GRAPHICS
E99A+CB88
                                        ;MODE
E99C 3AE001             LDA     XMODE   ;IS XMODE ON?
E99F B7                 ORA     A       ;SET FLAGS
                        JRZ     CTLCOM  ;GO TO COMMON
E9A0+2814
                                        ;CODE
E9A2 CD87EB             CALL    COFF    ;CURS OFF
                        ;
                        ;
                        JR      CTLCOM  ;GO TO
E9A5+180F
                                        ;COMMON CODE
                        ;
             SETGRF     SETB    GMODE,B ;SET GRAPHIC
E9A7+CBC8
                                        ;MODE
                        SETB    DRKVEC,B
E9A9+CBD8
                                        ;SET DARK VECTOR
                        RES     HIXFLG,B ;CLR HIGH-X
E9AB+CB90
                                        ;FLAG
E9AD C388E8  JLOOP1     JMP     LOOP    ;BACK TO MAIN
                                        ;LOOP...
                        ;
                        ;
                        ;
             SETESC     SETB    ESCFLG,B ;ENTER ESC
E9B0+CBC0
                                        ;SEQUENCE
                        JR      JLOOP1  ;BACK TO MAIN
E9B2+18F9
                                        ;LOOP
                        ;
E9B4 0E20    TABBLK     MVI     C,BLANK ;CONVERT
                                        ;TAB TO A
                                        ;BLANK
                        ;
             CTLCOM     BIT     GMODE,B ; GRAPHIC MODE?
E9B6+CB48
                        JRNZ    JLOOP1  ;IF SO,
E9B8+20F3
                                        ;THEN WE'RE DONE
```

```
E8EA B7                 ORA     A       ;SET FLAGS
                        JRNZ    XHLOOP  ;GO IF <>0
E8EB+2008
E8ED 3AC901             LDA     SIODAT  ;GET SIO DATA
E8F0 4F                 MOV     C,A     ;PORT #
                        OUTP    D       ;SEND IT
E8F1+ED51
                        JR      TSTRCV  ;AND CONTINUE
E8F3+1844
                        ;
                        ;
E8F5 7A         XHLOOP  MOV     A,D     ;RECOVER INPUT
                                        ;TO T.S.C.
                XHL
E8F6 FE03               CPI     CTRLC   ;CONTROL C?
                        JRZ     XHLOOP1 ;NO CODE SEND
E8F8+2810
E8FA FE19               CPI     CTRLY   ;CONTROL Y?
                        JRZ     XHLOOP1 ;NO CODE SEND
E8FC+280C
                        ;
E8FE FE0D       XHLP    CPI     CR      ;CR?
                        JRNZ    XHLOOP2 ; IF NOT,
E900+2010
E902 F5                 PUSH    PSW     ;SAVE CHAR
E903 CD87EB             CALL    COFF    ;TURN IT OFF
E906 F1                 POP     PSW     ;RECOVER IT
                        EXX             ;MAKE ALT SET SO
E907+D9
                                        ;CORRESPONDING
                                        ;EXX LATER SETS
                                        ;THINGS RIGHT
                        JR      XHEOT   ;BUT ALSO SEND
E908+1824
                XHLOOP1
E90A CD87EB             CALL    COFF    ;OFF W/ CURSOR
E90D CDCBEC             CALL    SCHAR   ;SEND THE CHAR
                        JR      TSTRCV  ;CONTINUE THERE
E910+1827
                        ;
                XHLOOP2
                        EXX             ; GET ALT SET
E912+D9
E913 2154EE             LXI     H,TABL  ;COM TBL
E916 110200             LXI     D,2     ;OFFSET-1
                        ;
                XHLOOP3
E919 34                 INR     M       ;++M;
E91A 35                 DCR     M       ;--M;
                        JRZ     XHEOT   ;IF NO MORE TO
E91B+2811
                                        ;CHECK...
                        CCI             ;COMPARE AND
E91D+EDA1
                                        ; ++HL;
                        JRZ     XHFND   ;WE FOUND IT
E91F+2803
```

*(Continued on page 20, column 2)*

```
                                        ;THAT A
E95C 32CF01             STA     CTRLQS  ;CTRL Q NEEDS
                                        ;TO BE SENT
                        ;
                        ;
                        ;
E95F FB         GETCHR  EI              ;ENABLE INTERR
E960 2ACA01             LHLD    NXTGET  ;GET PTR TO
                                        ;NEXT CHR
E963 7E                 MOV     A,M     ;GET THE NEXT
                                        ;CHARACTER
E964 23                 INX     H       ;++NXTGET;
E965 22CA01             SHLD    NXTGET
E968 B7                 ORA     A       ;CY=0;
E969 11C801             LXI     D,BUFEND ;PAST END
                                        ;OF BUFFER?
                        DSBC    D
E96C+ED52
                        JRNZ    ACTION  ;NO...
E96E+2006
E970 210001             LXI     H,BUFFER ;CORRECT PTR
E973 22CA01             SHLD    NXTGET
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;* NOW WE PROCESS THE            *
                ;* CHARACTER RECEIVED FROM THE   *
                ;* HOST COMPUTER.                *
                ;*                               *
                ;*********************************
                        ;
                        ;
                ACTION  BIT     ESCFLG,B ;ARE WE IN
E976+CB40
                                        ;AN ESCAPE
                                        ;SEQUENCE?
                        JRNZ    ESCSEQ  ;YES...
E978+2046
E97A 4F                 MOV     C,A     ;SAVE CHR
E97B FE20               CPI     BLANK   ;CTL CHR?
E97D FA86E9             JM      CTLCHR  ;YES...
                        BIT     GMODE,B ;ARE WE IN
E980+CB48
                                        ;GRAPHICS MODE?
                        JRZ     ALPHA   ;NO...
E982+2837
                        JR      GRAFIX  ;YES...
E984+1866
                        ;
E986 2141EE     CTLCHR  LXI     H,CTLTBL ;POINT TO START
                                        ;OF DISPATCH
                                        ;TABLE
                        ;
E989 7E         CTLLP   MOV     A,M     ;GET A TST CHR
E98A B7                 ORA     A       ;EOT?
                        JRZ     JLOOP1  ;YES...
```

*(Continued on page 20, column 3)*

```
                        ;
E98A 79                 MOV     A,C     ;GET CHR BACK IN
                                        ;ACC.
                        ;
                        ;
                        ;
E98B CD48F8     ALPHA   CALL    GCHAR   ;DISPLAY CHAR
                        JR      JLOOP1  ;BACK TO MAIN
E98E+18ED
                                        ;LOOP
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;* ESCAPE SEQUENCE -             *
                ;* TAKE APPROPRIATE ACTION.      *
                ;*                               *
                ;*********************************
                        ;
                        ;
                        ;
E9C0 FE1A       ESCSEQ  CPI     CHSUB   ;TURN ON CROSS
                                        ;HAIR CURSOR?
                        JRZ     XHON    ;YES...
E9C2+2811
E9C4 FE05               CPI     ENQ     ;SEND ADDRESS?
                        JRZ     ESCSND  ;YES...
E9C6+2812
E9C8 FE0C               CPI     FF      ;FORM FEED?
                        JRNZ    ESCSQX  ;NO...
E9CA+2005
E9CC CD2AEC             CALL    XERASE  ;ERASE SCRN AND
                                        ;NO UPDATE
                        RES     GMODE,B ;BACK TO ALPHA
E9CF+CB88
                                        ;MODE
                        ;
                        ;
                        ;
                        ;
                ESCSQX  RES     ESCFLG,B ;RESET THE
E9D1+CB80
                                        ;"ESCAPE
                                        ;SEQUENCE" FLAG
                        JR      JLOOP1  ;BACK TO
E9D3+18D8
                                        ;MAIN LOOP
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      XHON                     *
                ;*                               *
                ;*********************************
                        ;
                        ;
E9D5 CD80EB     XHON    CALL    CON     ;CURSOR ON
```

*(Continued on next page)*

# TEK4010

**(Listing continued, text begins on page 8)**

```
                        JR      ESCSQX
E9D8+18F7
                        ;
                        ;
                ;**********************************
                ;*                                *
                ;*      ESCSND                    *
                ;*                                *
                ;**********************************
                        ;
                        ;
E9DA 3AE001     ESCSND  LDA     XMODE   ;CURS MODE
E9DD B7                 ORA     A       ;SET FLAGS
                        JRZ     ESCSND2 ;NO...
E9DE+2805
E9E0 CD5CEC             CALL    XHSEND  ;NO TERM STAT
                        JR      ESCSQX  ;AND EXIT...
E9E3+18EC
                        ;
                ESCSND2
                        BIT     GMODE,B ;GRAPH MODE?
E9E5+CB48
E9E7 C43BEC             CNZ     SENDER  ;SEND ALL BYTES
                        ;
                        JR      ESCSQX  ;...
E9EA+18E5
                        ;
                        ;
                ;**********************************
                ;*                                *
                ;* GRAPHIC MODE -                 *
                ;* DECODE CHARACTER AS PIECE      *
                ;* OF COORDINATE INFORMATION.     *
                ;*                                *
                ;**********************************
                        ;
                        ;
                        ;
                        ;
E9EC E61F       GRAFIX  ANI     01FH    ;GET LOW FIVE
                                        ;BITS
                        BIT     6,C     ;LOW PART OF
E9EE+CB71
                                        ;COORDINATE
                        JRNZ    SETLOW  ;YES...
E9F0+200E
                        BIT     HIXFLG,B        ;IS THIS
E9F2+CB50
                                        ;THE HIGH
                                        ;PART OF X
                                        ;COORDINATE?
                        JRNZ    SETHIX  ;YES...
E9F4+2005
E9F6 32D501             STA     HIY     ;NO - SET HIGH Y
                        JR      JLOOP1  ;BACK TO
E9F9+18B2
                                        ;MAIN LOOP
                        ;
                                        ;NEXT HIGH
                                        ;COORD WILL BE
                                        ;HIGH Y.
EA2F C3ADE9             JMP     JLOOP1  ;BACK TO
                                        ;MAIN LOOP
                        ;
                        ;
                        ;
                        ;
                ;**********************************
                ;*                                *
                ;*      SUBROUTINES               *
                ;*                                *
                ;**********************************
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                ;**********************************
                ;*                                *
                ;*      COORDS                    *
                ;*                                *
                ;* ROUTINE TO CONVERT A           *
                ;* PAIR OF FIVE-BIT TEKTRONIX     *
                ;* COORDINATE FRAGMENTS           *
                ;* TO A COORDINATE IN OUR         *
                ;* OWN SYSTEM (FIVE-EIGHTHS       *
                ;* OF THE STANDARD TEKTRONIX      *
                ;* COORDINATE SYSTEM.             *
                ;*                                *
                ;**********************************
                        ;
                        ;
                        ;
                        ;ENTRY HL HAS TEK COORDINATE
                        ;  H HAS HIGH FIVE BITS,
                        ;  L HAS LOW  FIVE BITS.
                        ;RETURNS ADJUSTED COORDINATE
                        ;  IN HL.
                        ;CONSUMESHL,DE,PSW
                        ;
                        ;
                HCOORDS
EA32 3AE101             LDA     HALF    ;SET MODE
                        JR      COORD0  ;CONTINUE...
EA35+1801
                        ;
                        ;
                        ;
EA37 AF         COORDS  XRA     A       ;NOT HALF MODE
                COORD0  SLAR    L       ;MOVE THE LOWER
EA38+CB25
                        SLAR    L       ;FIVE BITS OVER
EA3A+CB25
                        SLAR    L       ;THREE POSITIONS
EA3C+CB25
                                        ;LEFT
EA60 3AE101     HSCALE  LDA     HALF    ;HALF MODE?
EA63 B7                 ORA     A       ;SET FLAGS
                        JRZ     RESCALE ;NO, JUMP
EA64+2804
EA66 C5                 PUSH    B       ;SAVE FLAG IN B
                        ;
EA67 29                 DAD     H       ;<C+1>
                        JR      RESC    ;CONTINUE...
EA68+180A
                        ;
                        ;
                        ;
                        ;
                ;**********************************
                ;*                                *
                ;*      RESCALE                   *
                ;*                                *
                ;* TURNS A COORDINATE IN OUR      *
                ;* SYSTEM BACK INTO A TEK         *
                ;* COORDINATE.                    *
                ;*                                *
                ;**********************************
                        ;
                        ; ENTRY HL HAS COORDINATE
                        ; RETURNS RESCALED COORD IN HL
                        ; CONSUMES DE,HL,F
                        ;
                        ;
                RESCALE
EA6A C5                 PUSH    B
                        ;
                        ; HL COMES IN THE FORM
                        ; <5C+4>/8
                        ;
                        ;
EA6B 29                 DAD     H       ; <5C+4>/4
EA6C 29                 DAD     H       ; <5C+4>/2
EA6D 29                 DAD     H       ; <5C+4>
EA6E 23                 INX     H       ; <5C+5>
                        ;
                        ;
                        ;
EA6F 0E05               MVI     C,5     ;DIV HL/5
EA71 CD83EA             CALL    DIV8    ;DO SO...
                        ;
                        ;
                        ;
                        ;
                        ;
                        ; NOW WE HAVE HL=<C+1>
                        ;
                        ; NOW SEPARATE INTO TWO
                        ; FIVE BIT HALVES AS
                        ; REQUIRED BY TEKTRONIX
                        ; CONVENTIONS.
                        ;
                        ;
EA74 29         RESC    DAD     H       ; << HL;
EA75 29                 DAD     H       ; << HL;
EA76 29                 DAD     H       ; << HL;
                        ;
                        SRLR    L       ; >> L;
EA77+CB3D
```

```
E9FB 32D301     SETHIX  STA     HIX     ;SET HIGH X
                        JR      JLOOP1  ;BACK TO
E9FE+18AD
                                        ;MAIN LOOP
                        ;
                        ;
                        ;
                SETLOW  BIT     5,C     ;LOW PART OF X?
EA00+CB69
                        JRZ     SETLOX  ;YES...
EA02+2807
EA04 32D401             STA     LOY     ;NO-SET LOW Y
                        SETB    HIXFLG,B ;REMEMBER THAT
EA07+CBD0
                                        ;NEXT HIGH COORD
                                        ;WILL BE HIGH X
                        JR      JLOOP1  ;BACK TO
EA09+18A2
                                        ;MAIN LOOP
                        ;
                        ;
                        ;
EA0B C5         SETLOX  PUSH    B       ;SAVE FLAG BYTE
EA0C 32D201             STA     LOX     ;SAVE LOW X
EA0F 2AD201             LHLD    LOX     ;GET BOTH PARTS
                                        ;OF X
EA12 CD37EA             CALL    COORDS  ;CALC
                                        ;THE X COORD.
                        DMOV    B,H     ;SAVE IT IN BC
EA15+44
EA16+4D
EA17 2AD401             LHLD    LOY     ;GET ALL OF Y
EA1A CD32EA             CALL    HCOORDS ;PROPER Y SCALE
                        DMOV    D,H     ;SAVE IT
EA1D+54
EA1E+5D
EA1F F1                 POP     PSW     ;GET FLAG BYTE
                        BIT     DRKVEC,A ;DARK
EA20+CB5F
                                        ;VECTOR?
                        JRZ     VECTOR  ;NO, GO DRAW
EA22+2807
                                        ;A VECTOR
EA24 CD39F8             CALL    GMOVE   ;MOVE TO NEW
                                        ;X,Y
                        RES     DRKVEC,A ;RESET DARK
EA27+CB9F
                                        ;VECTOR FLAG
                        JR      VCTRX   ;BACK TO MAIN
EA29+1803
                                        ;LOOP
                        ;
                        ;
                        ;
EA2B CD3CF8     VECTOR  CALL    GDRAW   ;DRAW THE VECTOR
EA2E 47         VCTRX   MOV     B,A     ;PUT FLAGS BACK
                                        ;IN B
```

*(Continued on page 22, column 2)*

```
                        SRLR    H       ;NOW MOVE ALL
EA3E+CB3C
                        RARR    L       ;TEN BITS
EA40+CB1D
                        SRLR    H       ;THREE
EA42+CB3C
                        RARR    L       ;POSITIONS
EA44+CB1D
                        SRLR    H       ;RIGHT
EA46+CB3C
                        RARR    L
EA48+CB1D
                        DMOV    D,H     ;COPY THE TEK
EA4A+54
EA4B+5D
                                        ;COORDINATE
                                        ;VALUE
EA4C 23                 INX     H       ;<C+1>
                        ;
EA4D B7                 ORA     A       ;HALF MODE?
                        JRNZ    OVER2   ;YES...
EA4E+200B
                        ;
EA50 29                 DAD     H       ;<2C+2>
EA51 29                 DAD     H       ;<4C+4>
EA52 19                 DAD     D       ;<5C+4>
                        SRLR    H
EA53+CB3C
                        RARR    L       ;<5C+4>/2
EA55+CB1D
                        SRLR    H
EA57+CB3C
                        RARR    L       ;<5C+4>/4
EA59+CB1D
OVER2           SRLR    H
EA5B+CB3C
                        RARR    L       ;<5C+4>/8
EA5D+CB1D
                                        ;OR <C+1>/2 IF
                                        ;HALF MODE
EA5F C9                 RET
                        ;
                        ;
                        ;
                        ;
                ;*******************************
                ;*                             *
                ;*      HSCALE                 *
                ;*                             *
                ;*******************************
                        ;
                        ; HALF MODE Y RESCALER
                        ;
                        ;ENTRY HL W/ <C+1>/2
                        ;RETURNS HL W/ RESCALED
                        ; COORDS.
                        ;
```

*(Continued on page 22, column 3)*

```
                        SRLR    L       ; >> L;
EA79+CB3D
                        SRLR    L       ; >> L;
EA7B+CB3D
                        ;
                        ; WE NOW HAVE TWO 5 BIT HALVES
                        ; IN H AND L RESPECTIVELY.
                        ;
                        ; TEKTRONIX CONVENTION REQUIRES
                        ; THAT BIT 5 BE SET ON EACH
                        ; HALF.
                        ;
                        SETB    5,H     ;SET BIT
EA7D+CBEC
                        SETB    5,L     ;SET BIT
EA7F+CBED
                        ;
                        ;
                        ;
                        ;
EA81 C1                 POP     B
EA82 C9                 RET             ; AND EXIT
                        ;
                        ;
                        ;
                        ;
                        ;
                ;*******************************
                ;*                             *
                ;*      DIV8                   *
                ;*                             *
                ;* DIVIDES HL/C.               *
                ;*                             *
                ;*******************************
                        ;
                        ;ENTRY HL,C W/ APPROPRIATE
                        ; VALUES.
                        ;RETURNS HL/C.
                        ;CONSUMES BC,DE,HL,PSW
                        ;
                        ;NOTE CY SET ON ERROR.
                        ;
EA83 AF         DIV8    XRA     A       ;ZERO ACC
EA84 47                 MOV     B,A     ;MAKE HIGH ORDER
                                        ;ZERO
EA85 91                 SUB     C       ;C=0?
EA86 3F                 CMC             ;IF SO,
EA87 D8                 RC              ;THEN ERROR...
                        ;
EA88 50                 MOV     D,B     ;ZERO DE
EA89 5A                 MOV     E,D     ;
                        ;
                        ;
                DIVLP   DSUB    B       ;SUBTRACT IT
EA8A+B7
EA8B+ED42
                        JRC     DONE    ;TOO MANY
```

*(Continued on next page)*

# TEK4010

**(Listing continued, text begins on page 8)**

```
EA8D+3803
EA8F 13                 INX     D       ;INCREMENT CNT
                        JR      DIVLP

EA90+18F8
EA92 09         DONE    DAD     B       ;ADD IT BACK
EA93 EB                 XCHG            ;ANSWER IN HL
                        ;
EA94 C9                 RET
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      CMOVE                    *
                ;*                               *
                ;* SAVES THE CURRENT CURSOR      *
                ;* POSITION ON A PUSH-UP         *
                ;* STACK AND SETS A NEW          *
                ;* CURSOR POSITION FROM INPUT    *
                ;* IN BC,DE.                     *
                ;*                               *
                ;*********************************
                        ;
                        ;
                        ; ENTRY BC,DE FORM A VALID
                        ;  X,Y ADDRESS FOR THE SYSTEM.
                        ; RETURNS BC,DE UNCHANGED
                        ; CONSUMES HL,X,SAVPTR
                        ;
                        ; NOTE CY IS SET IF THE PUSH
                        ;  UP STACK HAS OVERFLOWN.
                        ;  IF SO, NO SAVE IS MADE.
                        ;
                CMOVE   LIXD    SAVPTR  ; "STACK"
EA95+DD2A
EA97+E301
                                        ; PTR FOR
                                        ; OPERATIONS
                        ;
EA99 2A0CE6             LHLD    GCURXP  ; CURRENT X
                        STX     L,0     ; SAVE LOW ORDER
EA9C+DD7500
                        STX     H,1     ; AND HIGH ORDER
EA9F+DD7401
                        ;
EAA2 2A0EE6             LHLD    GCURYP  ; CURRENT Y
                        STX     L,2     ; SAVE LOW ORDER
EAA5+DD7502
                        STX     H,3     ; AND HIGH ORDER
EAA8+DD7403
                        ;
EAAB EB                 XCHG            ; SAVE Y CURSOR
                                        ; IN HL
EAAC 110400             LXI     D,4     ; OFFSET
EADA 23                 INX     H
EADB 5E                 MOV     E,M     ;LOW Y
EADC 23                 INX     H
EADD 56                 MOV     D,M     ;HIGH Y
                        ;
EADE C339F8             JMP     GMOVE   ;MOVE THERE
                        ;
                ;*********************************
                ;*                               *
                ;*      DECON                    *
                ;*                               *
                ;* DECIMAL OUTPUT ROUTINE        *
                ;*                               *
                ;*********************************
                        ;
                        ;
                        ;ENTRY HL W/ NUMBER TO SEND
                        ;RETURNS NUMBER SENT
                        ;CONSUMES HL,DE,PSW,IX
                        ;
                DECON   LXIX    TAB10 ;10S TABLE
EAE1+DD21
EAE3+09EB
EAE5 97         DECO0   SUB     A       ;DGT CNT=0
                        LDX     E,0     ;LOW ORDER
EAE6+DD5E00
                        LDX     D,1     ;HIGH ORDER
EAE9+DD5601
                        ;
EAEC B7         DECO1   ORA     A       ;CY=0;
                        DSBC    D       ;HL-=DE;
EAED+ED52
                        JRC     DECO2   ;PROCEED TIL CY
EAEF+3803
EAF1 3C                 INR     A       ;++A;
                        JR      DECO1   ;REPEAT
EAF2+18F8
                        ;
EAF4 19         DECO2   DAD     D       ;RESTORING ADD
EAF5 F630               ORI     30H     ;MAKE ASCII
                        PUSHIX          ;SAVE POINTER
EAF7+DDE5
EAF9 CD48F8             CALL    GCHAR   ;SEND IT
                        POPIX           ;RECOVER PTR
EAFC+DDE1
                        ;
                        INXIX           ;++IX;
EAFE+DD23
                        INXIX           ;++IX;
EB00+DD23
                        ;
                        ; DO DONE TEST...
                        ;
                        LDX     A,-2    ; DONE IF
EB02+DD7EFE
                                        ; SUBTRAHEND =1
EB05 3D                 DCR     A       ; =1?
                        JRNZ    DECO0   ; OUTER LOOP
EB06+20DD
EB08 C9                 RET             ;  LEAVE...
                        ;
EB09 1027       TAB10   DW      10000
EB0B E803               DW      1000
EB2F+5D
EB30 01DF01             LXI     B,YMAX-1 ;MAX VALUE
                        DSUB    B       ;SUBTRACT  IT
EB33+B7
EB34+ED42
                        JRC     XHUP1   ;OK
EB36+3802
                        DMOV    D,B     ;RESET TO YMAX
EB38+50
EB39+59
                XHUP1   SDED    CURSY   ;SAVE IT
EB3A+ED53
EB3C+DA01
EB3E C9                 RET             ;AND EXIT
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      XHLEFT                   *
                ;*                               *
                ;* CROSS-HAIR CURSOR LEFT        *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY NONE
                        ;RETURNS FUNCTION COMPLETE.
                        ; ON ERROR, CURSX SET TO 0.
                        ;CONSUMES DE,HL
                        ;
                        ;
                XHLEFT
                        LOAD    CURSX   ;GET VALUES
EB3F+2AD801
EB42+ED5B
EB44+D601
                        DSUB    D       ;SUB FROM HL
EB46+B7
EB47+ED52
                        JRNC    XHLFT1  ;OK...
EB49+3003
EB4B 210000             LXI     H,0     ;RESET
                        ;
EB4E 22D801     XHLFT1  SHLD    CURSX   ;UPDATE IT
EB51 C9                 RET             ;AND EXIT
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      XHRIGHT                  *
                ;*                               *
                ;* CROSS-HAIR CURSOR RIGHT       *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY NONE
                        ;RETURN FUNCTION COMPLETE.
                        ; ON ERROR, CURSX SET TO XMAX-1.
                        ;CONSUMES BC,DE,HL
                        ;
                        ;
                        ;
                XHRIGHT
                        LOAD    CURSX   ;GET VALUES
```

```
                        DADX    D       ; ADD IT
EAAF+DD19
                        PUSHIX          ; SAVE IT
EAB1+DDE5
EAB3 E3                 XTHL            ; GET IT HL
                                        ; SAVING Y ADDR
EAB4 112502             LXI     D,STKND ; END OF STACK
EAB7 B7                 ORA     A       ; CY=0;
                        DSBC    D       ; HL-=DE;
EAB8+ED52
EABA D1                 POP     D       ; RECOVER Y ADDR
EABB 3F                 CMC             ; CY<- -CY;
EABC D8                 RC              ; ERROR EXIT
                        SIXD    SAVPTR  ; UPDATE PTR
EABD+DD22
EABF+E301
                        ;
EAC1 C339F8             JMP     GMOVE   ; MOVE TO NEW
                                        ; ADDR.
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      UMOVE                    *
                ;*                               *
                ;* UNSAVES LAST CURSOR           *
                ;* POSITION AND GMOVES THERE.    *
                ;*                               *
                ;*********************************
                        ;
                        ;
                        ; ENTRY NONE
                        ; RETURNS BC=CURRENT X
                        ;  DE=CURRENT Y.
                        ; CONSUMES BC,DE,HL
                        ;
                        ; NOTE CY IS RETURNED SET IF
                        ;  UNDERFLOW OF STACK OCCURRED.
                        ;
EAC4 2AE301     UMOVE   LHLD    SAVPTR  ;STACK
EAC7 11FCFF             LXI     D,-4    ;OFFSET
EACA 19                 DAD     D       ;SUBTRACT IT
EACB E5                 PUSH    H       ;SAVE VALUE
EACC 11E501             LXI     D,STKST ;START OF STACK
EACF B7                 ORA     A       ;CY =0;
                        DSBC    D       ;SUBTRACT DE
EAD0+ED52
EAD2 E1                 POP     H       ;CLEAR UP STACK
EAD3 D8                 RC              ;ERROR EXIT...
                        ;
                        ; HL NOW HAS CORRECT ADDR
                        ;
EAD4 22E301             SHLD    SAVPTR  ;SAVE IT
EAD7 4E                 MOV     C,M     ;LOW X
EAD8 23                 INX     H
EAD9 46                 MOV     B,M     ;HIGH X
```

*(Continued on page 24, column 2)*

```
EB0D 6400               DW      100
EB0F 0A00               DW      10
EB11 0100               DW      1
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      XHDOWN                   *
                ;*                               *
                ;* CROSS-HAIR CURSOR DOWN.       *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY NONE
                        ;RETURNS FUNCTION COMPLETE.
                        ; ON ERROR, CURSY SET TO 0 .
                        ;CONSUMES DE,HL,FLAGS
                        ;
                XHDOWN  LOAD    CURSY   ;GET VALUES
EB13+2ADA01
EB16+ED5B
EB18+D601
                        DSUB    D       ;SUB OFFSET
EB1A+B7
EB1B+ED52
                        JRNC    XHDN1   ;NO ERROR
EB1D+3003
EB1F 210000             LXI     H,0     ;RESET
EB22 22DA01     XHDN1   SHLD    CURSY   ;UPDATE
EB25 C9                 RET             ;AND EXIT
                        ;
                ;*********************************
                ;*                               *
                ;*      XHUP                     *
                ;*                               *
                ;* CROSS-HAIR CURSOR UP          *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY NONE
                        ;RETURNS FUNCTION COMPLETE.
                        ; ON ERROR, CURSY SET TO YMAX-1.
                        ;CONSUMES BC,DE,HL
                        ;
                        ;
                        ;
                XHUP    LOAD    CURSY   ;GET VALUES
EB26+2ADA01
EB29+ED5B
EB2B+D601
EB2D 19                 DAD     D       ;ADD THEM
                        DMOV    D,H     ;SAVE EXTRA COPY
EB2E+54
```

*(Continued on page 24, column 3)*

```
EB52+2AD801
EB55+ED5B
EB57+D601
                        ;
EB59 19                 DAD     D       ;ADD THEM
                        ;
                        DMOV    D,H     ;SAVE XTRA COPY
EB5A+54
EB5B+5D
                        ;
EB5C 017F02             LXI     B,XMAX-1 ;MAX VALUE
                        DSUB    B       ;SUBTRACT IT
EB5F+B7
EB60+ED42
                        JRC     XHRGHT1 ; OK
EB62+3802
                        DMOV    D,B     ;RESET
EB64+50
EB65+59
                XHRGHT1
                        SDED    CURSX   ;UPDATE
EB66+ED53
EB68+D801
EB6A C9                 RET
                        ;
                ;*********************************
                ;*                               *
                ;*      XHRETN                   *
                ;*                               *
                ;* SET CURSX=MARGIN.             *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY NONE
                        ;RETURNS CURSX=MARGIN.
                        ;CONSUMES HL.
                        ;
                        ;
                        ;
EB6B 211C00     XHRETN  LXI     H,MARGIN ;LINE POSITION
EB6E 22D801             SHLD    CURSX   ;UPDATE IT
EB71 C9                 RET
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      BCHARS                   *
                ;*                               *
                ;* MAKE BIG CHARS FOR            *
                ;* DISPLAY.                      *
                ;*                               *
                ;*********************************
                        ;
                        ; ENTRY NONE.
                        ; RETURNS FUNCTION COMPLETE
                        ; CONSUMES HL
```

*(Continued on next page)*

# TEK4010

**(Listing continued, text begins on page 8)**

```
                        ;
EB72 2110E6     BCHARS  LXI     H,GCHSIZ ;POINT TO
                                         ;CHAR SIZE VARS
                                         ;DOUBLE CHR SIZE
                        SLAR    M
EB75+CB26
                        ;
EB77 23                 INX     H       ;++HL;
EB78 23                 INX     H       ;++HL;
                        SLAR    M       ;DOUBLE
EB79+CB26
                                         ;HORIZONTAL
                                         ;SPACING
EB7B 23                 INX     H       ;++HL;
EB7C 23                 INX     H       ;++HL;
                        SLAR    M       ;DOUBLE VERTICAL
EB7D+CB26
                                         ;SPACING
                        ;
EB7F C9                 RET             ;AND EXIT
                        ;
                        ;
                        IF      NOT USED
                        ;
                ;*********************************
                ;*                               *
                ;*      LCHARS                   *
                ;*                               *
                ;* MAKES LITTLE CHARS ON         *
                ;* DISPLAY.                      *
                ;*                               *
                ;*********************************


                        ;
                        ;ENTRY NONE
                        ;RETURNS FUNCTION COMPLETE
                        ;CONSUMES HL
                        ;
                LCHARS  LXI     H,GCHSIZ ;POINT TO
                                         ;CHAR SIZE VARS
                        SRLR    M       ;HALVE CHR SIZE
                        INX     H       ;++HL;
                        INX     H       ;++HL;
                        SRLR    M       ;HALVE
                                         ;HORIZONTAL
                                         ;SPACING
                        INX     H       ;++HL;
                        INX     H       ;++HL;
                        SRLR    M       ;HALVE VERTICAL
                                         ;SPACING
                        ;
                        RET             ;AND EXIT
                        ;
                        ENDIF
                        ;
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      RESPLT                   *
                ;*                               *
                ;* REMOVE CROSS-HAIR CURSOR      *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY NONE.
                        ;RETURNS CURSOR REMOVED.
                        ;CONSUMES PSW,BC,DE,HL
                        ;
                        ;
                        ;
EB96 3E01       RESPLT  MVI     A,1     ;NON-ZERO RESETS
                        ;
                        ;(FALL THROUGH TO CPLOTR)
                        ;
                ;*********************************
                ;*                               *
                ;*      CPLOTR                   *
                ;*                               *
                ;* PLACES OR REMOVES THE         *
                ;* CROSS-HAIR CURSOR.            *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY A= NON -ZERO FOR
                        ; RESET, ZERO FOR SET.
                        ;RETURNS CURSOR REMOVED OR
                        ;PLACED.
                        ;CONSUMES PSW,BC,DE,HL,IX
                        ;
                        ;
                CPLOTR  LXIX    CTABL   ;DATA UNDER CURS
EB98+DD21
EB9A+2502
                        LDY     C,0     ;GET LOW ORDER
EB9C+FD4E00
                        LDY     B,1     ;GET HIGH ORDER
EB9F+FD4601
                        LDY     E,2     ;GET LOW ORDER
EBA2+FD5E02
                        LDY     D,3     ;GET HIGH ORDER
EBA5+FD5603
EBA8 EB                 XCHG            ;HL HAS Y VALUE
EBA9 11E7FF             LXI     D,-(CCNT/2) ;OFFSET
EBAC 19                 DAD     D       ;ADD IT
EBAD EB                 XCHG            ;HL HAS CONST,
                                        ; DE HAS CURSY-
                                        ; CONST.
EBAE 09                 DAD     B       ;SAME FOR BC
                        DMOV    B,H     ;AND PLACE IN BC
EBAF+44
EBB0+4D
EBB1 2E33               MVI     L,CCNT  ;COUNT
                        ;
                        ;
EBB3 C5         CLOOP   PUSH    B       ;SAVE CURRENT
                        LDY     C,0     ;GET LOW ORDER
EBB4+FD4E00
                        LDY     B,1     ;GET HIGH ORDER
EBB7+DD23
                                        ;TABLE.
EBB9 C1                 POP     B       ;RECOVER PTR
EBBA F1                 POP     PSW     ;UNSAVE VALUE
EBBB E1                 POP     H       ;UNSAVE CTR
                        ;
EBBC C9                 RET             ;AND EXIT
                        ;
EBBD CD09F8     SETON   CALL    XYCHK   ;GET POSITION
                                        ;AND MASK
                        JRC     EXITX   ;ILLEGAL POINT
EBF0+38F5
EBF2 4F                 MOV     C,A     ;SAVE EXTRA
                                        ;COPY
EBF3 A6                 ANA     M       ;MAKE VALID SAVE
                                        ;MASK
                        STX     A,0     ;SAVE IT
EBF4+DD7700
EBF7 79                 MOV     A,C     ;RECOVER MASK
EBF8 B6                 ORA     M       ;OR IT IN
EBF9 77                 MOV     M,A     ;SET BIT.
                        JR      EXITX   ;AND EXIT
EBFA+18EB
                        ;
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      UPDATC                   *
                ;*                               *
                ;* UPDATE CROSS-HAIR CURSOR      *
                ;*                               *
                ;*********************************
                        ;
                        ;
EBFC C5         UPDATC  PUSH    B       ;SAVE FLAG IN B
                        ;
                        ;
                        ;
EBFD 2AD801             LHLD    CURSX   ;CHECK CURSX
                        LBCD    PCURSX  ;PREV VALUE
EC00+ED4B
EC02+DC01
                        LDED    PCURSY  ;PREV VALUE
EC04+ED5B
EC06+DE01
                        DSUB    B       ;SAME?
EC08+B7
EC09+ED42
                        JRNZ    UPDATE  ;DO UPDATE
EC0B+200A
EC0D 2ADA01             LHLD    CURSY   ;GET CURSY VAL
                        DSUB    D       ;SAME?
EC10+B7
EC11+ED52
                        JRNZ    UPDATE  ;
EC13+2002
                        ;
                        ; NO NEED TO UPDATE
                        ;
EC15 C1         UPEXIT  POP     B       ;RECOVER FLAG
EC16 C9                 RET             ;AND EXIT
```

```
                ;*********************************
                ;*                               *
                ;*      CPLOT                    *
                ;*                               *
                ;* THIS SET OF ROUTINES          *
                ;* HANDLES REMOVAL AND           *
                ;* PLACEMENT OF CROSS-HAIR       *
                ;* CURSOR.                       *
                ;*                               *
                ;*********************************
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      CON                      *
                ;*                               *
                ;* TURN CURSOR ON.               *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY NONE.
                        ;RETURNS FUNCTION COMPLETE.
                        ;CONSUMES PSW,IX,IY,DE,HL
                        ;
                        ;
EB80 CD2DEC     CON     CALL    DSPSET  ;NO UPDATE OF
                                        ;CURSOR POS.
EB83 3EFF               MVI     A,0FFH  ;ON CODE
                        JR      CPLOT
EB85+1801
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      COFF                     *
                ;*                               *
                ;* TURN CURSOR OFF.              *
                ;*                               *
                ;*********************************
                        ;
                        ;SAME AS CON BUT TURNS
                        ;CURSOR OFF INSTEAD OF ON.
                        ;
EB87 AF         COFF    XRA     A       ;ZERO ACC
                CPLOT
EB88 32E001             STA     XMODE   ;TURN IT OFF
EB8B 2F                 CMA             ;MEANS RESET
EB8C C5                 PUSH    B
                        LXIY    CURSX   ;POINT TO 4
EB8D+FD21
EB8F+D801
                                        ;BYTES
                        ;
EB91 CD98EB             CALL    CPLOTR  ;DO WORK
EB94 C1                 POP     B
EB95 C9                 RET
                        ;
```

*(Continued on page 26, column 2)*

```
EBB7+FD4601
EBBA CDD6EB             CALL    @SETCLR ;PLOT OR RESET
EBBD C1                 POP     B       ;RECOVER
EBBE 67                 MOV     H,A     ;SAVE ACC VALUE
EBBF 7D                 MOV     A,L     ;GET COUNT
EBC0 FE1A               CPI     (CCNT/2)+1 ;CRITICAL
                                        ;VALUE
EBC2 7C                 MOV     A,H     ;RECOVER A VALUE
                        JRZ     CLOP1   ;YES, JUMP...
EBC3+280B
EBC5 D5                 PUSH    D       ;SAVE CURRENT
                        LDY     E,2     ;GET LOW ORDER
EBC6+FD5E02
                        LDY     D,3     ;GET HIGH ORDER
EBC9+FD5603
                        ;
EBCC CDD6EB             CALL    @SETCLR ;PLOT OR RESET
EBCF D1                 POP     D       ;RECOVER CURRENT
                        ;
EBD0 13         CLOP1   INX     D       ;POINT TO NEXT
EBD1 03                 INX     B       ;
EBD2 2D                 DCR     L       ;DECREMENT CNT
                        JRNZ    CLOOP   ;REPEAT...
EBD3+20DE
                        ;
                        ;
EBD5 C9                 RET             ;AND EXIT
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      @SETCLR                  *
                ;*                               *
                ;*********************************
                        ;
                @SETCLR
EBD6 E5                 PUSH    H       ;SAVE CTR
EBD7 F5                 PUSH    PSW     ;SAVE THIS VALUE
EBD8 C5                 PUSH    B       ;WE NEED THIS
                                        ;FOR TEMP STOR.
EBD9 B7                 ORA     A       ;SET FLAGS
                SETOF
EBDA CD09F8             CALL    XYCHK   ;GET POSITION
                                        ;AND MASK
                        JRC     EXITX   ;ILLEGAL POINT
EBDD+3808
EBDF 2F                 CMA             ;COMPLEMENT
                                        ;SO WE CAN
                                        ;AND OUT THE
                                        ;CURRENT STATUS
EBE0 A6                 ANA     M       ;AND IT
EBE1 4F                 MOV     C,A     ;STORE IT TEMP.
                        LDX     A,0     ;GET OLD MASK
EBE2+DD7E00
EBE5 B1                 ORA     C       ;OR IT IN
EBE6 77                 MOV     M,A     ;SAVE IT
                EXITX   INXIX           ;NEXT POS IN
```

*(Continued on page 26, column 3)*

```
                        ;
                UPDATE
                        ;
                        LXIY    PCURSX  ;USE OLD VALUES
EC17+FD21
EC19+DC01
EC1B CD96EB             CALL    RESPLT  ;RESET WITH OLD
                                        ;VALUES
EC1E CD80EB             CALL    CON     ;REPLOT AT NEW
                                        ;POSITION.
                        ;
EC21 3AE201             LDA     DSPFLG  ;WRITE X,Y?
EC24 B7                 ORA     A       ;SET FLAGS
EC25 C488EC             CNZ     WRITEXY ;YES...
                        JR      UPEXIT  ;EXIT
EC28+18EB
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      XERASE                   *
                ;*                               *
                ;* DOES A CALL TO GERASE AND     *
                ;* DSPSET.                       *
                ;*                               *
                ;*********************************
                        ;
EC2A CD60F8     XERASE  CALL    GERASE  ;ERASE SCRN
                        ;
                        ; (FALL THROUGH TO DSPSET)
                        ;
                ;*********************************
                ;*                               *
                ;*      DSPSET                   *
                ;*                               *
                ;* SET SO THAT NO UPDATE         *
                ;* CONDITION IS SET.             *
                ;*                               *
                ;*********************************
                        ;
                        ;
EC2D C5         DSPSET  PUSH    B       ;SOME CALLERS
                                        ;NEED THIS SAVED
EC2E 21D801             LXI     H,CURSX ;SOURCE
EC31 11DC01             LXI     D,PCURSX        ;DEST
EC34 010400             LXI     B,4     ;COUNT
                        LDIR            ;UPDATE
EC37+EDB0
EC39 C1                 POP     B       ;UNSAVE PTR
EC3A C9                 RET
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
```

*(Continued on next page)*

# TEK4010

**(Listing continued, text begins on page 8)**

```
                ;**********************************
                ;*                                *
                ;*      SENDER                    *
                ;*                                *
                ;* SENDS TERMINAL STAT.           *
                ;* BYTE, XY CURS POS OF           *
                ;* GRAPHIC CURSOR.                *
                ;*                                *
                ;**********************************
                        ;
                        ;
                        ;ENTRY NONE.
                        ;RETURNS FUNCTION COMPLETE.
                        ;CONSUMES PSW,BC,HL
                        ;
                        ;
                        ;
EC3B+D9         SENDER  EXX             ;GET STAT IN B
EC3C 78                 MOV     A,B     ;STAT BYTE
                        EXX             ;RECOVER ALT SET
EC3D+D9
EC3E 47                 MOV     B,A     ;AND SAVE IN B
                        ;
EC3F 3EB5               MVI     A,10110101B ;STAT CODE
                        ;
                        ; STATUS BYTE IS MADE UP
                        ; AS FOLLOWS
                        ;
                        ; BIT 7 ON
                        ; BIT 6 OFF
                        ; BIT 5 ON
                        ; BIT 4 ON (NO HARD COPY UNIT)
                        ; BIT 3 OFF (TERM NOT READY
                        ;  TO SEND VECTORS)
                        ; BIT 2 ON (ALPHA MODE)
                        ; BIT 1 OFF (MARGIN 0 )
                        ; BIT 0 ON (NO AUX. DEVICE)
                        ;
                        ;
                        BIT     GMODE,B ;GRAPHICS MODE?
EC41+CB48
                        JRZ     SEND1   ;NO...
EC43+2808
                        RES     2,A     ;SET FOR GRAPH
EC45+CB97
                                        ;MODE.
                        ;
                        BIT     DRKVEC,B ;DARK VECTOR
EC47+CB58
                                        ;MODE?
                        JRNZ    SEND1   ;YES...
EC49+2002
                        SETB    3,A     ;
EC4B+CBDF
                        ;
EC4D 57         SEND1   MOV     D,A     ;SAVE CHAR
                        ENDIF
                        ;
EC87 C9                 RET             ;AND EXIT
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                ;**********************************
                ;*                                *
                ;*      WRITEXY                   *
                ;*                                *
                ;**********************************
                        ;
                        ;
                WRITEXY
EC88 CDA8ED             CALL    BLOCK   ;EMPTY SPACE
                        ;
EC8B 010000             LXI     B,0     ;X=0
                        DMOV    D,B     ;Y=0
EC8E+50
EC8F+59
                        ;
                WXY0    LXIX    CURSX   ;POINT TO OUTPUT
EC90+DD21
EC92+D801
                                        ;VALUES
                WXY1    PUSHIX          ;SAVE PTR
EC94+DDE5
EC96 CD95EA             CALL    CMOVE   ;MOVE CURSOR
                        POPIX           ;UNSAVE IT
EC99+DDE1
                        ;
                        ;
                WXY2    LDX     L,0     ;GET LOW ORDER
EC9B+DD6E00
                        LDX     H,1     ;GET HIGH ORDER
EC9E+DD6601
ECA1 CDB0EC             CALL    WRITE   ;WRITE IT
                        LDX     L,2     ;GET LOW ORDER
ECA4+DD6E02
                        LDX     H,3     ;GET HIGH ORDER
ECA7+DD6603
                        ;
ECAA CDB0EC             CALL    WRITE   ;WRITE IT
ECAD C3C4EA             JMP     UMOVE   ;UN MOVE CURSOR
                        ;
                        ;
                ;**********************************
                ;*                                *
                ;*      WRITE                     *
                ;*                                *
                ;**********************************
                        ;
                WRITE   PUSHIX          ;SAVE PTR
ECB0+DDE5
ECB2 CDE1EA             CALL    DECON   ;SEND #
ECB5 3E20               MVI     A,BLANK ;DELIMITER
ECB7 CD48F8             CALL    GCHAR   ;SEND IT
                        POPIX           ;RECOVER PTR
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                ;**********************************
                ;*                                *
                ;*      RESET                     *
                ;*                                *
                ;**********************************
                        ;
                        ;
ECDB 213200     RESET:  LXI     H,SOFF  ;OFFSET INCR
ECDE 22D601             SHLD    XOFF    ;STORE IT
                        ;
ECE1 C9                 RET
                        ;
                ;**********************************
                ;*                                *
                ;*      XHVERT                    *
                ;*                                *
                ;* RESET VERTICAL SCALING         *
                ;*                                *
                ;**********************************
                        ;
                        ;ENTRY: NONE.
                        ;RETURNS: FUNCTION COMPLETE
                        ; ON INPUT OF H OR F
                        ; WHERE H= HALF AND F=FULL
                        ; OR MAX.
                        ;
                XHVERT:
ECE2 CDA8ED             CALL    BLOCK   ;EMPTY BOTT
                                        ;LEFT CORNER
ECE5 010000             LXI     B,0     ;ADDRESS CORNER
                        DMOV    D,B     ;X=Y=0
ECE8+50
ECE9+59
ECEA CD95EA             CALL    CMOVE   ;MOVE THERE
ECED 210FED             LXI     H,MESG  ;SEND WARNING
                                        ;MESSAGE
ECF0 CD42F8             CALL    GCHSTR  ;SEND NULL
                                        ;TERMINATED
                                        ;STRING
ECF3 CDBDEC             CALL    CI      ;GET A CHAR
ECF6 FE48               CPI     'H'     ;HALF-MODE
                                        ;REQUEST?
                        JRZ     VERTHF  ;HALF MODE
ECF8+280E
ECFA FE46               CPI     'F'     ;FULL MODE
                        JRNZ    VERTEX  ;EXIT...
ECFC+2004
                        ;
                        ; SET TO FULL MODE
                        ;
                        ;
ECFE AF                 XRA     A
ECFF 32E101             STA     HALF    ;RESET FLAG
                VERTEX:
ED02 CDA8ED             CALL    BLOCK   ;ERASE MESSAGE
ED05 C3C4EA             JMP     UMOVE   ;RESET CURSOR
```

```
EC4E 3AC901             LDA     SIODAT  ;GET PORT #
EC51 4F                 MOV     C,A     ;STUFF IT
EC52 7A                 MOV     A,D     ;RECOVER CHAR
EC53 CDCBEC             CALL    SCHAR   ;SEND IT
                        ;
                        ;
                        LXIX    GCURXP  ;POINT TO X,Y
EC56+DD21
EC58+0CE6
                        ;
                        ;
                        ; NOW SEND XY POSITION
                        ;
                        JR      XHSND1  ;SKIP LXI
EC5A+1804
                        ;
                        ;
                        ;
                XHSEND  LXIX    CURSX   ;CURS ADDR PTR
EC5C+DD21
EC5E+D801
                XHSND1  LDX     L,0     ;GET LOW ORDER
EC60+DD6E00
                        LDX     H,1     ;GET HIGH ORDER
EC63+DD6601
EC66 CD6AEA             CALL    RESCALE ;RESCALE #
EC69 7C                 MOV     A,H     ;HIGH ORDER
EC6A CDCBEC             CALL    SCHAR   ;SEND CHAR
EC6D 7D                 MOV     A,L     ;LOW ORDER
EC6E CDCBEC             CALL    SCHAR   ;SEND CHAR
                        LDX     L,2     ;GET LOW ORDER
EC71+DD6E02
                        LDX     H,3     ;GET HIGH ORDER
EC74+DD6603
                        ;
EC77 CD60EA             CALL    HSCALE  ;HALF SCALING
                        ;
EC7A 7C                 MOV     A,H     ;HIGH ORDER
EC7B CDCBEC             CALL    SCHAR   ;SEND CHAR
EC7E 7D                 MOV     A,L     ;LOW ORDER
EC7F CDCBEC             CALL    SCHAR   ;SEND CHAR
                        ;
                        ; NOW SEND PROPER END CHARS
                        ;
                        IF      STRP1
                        ;
                        ;
                        IF      STRP2
EC82 3E0D               MVI     A,CR    ;CARRIAGE RET
EC84 CDCBEC             CALL    SCHAR   ;SEND CHAR
                        ENDIF
                        ;
                        IF      STRP3
                        MVI     A,EOT   ;CONTROL CHAR
                        CALL    SCHAR   ;SEND CHAR
                        ENDIF
                        ;
```

*(Continued on page 28, column 2)*

```
ECBA+DDE1
ECBC C9                 RET             ;AND EXIT
                        ;
                        ;
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      CI                       *
                ;*                               *
                ;*********************************
                        ;
                        ;
ECBD CD18F8     CI      CALL    KBCHEK  ;GET CHAR
                        JRC     CI      ;WAIT
ECC0+38FB
ECC2 C32AF8             JMP     KBREAD  ;GET CHAR
                                        ;AND EXIT
                        ;
                ;*********************************
                ;*                               *
                ;*      CO                       *
                ;*                               *
                ;*********************************
                        ;
ECC5 CD33F8     CO      CALL    SUBENT  ;SAVE REGS
ECC8 C348F8             JMP     GCHAR   ;SEND IT
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      SCHAR                    *
                ;*                               *
                ;*********************************
                        ;
                        ;
ECCB F5         SCHAR:  PUSH    PSW     ;SAVE INPUT TO
                                        ;ROUTINE
ECCC 3AC801             LDA     SIOCTL  ;GET CTRL PORT
ECCF 4F                 MOV     C,A     ;PLACE WHERE
ECCF 4F                 MOV     C,A     ;PLACE WHERE
                                        ;NEEDED
                SCHAR1: INP     A       ;GET STAT.
ECD0+ED78
ECD2 E601               ANI     SIOXMT  ;IS CHAR BEING
                                        ;SENT?
                        JRZ     SCHAR1  ;YES...
ECD4+28FA
ECD6 F1                 POP     PSW     ;GET SEND CHAR
                        ;
ECD7 0C                 INR     C       ;POINT TO
                                        ;(SIODAT)
                        OUTP    A       ;SEND IT
ECD8+ED79
ECDA C9                 RET             ;AND EXIT
```

*(Continued on page 28, column 3)*

```
                        ;
ED08 3E01       VERTHF: MVI     A,1     ;SET FLAG
ED0A 32E101             STA     HALF    ;SET IT
                        JR      VERTEX  ;AND EXIT...
ED0D+18F3
                        ;
                        ;
                MESG:   DBZ     'V-SCALING'
ED0F+562D534341
                        ;
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      XHOME                    *
                ;*                               *
                ;* HOME CROSS-HAIR CURSOR        *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY: NONE.
                        ;RETURNS: CURSX,CURSY SET TO
                        ; XMID, YMID RESPECTIVELY.
                        ;CONSUMES: HL.
                        ;
                        ;
                        ;
ED19 214001     XHOME:  LXI     H,XMID  ;CENTER
                                        ;OF SCREEN
ED1C 22D801             SHLD    CURSX   ;UPDATE IT
ED1F 21F000             LXI     H,YMID  ;Y CENTER
ED22 22DA01             SHLD    CURSY   ;UPDATE IT
ED25 C9                 RET             ;AND EXIT
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      XHADDR                   *
                ;*                               *
                ;* ADDRESS CURSOR ABSOLUTE       *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY: NONE
                        ;RETURNS: IF ENTRY IS VALID,
                        ; CURSOR IS SET TO NEW LOC.
                        ; IF NOT, NO CHANGE OCCURS.
                        ;
                        ; FORM OF INPUT:
                        ;
                        ; TWO THREE DIGIT DECIMAL
                        ; NUMBERS SEPARATED BY ANY
                        ; CHAR.  LEADING ZEROES MUST
                        ; BE SUPPLIED AS NEEDED.
                        ;
```

*(Continued on next page)*

# TEK4010

(Listing continued, text begins on page 8)

```
                        ;
                        ;
ED26 118002     XHADDR: LXI     D,XMAX  ;MAX X+1
ED29 CD68ED             CALL    FDEC    ;GET VALUE
                        JRC     XHAD1   ;DON'T CHANGE X
ED2C+3803
ED2E 22D801             SHLD    CURSX   ;UPDATE X
ED31 CDBDEC     XHAD1:  CALL    CI      ;GET DELIM CHAR
ED34 11E001             LXI     D,YMAX  ;MAX Y+1
ED37 CD68ED             CALL    FDEC    ;FAST INPUT
ED3A D8                 RC              ;ERROR
ED3B 22DA01             SHLD    CURSY   ;UPDATE Y
                        ;
ED3E C9                 RET
                        ;
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      OFFSET                   *
                ;*                               *
                ;*********************************
                        ;
                        ;
ED3F CDBDEC     OFFSET: CALL    CI      ;GET CHAR
                        RES     7,A     ;DUMP PARITY
ED42+CBBF
ED44 CD8EED             CALL    DCHK    ;VALID DEC.
                                        ;DIGIT?
ED47 D8                 RC              ;NO...
ED48 B7                 ORA     A       ;ZERO?
                        JRNZ    OFFS2   ;CONTINUE
ED49+2002
                        ;
                        ; ZERO ACTS LIKE A 10
                        ;
                        ;
ED4B 3E0A               MVI     A,10    ;MAKE VALID
                                        ;VALUE
ED4D 2AD601     OFFS2:  LHLD    XOFF    ;OFFSET VALUE
ED50 4F                 MOV     C,A     ;DIVISOR
ED51 CD83EA             CALL    DIV8    ;DIVIDE IT
ED54 7D                 MOV     A,L     ;MAKE SURE NOT
                                        ;ZERO
ED55 B4                 ORA     H       ;HL=0?
                        JRNZ    OFFS3   ;NO, SO OKAY
ED56+2001
ED58 23                 INX     H       ;MAKE OFFSET 1
ED59 22D601     OFFS3:  SHLD    XOFF    ;UPDATE
ED5C C9                 RET
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      OFFSET2                  *
                ;*                               *
                ;*********************************
                        ;
                ;*********************************
                ;*                               *
                ;*      DCHK                     *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY: CHAR TO TEST IN ACC
                        ;RETURNS: ASCII BIAS STRIPPED
                        ; AND CY SET ON ERROR.
                        ;CONSUMES: PSW
                        ;
                        ;
                        ;
ED8E D630       DCHK:   SUI     30H     ;ASCII BIAS
ED90 D8                 RC              ;ERROR
ED91 FE0A               CPI     10      ;TOO BIG?
ED93 3F                 CMC             ;WRONG IF NO
                                        ;CARRY
ED94 C9                 RET
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      MUL10                    *
                ;*                               *
                ;* MULTIPLIES HL BY 10.          *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY: HL .
                        ;RETURNS: 10*HL.  CY SET ON
                        ; OVERFLOW.
                        ;CONSUMES: HL.
                        ;
                        ;
ED95 D5         MUL10:  PUSH    D       ;SAVE DE VALUE
                        DMOV    D,H     ;SAVE EXTRA COPY
ED96+54
ED97+5D
ED98 29                 DAD     H       ;2*
ED99 29                 DAD     H       ;4*
ED9A 19                 DAD     D       ;5*
ED9B 29                 DAD     H       ;10*
ED9C D1                 POP     D       ;RECOVER DE
ED9D C9                 RET             ;AND EXIT
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      NDSP                     *
                ;*                               *
                ;* TOGGLE CROSS-HAIR CURSOR      *
                ;* COORDINATE DISPLAY.           *
                ;*                               *
                ;*********************************
                        ;
                        ;ENTRY: NONE.
                ;*                               *
                ;*********************************
                        ;
EDBF CD33F8     ZCTABL: CALL    SUBENT  ;SAVE REGS
EDC2 212502             LXI     H,CTABL ;CURSOR IMAGE
EDC5 016500             LXI     B,(CCNT*2)-1 ;COUNT
EDC8 AF                 XRA     A       ;ZERO ACC
EDC9 CDCDED             CALL    FLIR    ;FILL
EDCC C9                 RET             ;EXIT
                ;*********************************
                ;*                               *
                ;*      FLIR                     *
                ;*                               *
                ;* FILL INCREMENT AND REPEAT     *
                ;* WORKS ANALOGOUSLY TO LDIR     *
                ;*                               *
                ;*********************************
                        ;
                        ; ENTRY: HL POINTS TO
                        ;  FIRST LOC TO FILL.
                        ; BC HAS COUNT, ACC HAS CHR.
                        ; RETURNS: FN COMPLETE.
                        ; CONSUMES: HL,DE,BC
                        ;
EDCD F5         FLIR:   PUSH    PSW     ;SAVE A,F
EDCE 77                 MOV     M,A
EDCF 0B                 DCX     B       ;--COUNT SINCE
                                        ;ONE MOVE HAS
                                        ;BEEN DONE
EDD0 78                 MOV     A,B     ;BC=0?
EDD1 B1                 ORA     C
                        JRNZ    FLEXIT  ;EXIT
EDD2+2005
                        DMOV    D,H     ;MAKE DEST
EDD4+54
EDD5+5D
EDD6 13                 INX     D       ;=SOURCE+1
                        LDIR            ;FILL
EDD7+EDB0
EDD9 F1         FLEXIT: POP     PSW     ;RECOV A,F
EDDA C9                 RET             ;AND EXIT..
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      PIOSET                   *
                ;*                               *
                ;* SETS UP COMMUNICATION         *
                ;* TO T.S.C.                     *
                ;*                               *
                ;*********************************
                        ;
                        ;
                PIOSET: BIT     SIOSW,A ;SELECT SIO1
EDDB+CB67
                                        ;OR SIO2
                        JRZ     SIO2
EDDD+2806
                        ;
EDDF 3E00               MVI     A,SIO1CT ;CTRL PORT
EDE1 0640               MVI     B,IN1MSK ;MASK
```

```
                        ;
                        ;
                OFFSET2:
ED5D 216400             LXI     H,SOFF*2 ;MAX XOFF
ED60 CD68ED             CALL    FDEC     ;FAST INPUT
ED63 D8                 RC               ;ERROR
ED64 22D601             SHLD    XOFF     ;UPDATE OFFSET
ED67 C9                 RET
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      FDEC                     *
                ;*                               *
                ;* FAST DEC. INPUT ROUTINE.      *
                ;*                               *
                ;*********************************
                        ;
                        ;
ED68 C5         FDEC:   PUSH    B        ;SAVE BC
ED69 0603               MVI     B,3      ;DIGIT CNT
ED6B CD70ED             CALL    FGET     ;GET #
ED6E C1                 POP     B
ED6F C9                 RET
                        ;
ED70 210000     FGET:   LXI     H,0      ;ACCUMULATOR
                        ;
                        ;
                        ;
ED73 D5         FGET1:  PUSH    D        ;MAX VALUE+1
ED74 CDBDEC             CALL    CI       ;GET CHAR
ED77 CD8EED             CALL    DCHK     ;GOOD DIGIT?
ED7A D8                 RC               ;ERROR
                        ;
ED7B CD95ED             CALL    MULTO    ;HL=10*HL;
ED7E D8                 RC               ;ERROR
ED7F 1600               MVI     D,0      ;ZERO HIGH ORDER
ED81 5F                 MOV     E,A      ;LOW ORDER
ED82 19                 DAD     D        ;ADD
                        ;
ED83 D8                 RC               ;ERROR
                        DJNZ    FGET1    ;REPEAT TIL DONE

ED84+10ED
ED86 D1                 POP     D        ;RECOVER MAX+1
ED87 EB                 XCHG             ;MAX VAL TO HL
ED88 2B                 DCX     H        ;--HL;
ED89 B7                 ORA     A        ;SET FLAGS
                        DSBC    D        ;SUB ACTUAL VAL.

ED8A+ED52
ED8C EB                 XCHG             ;ACTUAL TO HL
                        ;
                        ; CY SET ON ERROR
                        ;
                        ;
ED8D C9                 RET              ;AND EXIT
                        ;
```

*(Continued on page 30, column 2)*

```
                        ;RETURNS: FUNCTION COMPLETE.
                        ;CONSUMES: AF,DE,HL
                        ;
ED9E 21E201     NDSP:   LXI     H,DSPFLG ;POINT TO
                                         ;CONTROL BYTE
EDA1 7E                 MOV     A,M      ;FETCH IT
EDA2 2F                 CMA              ;COMPLEMENT
                                         ;OPERATION
EDA3 77                 MOV     M,A      ;UPDATE IT
EDA4 B7                 ORA     A        ;SET FLAGS
                        JRZ     BLOCK    ;OFF MODE

EDA5+2801
EDA7 C9                 RET              ;AND EXIT
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      BLOCK                    *
                ;*                               *
                ;*********************************
                        ;
                        ;
                        ;ENTRY: NONE
                        ;RETURNS: LOWER LEFT DATA BLOCK
                        ; CLEARED.
                        ;CONSUMES: AF,DE,HL
                        ;
EDA8 C5         BLOCK:  PUSH    B        ;SAVE FLAG IN B
                        ;
EDA9 111000             LXI     D,80XH   ;Y POS
EDAC 010000     B0:     LXI     B,0      ;X POS
                        ;
                                         ;ABS ADDR
EDAF CD09F8             CALL    XYCHK    ;GET
EDB2 0614       B1:     MVI     B,80XW   ;COUNT
EDB4 71         B2:     MOV     M,C      ;FILL
EDB5 23                 INX     H        ;POINT TO NEXT
                        DJNZ    B2       ;UNTIL DONE

EDB6+10FC
                        ;
EDB8 1B                 DCX     D        ;--DE;
EDB9 7B                 MOV     A,E      ;IS DE=0?
EDBA B2                 ORA     D

                        JRNZ    B0       ;NO

EDBB+20EF
                        ;
EDBD C1                 POP     B
EDBE C9                 RET              ;AND EXIT
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      ZCTABL                   *
                ;*                               *
                ;* CLEAR CTABL. ZERO IMAGE OF    *
                ;* WHAT IS UNDER CURSOR.         *
```

*(Continued on page 30, column 3)*

```
                        JR      PIOSTX

EDE3+1804
                ;
                ;
EDE5 3E04       SIO2:   MVI     A,SIO2CT ;CTRL PORT
EDE7 0680               MVI     B,IN2MSK ;MASK
                ;
                ;
EDE9 32C801     PIOSTX: STA     SIOCTL ;STORE PORT
EDEC 3C                 INR     A
EDED 32C901             STA     SIODAT ;DATA PORT
                                       ;ONE MORE THAN
                                       ;CTRL PORT

                        ;
                        ;
                        ;
EDF0 3ECF               MVI     A,MODE3 ;INIT PIO
EDF2 D353               OUT     PIOCTL
EDF4 78                 MOV     A,B ;MASK
EDF5 D353               OUT     PIOCTL
EDF7 3E42               MVI     A,SIOVEC ;INTERRUPT
                                         ;VECTOR

EDF9 D353               OUT     PIOCTL
EDFB 3EB7               MVI     A,0B7H

EDFD D353               OUT     PIOCTL
EDFF 78                 MOV     A,B      ;COMPLEMENTED
EE00 2F                 CMA              ;MASK
EE01 D353               OUT     PIOCTL
EE03 C9                 RET
                        ;
                ;*********************************
                ;*                               *
                ;*  INTERRUPT SERVICE ROUTINE    *
                ;*                               *
                ;*********************************
                ;
EE04 CD36F8     SIOISR: CALL    INTENT   ;SAVE REGS
EE07 3AC901             LDA     SIODAT   ;GET SIO DATA
EE0A 4F                 MOV     C,A      ;PORT #
                        RES     7,A      ;MASK OFF PARITY

EE0B+CBBF
EE0D 4F                 MOV     C,A      ;SAVE IT
EE0E 3ACE01             LDA     NCHIBF   ;# OF
                                         ;CHARS IN BUFFER
EE11 FEC8               CPI     BUFSIZ   ;IS BUFFER FULL?
EE13 C8                 RZ               ;YES...
EE14 3C                 INR     A        ;BUMP THE COUNT
EE15 32CE01             STA     NCHIBF   ;UPDATE IT
EE18 FE96               CPI     SNDCTS   ;SHOULD WE
                                         ;SEND A ^S?
                        JRNZ    STFCHR   ;NO...

EE1A+200F
EE1C 3AD001             LDA     AUSTPD   ;HAVE WE
EE1F B7                 ORA     A        ;ALREADY?
                        JRNZ    STFCHR   ;YES,
```

*(Continued on next page)*

# TEK4010

**(Listing continued, text begins on page 8)**

```
EE20+2009
                                        ;DON'T SEND
                                        ;ANOTHER.
EE22 2F                 CMA             ;REMEMBER THAT
                                        ;WE'VE
                                        ;REQUESTED A ^S
EE23 32D001             STA     AUSTPD
EE26 3E13               MVI     A,CTRLS ;NOW REQUEST
EE28 32CF01             STA     CTRLQS  ;A ^S BE SENT
                ;
                ;
EE2B 2ACC01     STFCHR: LHLD    NXTPUT  ;GET PTR TO
                                        ;NEXT FREE LOC.
EE2E 71                 MOV     M,C     ;SAVE IT
EE2F 23                 INX     H       ;BUMP PTR
EE30 22CC01             SHLD    NXTPUT  ;UPDATE IT
EE33 B7                 ORA     A       ;CY:=0;
EE34 11C801             LXI     D,BUFEND        ;ARE WE
                                        ;PAST END OF
                                        ;BUFFER?
                        DSBC    D       ;SUBTRACT IT
EE37+ED52
EE39 C0                 RNZ             ; EXIT
EE3A 210001             LXI     H,BUFFER
                                        ;ADJUST PTR
EE3D 22CC01             SHLD    NXTPUT  ;UPDATE IT
                        ;
                        ;
EE40 C9                 RET             ; GO TO INTEXT
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;* DISPATCH TABLE FOR            *
                ;* CONTROL CHARACTERS.           *
                ;*                               *
                ;*********************************
                        ;
                        ;
                        ;
                CTLTBL: COM     CR,SETALF
EE41+0D
EE42+9AE9
                        COM     LF,CTLCOM
EE44+0A
EE45+B6E9
                        COM     GS,SETGRF
EE47+1D
EE48+A7E9
                        COM     US,SETALF
EE4A+1F
EE4B+9AE9
                        COM     ESC,SETESC
EE4D+1B
EE4E+B0E9
                        COM     TAB,TABBLK
EE50+09
```

```
                ; AXIOM 820 ROUTINES
                ; UPDATES AS OF 11/11/80
                ;
                ;
                ;
AXIOM:  CALL    SUBENT
        LDA     GMDE    ;CHECK RES
        ORA     A       ;NON-ZERO
                        ;-> HIGH RES
        JNZ     AXIH    ;IF <>0, HIGH
        ;
        ;
        ;(FALL THROUGH TO AXIL)
        ;
        ;
        ;
        ;
;*********************************
;*                               *
;*      AXIL                     *
;*                               *
;* OUTPUT LOW DENSITY DISP       *
;* TO AXIOM 820.                 *
;*                               *
;*********************************
        ;
        ;ENTRY: NONE.
        ;RETURNS: FN COMPLETE.
        ;CONSUMES: NONE.
AXIL:
        LXI     H,0E558H ;DEFAULT
        PUSH    H        ;SAVE IN IX
        POPIX
        ;
        LXI     D,-28H ;NEG OFFSET
        MVI     C,28H    ;40 COLUMNS
        MVI     A,SO    ;START REVERSE
                         ;GRAPHICS
        CALL    AOT      ;SEND IT
        MVI     A,RS     ;
        CALL    AOT      ;SEND IT
                         ;SELECT 256
                         ;BYTE GRAPHICS
MODE:   MVI     A,FS     ;FS
        CALL    AOT      ;SEND IT
        MVI     A,240    ;240 LINES
        MOV     B,A      ;SAVE ALSO IN
                         ;B
        CALL    AOT      ;SEND IT
PRINT:  MOV     A,M      ;GET NEXT
        CALL    AOT      ;SEND
        DAD     D        ;OFFSET
        DJNZ    PRINT    ;REPEAT...
        ;
        MVI     A,LF     ;LINE FEED
        CALL    AOT      ;SEND IT
        ;
        NOP              ;WASTE A MICRO SEC
        ;
        INXIX           ;++IX
        PUSHIX           ;BACK IN H
        POP     H
        ;
```

```
        JMP     SPACE
        ;
        ;
        ;
WRTLN:  PUSH    H       ;SAVE OUTER LOOP
                        ;COUNT
        MVI     H,2     ;HIGH COUNT
WLN1:   MVI     A,FS    ;LEN INDICATOR
        CALL    AOT     ;SEND IT
        MVI     L,240   ;LEN
        MOV     A,L     ;COPY FOR AXIOM
        CALL    AOT     ;SEND IT
WRTLN1: PUSH    H       ;SAVE COUNT
        CALL    XYCHK   ;GET BYTE
        MOV     A,M     ;GET IT
        POP     H       ;RECOVER COUNT
        CALL    AOT     ;SEND IT
        INX     D       ;++Y DIRECT
        DCR     L       ;--LOW COUNT
        JRNZ    WRTLN1  ;REPEAT
        DCR     H       ;--HIGH COUNT
        JRNZ    WLN1    ;REP...
        MVI     A,LF    ;NEWLINE
        CALL    AOT     ;SEND IT
        LXI     H,8     ;OFFSET FOR X
        DAD     B       ;ADD IT
        DMOV    B,H     ;SAVE IT
        POP     H       ;RECOVER HL
DELAY:
        PUSH    B       ;SAVE BC
        PUSH    D       ;SAVE DE
        PUSH    H       ;SAVE HL
        LXI     B,10H   ;COUNT
DEL1:   DCR     C       ;--LOW;
        JRNZ    DEL1    ;CONTINUE..
        POP     H       ;RECOV HL
        POP     D       ;RECOV DE
        POP     B       ;RECOV BC
        RET             ;THEN EXIT
        ;
        ;
        ;
AXEXIT:
        XRA     A       ;ZERO ACC
        STA     CONST   ;DRAIN CHAR
        RET             ;AND EXIT
        ;
        ;
        ENDIF           ;PRINTER
                        ;AVAILABLE


EE76    END
```

**End**

```
EE51+B4E9
EE53 00                 DB      0               ;EOT
                        ;
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      TABL FOR XHLOOP          *
                ;*                               *
                ;*********************************
                        ;
                        ;
                TABL:
                        COM     'A'-40H,XHADDR  ;ADDRESS
EE54+01
EE55+26ED
                        COM     'B'-40H,OFFSET2 ;SET INC
EE57+02
EE58+5DED
                        COM     'D'-40H,XHDOWN  ;DOWN
EE5A+04
EE5B+13EB
                        COM     'H'-40H,XHOME   ;HOME
EE5D+08
EE5E+19ED
                        COM     'J'-40H,XHRETN  ;JUSTIFY
EE60+0A
EE61+6BEB
                        COM     'L'-40H,XHLEFT  ;LEFT
EE63+0C
EE64+3FEB
                        COM     'N'-40H,NDSP    ;DISPLAY
EE66+0E
EE67+9EED
                        COM     'R'-40H,XHRIGHT ;RIGHT
EE69+12
EE6A+52EB
                        COM     'U'-40H,XHUP    ;UP CURS
EE6C+15
EE6D+26EB
                        ;
                        COM     'Z'-40H,RESET   ;RESET
EE6F+1A
EE70+DBEC
                                        ; XOFF VALUE
                        COM     '/',OFFSET      ;DIV INC
EE72+2F
EE73+3FED
                        ;
EE75 00                 DB      0       ; EOT
                        ;
                        ;
                        ;
                        IF      PRNTR   ; IF AVAIL
                        ;
                        ;
```

(Continued on page 32, column 2)

```
                        CALL    CONSTAT   ;CHECK KBD
                        ORA     A         ;?
                        JNZ     AXEXIT    ;YES..
                        DCR     C         ;--C;
                        JRNZ    MODE      ;REPEAT...
                SPACE:  MVI     B,8       ;EIGHT LFS
                SPACO:  MVI     A,LF      ;LINE FEED
                        CALL    AOT       ;SEND
                        DJNZ    SPACO     ;REPEAT
                        ;
                        RET               ;EXIT...
                        ;
                        ;
                        ;
                        ;
                AOT:    OUT     APORT     ;SEND
                RDY:    IN      APORT
                        ANI     1
                        JRNZ    RDY
                RDY1:   IN      APORT
                        ANI     10H
                        JRNZ    RDY1
                RDY2:   IN      APORT
                        ANI     10H
                        JRZ     RDY2
                        ;
                        RET
                        ;
                        ;
                        ;
                        ;
                ;*********************************
                ;*                               *
                ;*      AXIH                     *
                ;*                               *
                ;* HIGH DENSITY AXIOM 820        *
                ;*                               *
                ;*********************************
                        ;
                        ;
                AXIH:   MVI     A,SO     ;REV GRAPHICS
                        CALL    AOT      ;SEND IT
                        MVI     A,GS     ;512 MODE
                        CALL    AOT      ;SEND IT
                        ;
                        LXI     H,80     ;OUTER X LOOP
                                         ;COUNT
                        MOV     B,H      ;ZERO X
                        MOV     C,B
                AXIHO:  LXI     D,0
                        ;
                AXIH1:  CALL    WRTLN    ;WRITE A LINE
                        CALL    CONSTAT  ;STOP?
                        ORA     A        ;SET FLAGS
                        JRNZ    AXEXIT   ;YES...
                        DCR     L        ;--COUNT
                        JRNZ    AXIHO    ;REP...
```

(Continued on page 32, column 3)

## Listing II

```
;*********************************************
;*********************************************
;****                                     ****
;**** SUPPRT.LIB                          ****
;**** MACRO LIBRARY                       ****
;****                                     ****
;**** BY A.SKJELLUM                       ****
;****                                     ****
;**** VERS: 11/10/80                      ****
;****                                     ****
;****                                     ****
;****                                     ****
;*********************************************
;*********************************************
;*********************************************
        ;
        ;
        ;
;*********************************
;*                               *
;*      DCMP                     *
;*                               *
;*      DOUBLE COMPARE           *
;*                               *
;*********************************
        ;
        ;
DCMP    MACRO   ?REG
        LOCAL   ENDJ
        MOV     A,L
        SUB     ?REG
        JRNZ    ENDJ
        MOV     A,H
        SBB     ?REG+1
ENDJ:
        ENDM
        ;
        ;
;*********************************
;*                               *
;*      DSUB                     *
;*                               *
;*********************************
        ;
        ;
DSUB    MACRO   ?REG
        ORA     A
        DSBC    ?REG
        ENDM
        ;
        ;
;*********************************
```

(Continued on next page)

# TEK4010

(Listing continued, text begins on page 8)

```
;*                                      *
;*       COM                            *
;*                                      *
;**************************************
        ;
        ;
COM     MACRO   ?BYTE,?ADDR
        DB      ?BYTE
        DW      ?ADDR
        ENDM
        ;
        ;
;**************************************
;*                                      *
;*       LOAD                           *
;*                                      *
;**************************************
        ;
LOAD    MACRO   ?HL,?DE
        LHLD    ?HL
        IF      NOT NUL ?DE
        LDED    ?DE
        ELSE
        LDED    xOFF
        ENDIF
        ENDM
        ;
        ;
;**************************************
;*                                      *
;*       DMOV                           *
;*                                      *
;**************************************
        ;
DMOV    MACRO   ?REG1,?REG2
        MOV     ?REG1,?REG2
        MOV     ?REG1+1,?REG2+1
        ENDM
        ;
        ;
        ;
        ;
;**************************************
;*                                      *
;*       DBZ                            *
;*                                      *
;**************************************
        ;
DBZ     MACRO   ?EXP
        DB      ?EXP,0
        ENDM
        ;
        ;
;**************************************
;*                                      *
```

```
        ;
;
INITLZ  EQU     0F800H
;GCHAR  EQU     0F803H
SETCLR  EQU     0F806H
XYCHR   EQU     0F809H
TSTPNT  EQU     0F80CH
BCHARS  EQU     0F80FH
LCHARS  EQU     0F812H
SETIM   EQU     0F815H
CONSTAT EQU     0F818H
AX20    EQU     0F81BH
CONIN   EQU     0F82AH
LOADER  EQU     0F820H
SUBENT  EQU     0F833H
INTENT  EQU     0F836H
GMOVE   EQU     0F839H
GDRAW   EQU     0F83CH
GERSLN  EQU     0F83FH
GCHSTR  EQU     0F842H
GCHSTC  EQU     0F845H
GCHAR   EQU     0F848H
GCHARX  EQU     0F848H
GSIZE   EQU     0F84EH
SETPNT  EQU     0F851H
CLRPNT  EQU     0F854H
CONIN2  EQU     0F857H
KBSTAT  EQU     0F85AH
GERASX  EQU     0F85DH
GERASE  EQU     0F860H
GHOME   EQU     0F863H
TVSLWB  EQU     0F866H
TVSLBW  EQU     0F869H
TVSHWB  EQU     0F86CH
TVSHBW  EQU     0F86FH
SIOSET  EQU     0F872H
KBSET   EQU     0F875H
GMOVEB  EQU     0F878H
GDRAWB  EQU     0F87BH
GERLNB  EQU     0F87EH
;
GSETB   EQU     0F881H
GCLRB   EQU     0F884H
GERSXB  EQU     0F887H

GSCRLB  EQU     0F88AH
GSCRUL  EQU     0F88DH
;
;
CONST   EQU     0F890H  ;INPUT BYTE CHAR
GMDE    EQU     0E698H ;RES. FLAG
GCHSIZ  EQU     0E610H
;
        ;
        ;
GCURXP  EQU     0E60CH  ;X POS
GCURYP  EQU     0E60EH  ;Y POS
        ;
        ;
        ;
; EQUATES TO ALIGN SYMBOLS WITH
; THOSE USED IN TEK.ASM
;
KBREAD  EQU     CONIN
```

```
        ;
        ;
        ;
APORT   EQU     0EH     ;AXIOM PRINTER
                        ;PORT
        ;
SWREG   EQU     78H     ;SWTCH REGISTERS
        ;
PIOCTL  EQU     53H     ;PIO CONTRL PORT
SIO1CT  EQU     0       ;SIO 1 CTRL PORT
SIO2CT  EQU     4       ;SIO 2 CTRL PORT
SIOXMT  EQU     1       ;SIO TRANSMIT
                        ;READY FLAG
SIOVEC  EQU     042H    ;SIO INTERRUPT
                        ;VECTOR
        ;
        ;
MODE3   EQU     0CFH    ;PIO MODE 3 FLAG
IN1MSK  EQU     40H     ;PIO INPUT MASK
                        ;FOR SIO 1

IN2MSK  EQU     80H     ;PIO INPUT MASK
                        ;FOR SIO 2
MSKCTL  EQU     37H     ;PIO MSK FOLLOWS
                        ;COMMAND
PIOENB  EQU     0A7H    ;PIO INTERR
                        ;ENABLE COMMAND
        ;
        ;
        ;
        ;
        ;
        ;
        ;
        ;
;
SIOSW   EQU     4       ;SWREG SIO1/SIO2
                        ;SWITCH
BWFLG   EQU     3       ;SWREG "BLACK
                        ;ON WHITE" COMM
BIGCHR  EQU     2       ;SWREG "BIG CHR"
                        ;FLAG
LRFLG   EQU     1       ;SWREG "LOW RES"
                        ;FLAG
        ;
```

**End**

## Listing III

```
E976 ACTION     E9BB ALPHA      000E APORT      0100 AUSTPO     F81B AX20
EDAC B0         EDB2 B1         EDB4 B2         0000 BC         EB72 BCHARS
0002 BIGCHR     0020 BLANK      EDA8 BLOCK      E81B BONW       0010 BOXH
0014 BOXW       01C8 BUFEND     0100 BUFFER     00C8 BUFSIZ     0003 BWFLG
0033 CONT       001A CHSUB      ECBD C1         EBB3 CLOOP      EBD0 CLOP1
F854 CLRPNT     EA95 CMOVE      EB87 COFF       ECC5 CO         F82A CONIN
F857 CONIN2     EB80 CON        F818 CONSTAT    E607 CONST      EA38 COORDO
```

```
;*      LOC                     *
;*                              *
;********************************
        ;
LOC     MACRO   ?EXP,?INCR
?EXP    EQU     @ADDR
@ADDR   SET     @ADDR+?INCR
        ENDM
        ;
        ;
;********************************
;*                              *
;* ON-OFF LOGICAL EQUATES       *
;*                              *
;********************************
        ;
ON      EQU     0FFFFH  ;LOGICAL ON
OFF     EQU     0       ;LOGICAL OFF
        ;
        ;
        ;
;********************************
;*                              *
;*      SPECIAL EQUATES         *
;*                              *
;********************************
        ;
BOXW    EQU     20      ;160 POINT WIDE
                        ;BLOCK BOX
BOXH    EQU     16      ;AND 16 POINTS
                        ;HIGH
        ;
CCNT    EQU     51      ;SIZE OF CURSOR
                        ;MUST BE ODD #
        ;
        ;
XMAX    EQU     640
YMAX    EQU     480
XMID    EQU     XMAX/2
YMID    EQU     YMAX/2
        ;
        ;
MARGIN  EQU     (CCNT/2)+3
        ;
SOFF    EQU     50      ;DEF. OFFSET
        ;
        ;
        ;
;********************************
;*                              *
;*      ROM EQUATES             *
;*                              *
;* VERSION R6 8. SYS E          *
;*                              *
;********************************
        ;
        ;
```

*(Continued on page 34, column 2)*

```
KBCHEK  EQU     CONSTAT
;
;
;
;
;
;
        ;
        ;
;********************************
;*                              *
;*      EQUATES                 *
;*                              *
;********************************
        ;
        ;
        ;
        ;
BUFSIZ  EQU     200     ;SIZE OF SIO
                        ;BUFFER (BYTES)
SNDCTS  EQU     150     ;BUFFER FULLNESS
                        ;FOR SENDING  S
SNDCTQ  EQU     50      ;BUFFER FULLNESS
                        ;FOR SENDING  Q
        ;
BLANK   EQU     20H     ;SPACE CHAR
CR      EQU     0DH     ;CARRIAGE RET
LF      EQU     0AH     ;LINEFEED
FS      EQU     1CH
GS      EQU     35Q     ;GRAPH MODE
RS      EQU     1EH
SI      EQU     15
SO      EQU     14
US      EQU     37Q     ;ALPHA MODE
ESC     EQU     33Q     ;ESCAPE
TAB     EQU     11Q     ;TAB
FF      EQU     14Q     ;FORM FEED
        ;
  IRPC ?X,ABCDEFGHIJKLMNOPQRSTUVWXYZ
CTRL&?X EQU     '&?X'-40H
        ENDM
        ;
CHSUB   EQU     26      ;CURSOR MODE
EOT     EQU     4       ;END-OF-TAPE
ENQ     EQU     5
WRU     EQU     ENQ
        ;
        ;
ESCFLG  EQU     0       ;ESC SEQ FLAG
GMODE   EQU     1       ;GRAPH MODE FLAG
HIXFLG  EQU     2       ;HIGH X FLAG
DRKVEC  EQU     3       ;DARK VECTOR FLG
        ;
        ;
        ;
        ;
        ; HARDWARE DEPENDENT
        ; EQUATES:
```

*(Continued onpage 34, column 3)*

```
EA37 COORDS     EB88 CPLOT      EB98 CPLOTR     000D CR         0225 CTABL
E986 CTLCHR     E9B6 CTLCOM     E995 CTLDSP     E989 CTLLP      EE41 CTLTBL
0001 CTRLA      0002 CTRLB      0003 CTRLC      0004 CTRLD      0005 CTRLE
0006 CTRLF      0007 CTRLG      0008 CTRLH      0009 CTRLI      000A CTRLJ
000B CTRLK      000C CTRLL      000D CTRLM      000E CTRLN      000F CTRLO
0010 CTRLP      0011 CTRLQ      01CF CTRLQS     0012 CTRLR      0013 CTRLS
0014 CTRLT      0015 CTRLU      0016 CTRLV      0017 CTRLW      0018 CTRLX
0019 CTRLY      001A CTRLZ      01D8 CURSX      01DA CURSY      ED8E DCHK
0002 DE         EAE5 DEC00      EAEC DEC01      EAF4 DEC02      EAE1 DELCN
EA83 DIV8       EA8A DIVLP      EA92 DONE       0003 DRKVEC     01E2 DSPFLG
EC2D DSPSET     0005 ENQ        0004 EOT        E823 ERASE      001B ESC
0000 ESCFLG     E9C0 ESCSEQ     E9DA ESCSND     E9E5 ESCSND2    E901 ESCSQX
EBE7 EXITX      ED68 FDEC       000C FF         ED70 FGET       E073 FGET1
EDD9 FLEXIT     EDCD FLIR       001C FS         F848 GCHAR      F848 GCHARX
E610 GCHSIZ     F845 GCHSTC     F842 GCHSTR     F884 GCLRB      E60C GCURXP
E60E GCURYP     F83C GDRAW      F87B GDRAWB     F860 GERASE     F85D GERASX
F87E GERLNB     F83F GERSLN     F887 GERSXB     E95F GETCHR     F863 GHOME
E608 GMDE       0001 GMODE      F839 GMOVE      F878 GMOVEB     E9EC GRAFIX
001D GS         F88A GSCRLB     F88D GSCROL     F881 GSETB      F84E GSIZE
01E1 HALF       EA32 HCOORDS    0002 HIXFLG     01D3 HIX        01D5 HIY
0004 HL         EA60 HSCALE     0040 IN1MSK     0080 IN2MSK     F800 INITLZ
F836 INTENT     0004 IX         0004 IY         E9AD JLOOP1     E8C7 KB0
E8C7 KB1        F818 KBCHEK     F82A KBREAD     F875 KBSET      F85A KBSTAT
F812 LCHARS     000A LF         F82D LOADER     E887 LOOPX      E888 LOOP
01D2 LOX        01D4 LOY        0001 LRFLG      001C MARGIN     ED0F MESG
01D1 MNSTPD     00CF MODE3      0037 MSKCTL     ED95 MUL10      01CE NCHIBF
ED9E NDSP       E80A NOSWRG     01CA NXTGET     01CC NXTPUT     0000 OFF
ED4D OFFS2      ED59 OFFS3      ED3F OFFSET     ED5D OFFSET2    FFFF ON
EA5B OVER2      01DC PCURSX     01DE PCURSY     0053 PIOCTL     00A7 PIOENB
EDDB PIOSET     EDE9 PIOSTX     0000 PRNTR      EA6A RESCALE    EA74 RESC
ECDB RESET      EB96 RESPLT     E800 ROM        001E RS         01E3 SAVPTR
ECCB SCHAR      ECD0 SCHAR1     EC4D SEND1      EC3B SENDER     E99A SETALF
F806 SETCLR     E9B0 SETESC     E9A7 SETGRF     E9FB SETHIX     F815 SETIM
EA00 SETLOW     EA0B SETLOX     EBDA SETOF      EBED SETON      E835 SETPIO
F851 SETPNT     000F SI         0000 SIO1CT     0004 SIO2CT     EDE5 SIO2
01C8 SIOCTL     01C9 SIODAT     EE04 SIOISR     F872 SIOSET     0004 SIOSW
0042 SIOVEC     0001 SIOXMT     E8E6 SNDCHR     0032 SNDCTQ     0096 SNDCTS
000E SO         0032 SOFF       1000 STAK       EE2B STFCHR     0225 STKNQ
01E5 STKST      FFFF STRP1      FFFF STRP2      0000 STRP3      F833 SUBENT
0078 SWREG      0009 TAB        EB09 TAB10      E9B4 TABBLK     EE54 TABL
E800 TEK        0100 TEKVAR     E8B0 TSTKB      E8D3 TSTKB0     E80C TSTKB1
E80C TSTPNT     E939 TSTRCV     F86F TVSHBW     F86C TVSHWB     F869 TVSLBW
F866 TVSLWB     EAC4 UMOVE      EBFC UPDATC     EC17 UPDATE     EC15 UPEXIT
001F US         FFFF USED       EA2E VCTRX      EA2B VECTOR     ED02 VERTEX
ED08 VERTHF     EC80 WRITE      EC88 WRITEXY    0005 WRU        EC90 WXY0
EC94 WXY1       EC9B WXY2       EC2A XERASE     ED31 XHAD1      ED26 XHADDR
EB22 XHDN1      EB13 XHDOWN     E92E XHEOT      E938 XHERE      E924 XHEND
EB3F XHLEFT     EB4E XHLFT1     E8F5 XHLOOP     E8F6 XHL        E90A XHLOOP1
E912 XHLOOP2    E919 XHLOOP3    E8FE XHLP       ED19 XHOME      E9D5 XHON
EB6B XHRETN     EB66 XHRGHT1    EB52 XHRIGHT    EC5C XHSEND     EC60 XHSND1
EB26 XHUP       EB3A XHUP1      ECE2 XHVERT     0280 XMAX       0140 XMID
01E0 XMODE      01D6 XOFF       F809 XYCHK      01E0 YMAX       00F0 YMID
EDBF ZCTABL     EBD6 @SETCLR
```

## End Listing

# DR. DOBB'S CLINIC (Text begins on page 7)

**Listing I**

```
;
; SHIFT10: divide the contents of HL by 10
;  using the method of multiplying by the binary
;  fraction closest to 0.1 decimal.
;
; Input  = positive integer in HL
;
; Output = quotient in HL
;          remainder (single BCD digit) in A
;
; Preserves all others.
;
SHIFT10:
        push    b       ; save work registers
        push    d
        push    h       ; save dividend for later
        call    shr1    ; v = u shr 1
        push    h
        call    shr1
        pop     d
        dad     d       ; v = v + v shr 1
        push    h
        call    shr4
        pop     d
        dad     d       ; v = v + v shr 4
        push    h
        mov     l,h
        mvi     h,0     ; (effectively HL shr 8)
        pop     d
        dad     d       ; v = v + v shr 8
        call    shr3    ; v = v shr 3
;
; HL = quotient, i.e. floor(input/10).  compute
; the remainder.
;
        pop     d       ; DE = input
        push    h       ; save quotient for exit
        push    h       ; save another copy
        dad     h       ; HL = 2 * quotient
        dad     h       ; HL = 4 * quotient
        pop     b       ;   (copy of quotient)
```

```
; For a different divisor from 3..255, set the
; equate DIVISR. If divisor>15, set UNDR16 false.
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
UNDR16  EQU     TRUE
DIVISR  EQU     10
;
      IF UNDR16
COUNT   EQU     13
      ELSE
COUNT   EQU     16
      ENDIF
;
DIVID:  PUSH    BC
        LD      BC,COUNT*256+DIVISR
        XOR     A
      IF UNDR16
        ADD     HL,HL
        RLA
        ADD     HL,HL
        RLA
        ADD     HL,HL
        RLA
      ENDIF
LOOP:   ADD     HL,HL
        RLA
        CP      C
        JR      C,SKIP
        SUB     C
        INC     L
SKIP:   DJNZ    LOOP
        POP     BC
        RET
```

**Listing III**

```
;
; Ashley's Unique Division Routine:
;
; INPUT:  HL contains a positive binary integer
;            less than 2560.
;
; OUTPUT: H only contains the input divided by 10
;
; ALTERS: all.
```

```
        dad     b       ; HL = 5 * quotient
        dad     h       ; HL = 10 * quotient
        xchg            ; HL=input, DE=10*quotient
        dsbc    d       ; HL = remainder
        mov     a,l     ; A = remainder
        pop     h       ; return HL = quotient
        pop     d       ; restore work registers
        pop     b
;
; apply a correction.  One time in 4 1/2 or so,
; the algorithm delivers a value over 10 and a
; quotient low by 1.  Fix it.

        cpi     10      ; erroneous result?
        rc              ; exit if not
        sui     10      ; correct remainder by 10
        inx     h       ; and quotient by 1.
        ret
;
; shift routines -- putting the shifts in-line
; with the code would cost about 30 bytes and
; save about 150 clock periods.
;
shr4:   call    shr1
;
shr3:   call    shr1
        call    shr1
;
shr1:   ora     a       ; clear carry
        rrar    h       ; Zilog mnemonic: "RR H"
        rrar    l
        ret
```

**Listing II**

```
;
; DIVID: a faster divide by 10 (or any small
;        integer), after code by W. Farmer.
;
; Input:  HL = a positive integer in 0..32767
; Output: HL = quotient(HL/10)
;          A = remainder
; Preserves: BC, DE
;
```

*(Continued on top of page 36)*

```
;
HLD10:  mov     b,h
        mov     c,l
        call    HX25 ; HL = 25*HL
        ora     a
        mov     a,b
        rar
        mov     b,a
        mov     a,c
        rar
        mov     c,a
        dad     b    ; HL = 25*HL + HL/2
        mov     e,h
        mvi     d,0
; DE = (25.5*HL)/256, or about HL/10
        dad     d    ; HL = 25*HL + HL/2 + HL/10
        inx     h    ; (after fudge)
        ret          ; H = HL/10 when HL<2560
HX25:   call    HX5
HX5:    mov     d,h
        mov     e,l
        dad     h
        dad     h
        dad     d
        ret
```

**Listing IV**

```
; macro to build one bit of decimal digit in A
; register, assuming A already has the bit set.
; HL = remaining magnitude value.
;
buildb  macro   bit,digitweight
        local   done ; unique label each time
; make B the weight of this bit of the result:
        lxi     b,-((1 shl bit)*digitweight)
        dad     b    ; subtract from value
        jc      done ; value >= weight, leave bit
; remaining value < weight of result bit so
; restore value and clear result bit.
        lxi     b,(1 shl bit)*digitweight
        dad     b
        ani     not(1 shl bit)
done:
```

*(Continued on next page)*

# DR. DOBB'S CLINIC

(Listing continued, text begins on page 7)

```
        endm
;
;Subroutine to convert 16-bit signed integers
;to ASCII.  This version produces leading zeros.
;
;Once the sign is generated and the positive
;magnitude is calculated, the algorithm is to
;check if each bit of the possible output digits
;deserves to be set.  If so the magnitude is
;decreased by the proper amount, the next lower
;bit of the result is checked, and so on.
;
; HL = signed integer
; DE-> six bytes to receive blank/minus, digits
; preserves HL, BC, increments DE.
;
cvtdecimal:
        push    h
        push    b
        mov     a,h
        ora     a
        mvi     a,' '       ; assume positive sign
        jp      cvtdec2 ; if positive
; negative, calculate magnitude, set up minus
        mov     a,h
        cma
        mov     h,a
        mov     a,l
        cma
        mov     l,a
        inx     h
        mvi     a,'-'
cvtdec2:
        stax    d       ; output the sign
        inx     d
; now convert high digit of magnitude to ASCII
; magnitude < 32768, so digit is < 3
        mvi     a,33h   ; ASCII 3
        buildb  1,10000 ; adjust bit 1
        buildb  0,10000 ; adjust bit 0
;
; 1) 8000h is converted to -0, not 32768.
; 2) the output always occupies 6 bytes, right-
;    justified.
;
; Input  = HL: the number to be converted
;          DE->the first byte of the field
;
; Output = DE advanced to byte after last digit.
;          field has blanks, blank/minus, digits.
;
blank   equ     ' '
minus   equ     '-'
zero    equ     '0'
;
cvtdec: add     hl,hl   ;HL sign to CY, and shift
        push    af      ;save CY
        jp      nc,cvt  ;if HL>0, go convert
;
        ld      a,h     ;else get 2's complement
        cpl
        ld      h,a
        ld      a,l
        cpl
        ld      l,a
        inc     hl
;
cvt:    push    de      ;save destination address
        call    hldec   ;convert HL into 6-digit
;                        packed decimal in C,D,E
        pop     hl      ;destination addr. to HL
        push    hl      ;and save for later
        ld      b,c     ;unpack C,D,E to output
        call    unpack
        ld      b,d
        call    unpack
        ld      b,e
        call    unpack
        ex      de,hl   ;final address to DE
        ld      b,5     ;last, do zero-blanking
        pop     hl      ;on unpacked digits
        call    zeroblank
;
        pop     af      ;if input was positive,
```

```
        stax    d       ; output digit
        inx     d
; now build the thousands digit
        mvi     a,3Fh   ; all possible result bits
        buildb  3,1000  ; adjust bit 3
        buildb  2,1000
        buildb  1,1000
        buildb  0,1000  ; adjust bit 0
        stax    d
        inx     d
; build hundreds digit
        mvi     a,3Fh
        buildb  3,100
        buildb  2,100
        buildb  1,100
        buildb  0,100
; tens digit
        mvi     a,3Fh
        buildb  3,10
        buildb  2,10
        buildb  1,10
        buildb  0,10
        stax    d
        inx     d
; units
        mvi     a,3Fh
        buildb  3,1
        buildb  2,1
        buildb  1,1
        buildb  0,1
        stax    d
        inx     d       ; to be like DDJ version
; all done
        pop     b
        pop     h
        ret
```

**Listing V**

```
; CVTDEC -- Binary to decimal character routine.
;
; Stores the decimal representation of HL in the
; six bytes at [DE+].  This routine differs from
; that in DDJ #56 in two ways:
```

*(Continued on top of page 38)*

```
        ret     nc      ;then return.
;
        dec     hl      ;HL->before 1st non-zero
        ld      (hl),minus ; digit, put in sign.
        ret
;
; HLDEC -- convert 15-bit unsigned binary number
; in HL to 5 packed decimal digits in C, D, E.
;
; Input  = HL contains 15-bit unsigned number,
;          left-justified in 16 bits of HL.
; Output = C, D, E contain a 6-digit BCD number
;          in the range 000000 to 032767.
;
; Alters AF, BC, DE, HL
;
hldec:  ld      b,15    ; bit count
        ld      c,0     ; initialize decimal
        ld      de,0    ; accumulators.
;
hldlp:  add     hl,hl   ; shift binary bit to CY.
        ld      a,e     ; double decimal,
        adc     a,a     ; propogate the carry
        daa             ; (in BCD)
        ld      e,a
;
        ld      a,d
        adc     a,a
        daa
        ld      d,a
;
        ld      a,c
        adc     a,a
        daa
        ld      c,a
;
        dec     b       ; continue for 16 bits
        jp      nz,hldlp
        ret
;
; UNPACK -- unpack BCD in B to ASCII at [HL+]
;
```

*(Continued on next page)*

# DR. DOBB'S CLINIC

(Listing continued, text begins on page 7)

```
; Input  = B contains two BCD digits
;          HL->destination for ASCII
;
; Output = HL->byte after rightmost digit
;
; Alters AF
;
unpack: ld      a,b     ; get left digit
        rra
        rra
        rra
        rra
        call    conv    ; and convert to ASCII
        ld      a,b     ; get right digit
conv:   and     0Fh     ; isolate digit
        add     a,zero  ; convert to ASCII
        ld      (hl),a  ; store it
        inc     hl      ; bump pointer
        ret             ; and return
;
; ZEROBLANK -- Blank leading ASCII zeroes within
; the field pointed to by HL.  Maximum number of
; bytes to be examined is specified in B.
;
zeroblank:ld    a,zero  ;ASCII zero for compares
zbloop: cp      (hl)    ;if digit is not zero,
        ret     nz      ;then done, return
        ld      (hl),blank; else blank it
        inc     hl      ; point to next digit
        dec     b       ; continue for all digits
        jp      nz,zbloop
        ret
```

**Listing VI**

```
;
; BINDEC converts a 16-bit binary value in HL to
; a five byte (ASCII) decimal string starting at
; [DE+].  Converted characters are stored MSD
; to LSD with leading an trailing zeroes.
;
; REGISTERS: BC  INTERNAL - Place holder value
```

# BASIC FORTH (Text begins on page 12)

```
0001 REM **********   BASIC-FORTH  V. 3 ****************
0002 DIM S(40),R(20),L(10),L0(10),I$(800)
0003 DIM B$(80)
0004 PRINT "BASIC-FORTH  V.3"
0020 REM  N IS SP, M IS RP, K IS IP, AND L IS W.
0024 ON ERR THEN GOTO 0029
0026 ON ESC THEN STOP
0028 GOTO 0030
0029 PRINT A$," ?"
0030 LET M=0
0032 LET N=0
0050 REM ********  TEXT INTERPRETER   **********
0060 LET K=1
0062 INPUT I$
0064 LET L1=0
0070 LET L(K)=L1
0072 LET L0(K)=LEN(I$)
0074 LET L1=L0(K)
0100 IF N<0 THEN GOTO 0106
0104 GOTO 0110
0106 PRINT "STACK EMPTY"
0108 GOTO 0030
0110 LET L(K)=L(K)+1
0112 IF L(K)>L0(K) THEN GOTO 0132
0114 LET B$=I$(L(K),L(K))
0116 IF B$=" " THEN GOTO 0110
0118 LET A$=B$
0120 LET L(K)=L(K)+1
0121 IF L(K)>L0(K) THEN GOTO 0130
0122 LET B$=I$(L(K),L(K))
0124 IF B$=" " THEN GOTO 0130
0126 LET A$=A$,B$
0128 GOTO 0120
0130 GOTO 0200
0132 IF K<2 THEN GOTO 0060
0134 LET K=K-1
0135 LET I$=I$(1,L0(K))
0136 LET L1=L0(K)
0138 GOTO 0110
0200 REM **********  DICTIONARY  **********
0210 REM   300-900 :: HIGH LEVEL DEFINITIONS
0300 IF A$<>"SQUARE" THEN GOTO 0310
0302 LET B$="DUP *"
0304 LET I$=I$,B$
0306 LET K=K+1
0308 GOTO 0070
0310 IF A$<>"CUBE" THEN GOTO 0320
0312 LET B$="DUP SQUARE *"
0314 LET I$=I$,B$
0316 LET K=K+1
0318 GOTO 0070
0320 IF A$<>"TEST" THEN GOTO 0330
0322 LET B$="DO PI 10 / R@ * SIN . LOOP"
0324 LET I$=I$,B$
0326 LET K=K+1
0328 GOTO 0070
0330 REM
0900 REM  **********  LOW LEVEL DEFINITIONS-- NUCLEUS **********
0902 IF A$<>"+" THEN GOTO 0910
0904 LET N=N-1
```

```
;               DE  ENTRY - Pointer to output string
;               HL  ENTRY - Binary value to convert
;               AF  INTERNAL - Temporary value
;
; OUTPUT:       BC  preserved
;               DE  preserved
;               HL  preserved
;               AF  unpredictable
;
BINDEC: PUSH    BC          ; SAVE B AND C
        PUSH    DE          ; SAVE D AND E
        PUSH    HL          ; SAVE H AND L
;
        LD      BC,10000 ; LOAD BC WITH 10**4
        CALL    DIVSUB   ;    PROCESS
        INC     DE       ;    BUMP STRING POINTER
        LD      BC,1000  ; LOAD BC WITH 10**3
        CALL    DIVSUB   ;    PROCESS
        INC     DE       ;    BUMP STRING POINTER
        LD      BC,100   ; LOAD BC WITH 10**2
        CALL    DIVSUB   ;    PROCESS
        INC     DE       ;    BUMP STRING POINTER
        LD      BC,10    ; LOAD BC WITH 10**1
        CALL    DIVSUB   ;    PROCESS
        INC     DE       ;    BUMP STRING POINTER
        LD      A,L      ; REMAINDER TO A
        ADD     A,30H    ;    ADJUST FOR ASCII
        LD      (DE),A   ;    STORE AT STRING+4
;
        POP     HL
        POP     DE
        POP     BC
        RET
;
DIVSUB: XOR     A           ; CLEAR THE PLACE COUNTER
DIVS01: OR      A           ; CLEAR THE CARRY FLAG
        SBC     HL,BC       ; SUBTRACT PLACE VALUE
        JP      M,DIVS02 ;    AND IF MINUS, DONE
        INC     A           ; ELSE BUMP PLACE COUNTER
        JP      DIVS01      ;    AND REPEAT
DIVS02: ADD     HL,BC       ; RESTORE REMAINDER
        ADD     A,30H       ; ADJUST FOR ASCII
        LD      (DE),A      ;    STORE IN STRING
        RET
```

**End Listing**

```
0906 LET S(N)=S(N)+S(N+1)
0908 GOTO 0100
0910 IF A$<>"-" THEN GOTO 0920
0912 LET N=N-1
0914 LET S(N)=S(N)-S(N+1)
0916 GOTO 0100
0920 IF A$<>"*" THEN GOTO 0930
0922 LET N=N-1
0924 LET S(N)=S(N)*S(N+1)
0926 GOTO 0100
0930 IF A$<>"/" THEN GOTO 0940
0932 LET N=N-1
0934 LET S(N)=S(N)/S(N+1)
0936 GOTO 0100
0940 IF A$<>"ABS" THEN GOTO 0950
0942 LET S(N)=ABS(S(N))
0944 GOTO 0100
0950 IF A$<>"ATN" THEN GOTO 0960
0952 LET S(N)=ATN(S(N))
0954 GOTO 0100
0960 IF A$<>"COS" THEN GOTO 0970
0962 LET S(N)=COS(S(N))
0964 GOTO 0100
0970 IF A$<>"EXP" THEN GOTO 0980
0972 LET S(N)=EXP(S(N))
0974 GOTO 0100
0980 IF A$<>"INT" THEN GOTO 0990
0982 LET S(N)=INT(S(N))
0984 GOTO 0100
0990 IF A$<>"LOG" THEN GOTO 1000
0992 LET S(N)=LOG(S(N))
0994 GOTO 0100
1000 IF A$<>"RND" THEN GOTO 1010
1002 LET S(N)=RND(-N)
1004 GOTO 0100
1010 IF A$<>"SGN" THEN GOTO 1020
1012 LET S(N)=SGN(S(N))
1014 GOTO 0100
1020 IF A$<>"SIN" THEN GOTO 1030
1022 LET S(N)=SIN(S(N))
1024 GOTO 0100
1030 IF A$<>"SQR" THEN GOTO 1040
1032 LET S(N)=SQR(S(N))
1034 GOTO 0100
1040 IF A$<>"TAN" THEN GOTO 1050
1042 LET S(N)=TAN(S(N))
1044 GOTO 0100
1050 IF A$<>"^" THEN GOTO 1060
1052 LET N=N-1
1054 LET S(N)=S(N)^S(N+1)
1056 GOTO 0100
1060 IF A$<>"S?" THEN GOTO 1070
1062 FOR I=1 TO N
1064   PRINT S(N-I+1)
1066 NEXT I
1068 GOTO 0100
1070 IF A$<>"." THEN GOTO 1080
1071 IF N<1 THEN GOTO 0106
1072 PRINT S(N)
1074 LET N=N-1
1076 GOTO 0100
```

*(Continued on next page)*

# BASIC-FORTH

**(Listing continued, text begins on page 12)**

```
1080 IF A$<>"DUP" THEN GOTO 1090
1082 LET N=N+1
1084 LET S(N)=S(N-1)
1086 GOTO 0100
1090 IF A$<>"DROP" THEN GOTO 1100
1092 LET N=N-1
1094 GOTO 0100
1100 IF A$<>"SWAP" THEN GOTO 1110
1102 LET S(N+1)=S(N-1)
1104 LET S(N-1)=S(N)
1106 LET S(N)=S(N+1)
1108 GOTO 0100
1110 IF A$<>"OVER" THEN GOTO 1120
1112 LET N=N+1
1114 LET S(N)=S(N-2)
1116 GOTO 0100
1120 IF A$<>">R" THEN GOTO 1130
1122 LET M=M+1
1124 LET R(M)=S(N)
1126 LET N=N-1
1128 GOTO 0100
1130 IF A$<>"R>" THEN GOTO 1140
1132 LET N=N+1
1134 LET S(N)=R(M)
1136 LET M=M-1
1138 GOTO 0100
1140 IF A$<>"R@" THEN GOTO 1200
1142 LET N=N+1
1144 LET S(N)=R(M)
1146 GOTO 0100
1200 REM **********  CONTROL STRUCTURES  *********
1202 IF A$<>"=" THEN GOTO 1210
1203 LET N=N-1
1204 IF S(N)=S(N+1) THEN GOTO 1207
1205 LET S(N)=0
1206 GOTO 0100
1207 LET S(N)=1
1209 GOTO 0100
1210 IF A$<>">" THEN GOTO 1220
1212 LET N=N-1
1214 IF S(N)>S(N+1) THEN GOTO 1217
1215 LET S(N)=0
1216 GOTO 0100
1217 LET S(N)=1
1218 GOTO 0100
1220 IF A$<>"<" THEN GOTO 1230
1222 LET N=N-1
1223 IF S(N)<S(N+1) THEN GOTO 1227
1224 LET S(N)=0
1225 GOTO 0100
1227 LET S(N)=1
1228 GOTO 0100
1230 IF A$<>"IF" THEN GOTO 1250
1231 LET N=N-1
1232 IF S(N+1) THEN GOTO 0100
1233 FOR I=L(K) TO LO(K)-3
```

*(Continued on top of page 42)*

```
1234   LET B$=I$(I,I+3)
1235   IF B$="ELSE" THEN GOTO 1240
1236   IF B$="THEN" THEN GOTO 1240
1237 NEXT I

1238 PRINT "IF?"
1239 GOTO 0030
1240 LET L(K)=I+4
1241 GOTO 0100
1242 GOTO 0100
1250 IF A$<>"ELSE" THEN GOTO 1260
1252 GOTO 1233
1260 IF A$<>"THEN" THEN GOTO 1270
1262 GOTO 0100
1270 IF A$<>"BEGIN" THEN GOTO 1280
1272 LET M=M+1
1274 LET R(M)=L(K)
1276 GOTO 0100
1280 IF A$<>"UNTIL" THEN GOTO 1300
1282 LET N=N-1
1283 IF S(N+1) THEN GOTO 1288
1284 IF S(N+1) THEN GOTO 0100
1286 LET L(K)=R(M)
1287 GOTO 0100
1288 LET M=M-1
1289 GOTO 0100
1300 IF A$<>"DO" THEN GOTO 1320
1302 LET M=M+1
1304 LET R(M)=L(K)
1305 LET M=M+1
1306 LET R(M)=S(N-1)
1308 LET M=M+1
1309 LET R(M)=S(N)
1310 LET N=N-2
1312 GOTO 0100
1320 IF A$<>"LOOP" THEN GOTO 1340
1322 LET R(M)=R(M)+1
1324 IF R(M-1)>R(M) THEN GOTO 1330
1326 LET M=M-3
1328 GOTO 0100
1330 LET L(K)=R(M-2)
1332 GOTO 0100
1340 REM *******  CONSTANTS  *********
1500 IF A$<>"PI" THEN GOTO 1510
1502 LET N=N+1
1504 LET S(N)=3.14159
1506 GOTO 0100
1510 IF A$<>"0" THEN GOTO 1520
1512 LET N=N+1
1514 LET S(N)=0
1516 GOTO 0100
1520 IF A$<>"STOP" THEN GOTO 1600
1522 STOP
1600 REM **********  NUMBER  **********
1604 LET N=N+1
1606 LET S(N)=VAL(A$)
1608 GOTO 0100
```

**End Listing**

**Dr. Dobb's Clinic**

*(Continued from page 7)*

very similar algorithm appeared in the August, 1981, issue of *Byte*.

## Decimal, Subtracting Binary Conversion (DSBC)

M. Patton Davis of Ann Arbor, MI sent a simple conversion routine that relies on the Z80's 16-bit subtract instruction (Listing 6, p. ). Davis describes it:

> "The method employed is to repeatedly subtract decreasing powers of 10 starting with the largest that can be contained in 16 bits, i.e., 10,000. The number of successive subtracts thus represents the decimal multiplier for that power of ten, and converting to ASCII merely involves adding 30H and storing. The routine occupies only 55 bytes; execution times vary from about 500 cycles (HL=0000H) to approximately 1350 (HL= FFFFH)."

Actually, its time is proportional to the sum of the digits in the result; it would be slower at producing 9999 (36 subtracts) than at producing 32767 (25 subtracts).

## Summary

It's a great pleasure to watch our readers pounce on a problem; the variety of the solutions, and the insightful explanations that accompany them, demonstrate what a powerful brain trust you all represent. Think of the Clinic whenever you meet a computing problem. Not only can your fellow readers solve it, they'll enjoy doing so.

Next month we'll have suggestions for taming the MX-80, more about calendars, and a fast integer square-root routine. In the meantime, consider this question from Jerry Throckmorton of Fairborn, OH:

> "I am developing some business application programs in which I want to use a check digit algorithm to detect data entry errors. My problem is in locating and selecting the "best" algorithms.
>
> "Should I use Modulus 10 or Modulus 11 and should I weight the digits based upon an arithmetic progression, a geometric progression, or some other scheme? What algorithms are used in credit card numbers, bank account numbers, etc.? I need some good information upon which to base a decision. Can you help?" ▶▶J

[1] Tab, J. A. and Roginsky, M. L. "uP Algorithms Make BCD-Binary Conversions Super Fast," *EDN*, Jan 5, 1977, p. 46.

**(LISTING ON PAGE 36)**

**BASIC-FORTH**

*(Continued from page 13)*

The program listings are shown on page 40. This program was coded in the Extended BASIC on a NOVA-3 computer made by Data General Co. It would be very easy to modify this program for execution on small BASIC computers. The only problem which might arise is in the area of string functions used to implement the text interpreter. Some modification may be needed depending on how string variables are handled and how substrings are extracted and concatenated. The program can be extended or modified to suit specific applications. The area between line 300 and 900 is reserved for additional high-level or low-level instructions. The FORTH interpreter runs slowly under BASIC, but not terribly so. Its advantage over regular FORTH systems is that all numbers are floating-point numbers, and most of the transcendental functions in BASIC are available for interactive execution.

The main purpose of this BASIC-FORTH is educational – to demonstrate the basic principles involved in a virtual FORTH computer such as the dictionary, the stacks, and the interpreter. FORTH as a programming language can be implemented on different computers, even on top of BASIC as in this case. The program itself is also useful to BASIC programmers designing small control systems using these low-cost BASIC computers.

▶▶J

**(LISTING ON PAGE 40)**

*DDJ's* responsive readership can give you invaluable feedback on your ideas and innovations. Find out how you can reach them by contacting us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# OF INTEREST

by Dave Cortesi

## New Hardware: Large, Small, Distributed.

We heard this month of several new hardware systems, ranging from the very large to the very small. The largest is the DPS-8000 from **Ithaca Intersystems**. That's a multi-user system based on the S-100 bus and the Z8002 16-bit chip. Ithaca have opted for a new operating system called COHERENT; it is claimed to provide "complete C language source compatibility with Edition Seven of UNIX." So why not UNIX? Pascal and C compilers and "standard commands and utilities of UNIX" are included; prices begin at $20,000.

**MicroDaSys** wrote to say that they have their "68K MiniFrame" in full production. It contains two M68000 chips and an M6809 to handle the I/O chores. MicroDaSys went for real UNIX version 7; they include translators for C, FORTRAN-77, and BASIC. Prices start at "under $12,000;" a hard-disk system able to handle 6 users costs "under $20,000."

While the micro makers scale their machinery up, an old mini-maker is moving down into the desktop market. **Data General** has prepared a new model MPT/100, a handsome unit that resembles a slightly bloated H89. It contains 64K RAM, two double-density 5" drives, and two serial ports. Its software is based on a multi-tasking monitor called MP/OS and includes FORTRAN IV, BASIC, Pascal, and an assembler. The CPU is Data General's 16-bit microNOVA; the system costs $5400.

Have we gotten in range of your pocketbook yet? No? Well, Radio Shack have gotten around to making a complete system out of their Pocket Computer. $150 gets you an L-shaped nest into which the little gem nestles beside a tiny matrix printer and a cassette interface. If you toss in another $15, they'll give you a "plush-lined softcase" to carry it in. Radio Shack also announced a 16-line telecommunications multiplexer. Through it, a TRS-80 II can supply data to 16 Videotex terminals or other TRS-80's. This box opens all sorts of intriguing possibilities for local networking, especially in light of the fact that the Model II can also communicate with an IBM mainframe (see last month).

## Two Reports for Software Sellers

The National Bureau of Standards has a publication on "language-independent and machine-independent criteria for assessing the quality of floating point arithmetic operations and functions." It sounds like a must read for compiler designers; order stock number 003-003-02324-8 from the **Superintendent of Documents**; it costs $2.75. People who sell software might be curious about a $20 report, "How to Copyright Computer Software," from **SOFTPROTEX**.

## Hard and Soft Books

Readers deep into hardware should be interested in *Intuitive IC Electronics* by T. M. Frederiksen (McGraw-Hill, $18.50). The book claims to have "thorough coverage of the internal mechanisms common to all IC devices," to use "an absolute minimum" of mathematics, and to require "no prior knowledge of semiconductors."

Software folks, especially those involved in control applications, might investigate *Real Time Programming* by C. C. Foster (Addison-Wesley, $9.00). It provides "a broad survey of the problems involved in trying to connect a digital computer to the real world," and contains "practical and applied exercises" including a barbershop quartet of computers (?!).

## Two Compendia

Those indefatigable compilers at **Datapro** have collected profiles of 2,000 software products and 1,000 vendors in their *Directory of Microcomputer Software*. Products are indexed by application areas, name, system, and vendor; vendors are indexed by state and zip code. User ratings of the products have been assembled and tabulated. The whole stupendous compendium, with 12 monthly supplements, costs $300. If we ran a computer store we'd keep one of these behind the counter.

If you have a collection of magazine back-issues, you might want to know of yet another diskette-based article index. **FYI** has published this one; it contains 984 entries indexing articles from *Dr. Dobb's* and most of the other micro journals. The index is available for CP/M systems in both 8" and 5" formats; it's $30 alone or $70 with the company's IC-IRS retrieval software.

## Math Libraries for All

The mail brought us happy news of a math library that has been upgraded for the Cray-1. We can't imagine why they told us, but we're happy to spread the word. If your Cray-1 has been gathering dust in the bedroom for lack of a math package, call us for details.

On the other hand, you might have a CP/M system that needs a real math package. One of those came in the same mail: the Mathematical Programming System for Micros from **MPI**. MPSM is "capable of solving linear and separable programming problems," and can handle "problems with literally thousands of variables and constraints using a 64K computer." The package is said to resemble IBM's package MPSX; it costs $750.

## General Software for CP/M

At this year's Computer Faire, Gary Kildall promised a sophisticated program to convert 8080 code to 8086 code. Now **Digital Research** has announced XLT86. It reads an ASM or MAC source file and writes a source file for ASM-86, the assembler in CP/M-86. The program is said to perform "extensive data flow analysis to determine register usage throughout the original program. The information collected . . . is used during translation to eliminate unnecessary flag save and restore operations." The CP/M version of XLT86 is priced at $150; a version to run on the DEC VAX series (!) costs $8000 (!!).

Three smaller companies turned up with CP/M add-ons. "Unica" from **Knowlogy** is said to be a "package of software instruments" (nice phrasing, that) which brings something of the capabilities of UNIX to CP/M. The package contains a number of UNIX-like commands and a structured assembler; it requires a Z80 and CP/M 2.x and costs $200. **Ficomp** tells us of some nice directory utilities, the most interesting of which keeps track of a version number for each file on a disk. And **Southern Computer Systems** have an ISIS disk utility for CP/M; with it you can read the disks from the Intel User Library, or write files to be read by an ISIS system.

## 5 Compilers, 4 Systems, 3 Languages

This, it seems, is a good month for compilers. We've got compilers for BASIC, COBOL, and Pascal on a variety of hardware and software systems. The first notice out of the pile came from Radio Shack. It announced the RSBASIC development system for the Models I and III. Its centerpiece is a compiler that converts a program in a "business-oriented BASIC" – *not* compatible with normal

TRS-80 BASIC – into a p-code representation. A run-time interpreter executes the compiled code, and a special purpose program editor is included in the $150 price.

Then we heard of a new release of S-BASIC for CP/M systems from **Micro-Ap**. This $300 business BASIC has been enhanced with a set of "ALGOL-like" control constructs: While-Do, Repeat-Until, and Case. It supports nested and recursive functions and procedures; it adds fixed-point BCD numbers to the usual complement of data types; and it can dynamically assign the destination of I/O at run time.

**Ellis Computing** chimed in with Edition II of their Nevada COBOL compiler. We can't tell which of the features listed in the press release are new to this Edition, but in sum they paint a picture of a very complete subset of ANSI COBOL, especially considering the $150 price.

And we got news of two new Pascal implementations. One is an addition to OASIS from **Phase One**; it claims to be modelled on UCSD Pascal, with improvements. One improvement that caught our eye was that this compiler does not restrict the order of declarations; constants, types, variables and procedures may be grouped in any order. The rigid sequence imposed by one-pass compiler design irritates us every time we use Pascal; it's nice to find an implementation that removes it. The $450 compiler produces linkable machine code.

**TSC** turned up with another, equally nice Pascal for the FLEX and UniFLEX operating systems. It compiles to assembly source code for the 6809. This implementation has 16-digit numeric precision and distinguishes names as long as 160 characters, two features we commend to other designers. TSC chose to implement the file I/O syntax that Wirth specified, instead of making random extensions to the read and write procedures in the name of "simplicity." Good for them.

## Late News From IBM

On August 12th, IBM announced the IBM Personal Computer, a small home machine. It will be sold, IBM announced, through Computerland stores and Sears Business Machines outlets. The machine consists of a console with keyboard, a separate video monitor (which may be a home TV), and cassette and disk units. It uses 5-inch disks; it's apparently limited to having only two of them as the maximum disk storage is given as 320 KB.

We weren't told what kind of CPU the Personal Computer uses, but we have several clues in the software it supports. The machine will use both CP/M and the UCSD Pascal system. Its main programming language is Microsoft BASIC, enhanced with statements for color control. The application packages available initially include accounting software from Peachtree, VisiCalc (tm) from Personal Software, and communications software.

The prices quoted for the machine range from $1600 for a 16K starter system up to $6300 for a machine with two disks and 256K of storage. In what we suspect will prove to be a very wise move, the company will sell the machine at a discount to its employees. That will create in short order the pool of experienced, talented users that it took other makers years to develop.

This machine, not the System 23 Datamaster we described last month, is IBM's entry into the personal market. Of the Datamaster's $10,000 price we said, "that rushing sound you hear is an industry sighing in relief." We were overhasty there; that red glow you see in the west is us blushing. Even so, we're excited about this announcement. IBM's presence in the market will intensify competition; the standards of support and documentation will go up and prices will go down as the other makers prepare to meet the challenge. DDJ

## Contact Points

**Dr. Gary Bitter**, Arizona State University, 203 Payne, Tempe, AZ 85287; (602) 965-3322.

**Data General**, Route 9, Westboro, MA 01581; (617) 366-8911.

**Datapro** Research Corporation, 1805 Underwood Blvd., Delran, NJ 08075; (609) 764-0100.

**Digital Research**, POB 579, Pacific Grove, CA 93950; (408) 649-3896.

**Ellis Computing**, 600 41st Ave., San Francisco, CA 94121; (415) 751-1522.

**Ficomp**, Inc., 3017 Talking Rock Drive, Fairfax, VA 22031; (703) 280-1394.

**FYI**, Inc., POB 10998 #615, Austin, TX 78766.

**Ithaca Intersystems**, 1650 Hanshaw Road, Ithaca, NY 14850; (607) 257-0190. Or, 58 Crouch Hall Road, London N8 8HG, U.K.

**Knowlogy**, POB 283, Wilsonville, OR 97070; (503) 635-5701.

**Micro-Ap**, 7033 Village Parkway, Suite 206, Dublin, CA 94566; (415) 828-6697.

**MicroDaSys**, Inc., POB 36215, Los Angeles, CA 90036; (213) 731-0876.

**MPI**, 7602 Pin Oak Drive, Humble, TX 77338.

**Osborne Computer** Corporation, 26500 Corporate Ave., Hayward, CA 94545; (415) 887-8080.

**Phase One**, 7700 Edgewater Drive, Suite 830, Oakland, CA 94621; (415) 562-8085.

**SOFTPROTEX**, POB 271, Belmont, CA 94002.

**Southern Computer Systems**, POB 3373A, Birmingham, AL; (205) 933-1659.

**Superintendent of Documents**, U.S. Government Printing Office, Washington, D.C. 20402.

**TSC**, POB 2570, West Lafayette, IN 47906; (317) 463-2502.

**Vancouver Amateur Digital Communications Group**: contact Hugh Pett (VE3FLL), 36 Lismer Cr., Kanata, ON K2K 1A2.

## The Conference Tree

*(Continued from page 15)*

## Conclusion

We are satisfied that The Conference Tree meets its major design goals of versatility, low cost, and ease of use. Learning is painless enough that the system is acceptable as a party toy for people who have never used any computer before. It works equally well for more serious uses, such as the ongoing public computer conference run by the FORTH Interest Group. A conferencing system can often be set up to use a computer's and phone line's idle time, for no extra cost except electricity. All calls at the computer are incoming.

The biggest limitation on the use of this and other telecommunications for the general public is the availability of terminals. Most people will not invest four to five hundred dollars or more on unfamiliar equipment with unknown usefulness to them. We are considering renting out low-cost terminals to allow computer communication with a low entry fee.

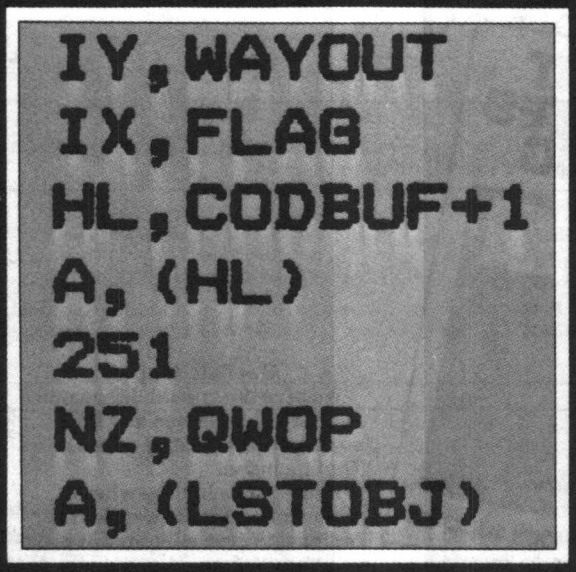
The two original Conference Tree systems run by CommuniTree Group, are at (415) 526-7733 and (415) 928-0641; these have referrals to others. The FORTH conference is at (415) 538-3580. Use a 300 or 110 baud modem, and type two carriage returns to start; then the system is self-instructing. Or for more information contact CommuniTree Group, 470 Castro St., Suite 207-3002, San Francisco, CA 94114, (415) 474-0933 (voice). DDJ

$2.50
($3.00 in Canada)

# DR. DOBB'S JOURNAL

For Users of Small Computer Systems

Number 61 November 1981 Volume 6, Issue 11




**ADVEN80: An Advanced System for Developing "Adventures"**
*Peter D. Scargill*

**North Star Terminal Programs**
*Richard Blessing*

**Small VM: Nucleus of a Portable Software Development Environment**
*J. E. Hendrix*

**Small-c Grows Up**
*Richard Roth*

**plus more!**

## A Clinic for Ailing Algorithms . . .

## and Late-Breaking Product News

# DR. DOBB'S JOURNAL
For Users of Small Computer Systems

November 1981 Volume 6, Issue 11, Number 61


## CONTENTS


**ADVEN-80 – An Advanced Adventure Development System** ..... 8
*by Peter D. Scargill*

**Terminal Programs for North Star** ..... 28
*by Richard Blessing*

**Small VM, Nucleus of a Portable Software Development System** ..... 34
*by J. E. Hendrix*

**Small-C Grows Up** ..... 46
*by Richard Roth*

**The Hacker Papers** ..... 52
*by "Gandalf" and others*

Letters ..... 4
Of Interest ..... 6
Dr. Dobb's Clinic ..... 58



© 1981 by People's Computer Company unless otherwise noted on specific articles. All rights reserved.



Editor, Marlin Ouverson
Assistant Editor, Julie Anton
Contributing Editors, Dave Caulkins and Dave Cortesi

Marketing Director, Craig S. Harper
Advertising Sales, Laird Foshay
Circulation Manager, Peter Clark
Circulation Assistant, Susan Hinton

Art Director, Clifford West
Artist, Barbara Ruzgerian
Typesetter, Renny Wiggins
Proofreader, Nancy Heubach

*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025.


## EDITORIAL

People's Computer Company is dedicated to computers (and computing, a very different matter) for *people*. The beneficial result of this credo will be a much friendlier and more pleasant future than that formed out of the concept of keeping these powerful machines reserved for companies. This is an admitted bias, however, and we must not allow it to prevent us from taking a close look at the technological growing pains some people experience. Are they brought about by computing? Or perhaps these are personality-related events which would manifest regardless of the environment. What the technology-laden future holds for humanity is a question we may well address.

What, specifically, will computers do for people? We suspect their potential will only be fully maximized when they are integrated into a home information/management/entertainment system. So we assume that telecommunications research and the issues of compatibility (of both software and hardware) are of prime importance. But when these are realized, will it isolate individuals from contact with one another, or will it bring families, local communities and national and global concerns closer together? We believe the latter.

What about the research currently being expended into artificial intelligence, into programs to analyze personality or to be used as "self improvement" tools. Will these help people to relate to one another more easily, to make broader social and personal contacts, or will it merely make it simpler to relate to machines than to humans?

The answers to these issues are complex and vary widely depending upon who one asks. But, all theory aside, what will happen when personal computing is a reality for the majority of our population? All we can do is take a glimpse at what is happening now. Whether the sampling available today is indicative of tomorrow is itself a moot topic. In the interests of those who feel these subjects must be addressed, this issue includes "The Hacker Papers." First published in *Psychology Today*, it consists of an on-line dialogue of computer hobbyists and students. Because it addresses many key points from a firsthand view, we consider it very worthy of publication in these pages.

May it inspire much thought and comment on the part of the broad *DDJ* audience.... *–MDO*


**Subscription Rates:** $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. dollars, drawn on a U.S. bank.

**Writer's Guidelines:** All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

**Donating Subscribers** Contributing Subscriber: $50/year ($25 tax deductible). Retaining Subscriber: $75/year ($50 tax deductible). Sustaining Subscriber: $100/year ($75 tax deductible). Lifetime Subscriber: $1000 ($800 tax deductible). Corporate Subscriber: $500/year ($400 tax deductible, receives *five* one-year subscriptions).

**Contributing Subscribers:** Eric H. Johnson, Jerry Horn, Edward W. Carr, Steven Fisher, Thomas H. Grohne, G. V. Elkins, Robert M. Connors, William J. McEown, Robert C. Luckey, Kim Leary, Robert N. Stabler, DeWitt S. Brown, John B. Palmer, E. Glion Curtis, Jack Miller, Burks A. Smith, Transdata Corporation, Mark Ketter, John W. Campbell, Friden Mailing Equipment, Howard E. Decker, Jim Wilson, Frank Lawyer. **Retaining Subscribers:** Osamu Shimizu, Scott B. Guthery, Robert L. Edens, Titus Purdin. **Sustaining Subscribers:** Gordon Eubanks, Jr. and Pete Roberts. **Lifetime:** Michael S. Zick.

**Foreign Distributors UK & Europe:** LP Enterprises, 8/11 Cambridge House, Cambridge Road, Barking, Essex, 1G11 8NT, England (Tel. 01-591-6511, TLX 892395). Hofacker-Verlag, Tegernseer Strass 18, D-8150 Holzkirchen, West Germany. **Asia & Australia:** Computer Store, POB 31-261, 22B Milford Road, Milford, Auckland 9, New Zealand. ASCII Publishing Co., Ltd., Aoyama Building 5F, 5-16-1 Minami Aoyama, Minato-ku, Tokyo 107, Japan.

# LETTERS




## Who's On FORTH?

Dear Marlin:

Congratulations on the very fine special FORTH issue of *DDJ*. It was refreshing to read some new things about FORTH that were not muddled up with the usual FORTH Interest Group hyperbole. I am sure that the articles will be quoted and reprinted for years to come.

I would like to point out a slight problem with the FORTH Decompiler listing that may cause trouble for some readers. Somewhere between my line printer and page 51 of the *DDJ* final product, a smudge appeared to the right of the "@" in line 14 of screen 35 that could be mistakenly taken for an apostrophe or a comma. Either one is a legal operator in FORTH, but would cause the program to fail.

Sincerely yours,
Ray Duncan
Laboratory Microsystems
4147 Beethoven Street
Los Angeles, CA 90066

Dear Sirs:

The design of FORTH presents virtues and vices which reflect the time and place when it sprung.

1. FORTH provided a unique solution to the dilemma of choosing between an inefficient interpretive language, and the alternative of compilers and assemblers which seem to require a debugging cycle of significant time spent in editing, recompiling, and re-creating the test environment. In contrast, FORTH offers an environment allowing editing, deletion, and recompilation of routines, without disturbing the stack. (Routines will only be disturbed if they were defined after the one which one desires to re-compile.) And it allows direct testing of routines at all levels, using parameters which may be easily set up on the stack.

2. Its reverse Polish notation is probably primarily a reflection of the needs for compilation in minimum memory and minimum time. (I do not regard RPN as a handicap to the professional programmer. I do regard it as a significant deterrent to the casual amateur.)

3. Its extreme stack orientation suggests strongly that the PDP machines on which it was developed are strongly stack oriented, and it was developed for functions which are particularly suited to stack solutions.

4. It was developed in a time of expensive memory (by today's standards), and consequently necessarily generates very compact code.

Its stack approach is fine for problems in which most subroutines functions will be using only results placed near the top of the stack. My experience indicates that for systems programming and commercial data processing this is generally not the case. My experience suggests that we will often want to use a few variables which cannot conveniently be passed on the stack. Well, FORTH does provide named variables. But I find locality of definition and dynamic allocation highly desirable. At the lowest level, variables on the stack will provide locality of definition and dynamic allocation; and at the highest level both of these considerations lose their force, but I think there is a substantial intermediate area.

I think the stack and global variable approach also leaves much to be desired in terms of record handling.

I don't believe that changes providing improved variable handling, and for that matter an optional algebraic passer, are incompatible with the essential virtue of FORTH, which is its easy code-and-test cycle.

Bill Bunn
Sen. Systems Analyst
P. O. Box 67
Covelo, CA 95428

## The Little Engine That Couldn't

Sirs:

I need help. Nearly three years ago, I plunked my money down on a Western Digital Pascal MicroEngine. When it finally came, it did not work right, and several trips in for warranty service, and several upgrades, have not significantly improved matters. Mean time between failures is still on the order of five minutes.

The problem is somewhere in the disk controller DMA path, that intermittently trashes the data, and only last month did I finally get conclusive proof that it was a hardware problem, not environmental.

I have several options: I can send it back to WD for yet another service (this time out of warranty, so at much bucks); maybe this time they will fix it, and maybe it will refuse to fail for them. Or (maybe and) I can program around the problem. Software is my bag, so I like that option. Unfortunately, the MicroEngine came without documentation (no source). I sell software that way too, but $5 is lots less to throw away if it does not work (however, mine works). To fix the software, I can:

1. Buy the source for $x,xxx. Double the cost of the computer to make it run. Not attractive.
2. Decompile the system. Could be done, and the decompiler would be (possibly) marketable, but lots of time. In the long run, probably more costly than buying source.
3. Rewrite system for version II (I have access to source). Would mean trying to figure out the differences. Less work, but still a pain. It would be easier if I had help, which brings me to
4. Find someone who has paid for the source, who would be willing to recompile it with a disk driver that I write (no need to violate nondisclosure; just give me the compiled object code).

Any takers? I would be happy to barter services (ask anyone about my software quality) for the favor.

Tom Pittman
P. O. Box 6539
San Jose, CA 95150

## Oh Say, Can I C!

Dear *DDJ*,

Add me to the list of Small-c users. I'm lazy, though, and didn't copy it from Ron Cain's articles, but bought object and source code for CP/M from the Code Works for less than it would cost me to type it in.

The reason I am writing is that I want to share with your readers a few simple features I added. As I modified the compiler to handle constant evaluation, multi-dimensional arrays, structures, and more, I soon filled all 48K of my memory. Since I have more additions planned, I either had to do a major rewrite (i.e., different algorithms) or try to compact somewhere. As the latter seemed easiest and my expected additions won't need a lot of space, I opted for it. The first thing I did was write a program (in Small-c, of course) to compress the assembly code produced by the compiler, removing redundant POPs and PUSHes, changing certain addition operations into simple increments, etc. This removed about 2K from the compiler. Next, I reduced the size of the literal pool storage by dumping the pool to the output file at the end of each function and then re-initializing it. The works because only code generates literals and code occurs

*(Continued on page 51)*

# OF INTEREST

## Pegasus Not Repulsive

In August we mentioned a BASIC compiler for OSI systems from **Pegasus Software**. It allowed coding GOTOs to absolute addresses, and we called that idea "repulsive." We certainly did not intend the adjective to apply to either the entire product or to its publisher. Reader John Fuller understood that, but he wrote to defend Pegasus' product anyway. The OSI software environment is fairly primitive, he pointed out, and such features are necessary in it. He offers the testimonial that "Pegasus Software is the main reason I have held onto my OSI computer."

## Input on Output

Notices have come to us of several sources that could help readers who sell or buy professionally. For a start, **Micro-Serve** produce a Software Vendor Directory of 200 pages, said to list 1000 vendors and 4000 products. Vendors are cross-referenced to hardware and operating system types. The single-copy price is $58; a subscription covering two semi-annual revisions is $100.

**Commodore** is now preparing the second edition of their Software Encyclopedia, a listing of products for the PET and CBM machines. The vendors are not charged for a listing, but they have to satisfy Commodore of the suitability of their product.

Irv Brechner of **Irv Brechner Enterprises** sent us a copy of his report, *Computer Store Advertising*. The $25 report discusses how a computer store (or any small retail shop) can go about making itself known effectively and economically. The advice in it struck us as practical and enlightening. We showed it to a computer store manager of our acquaintance who was also impressed. The report should be worth its $2.50-per-page price, especially to beginning entrepreneurs.

## General Input

Those who use the well-known Osborne/McGraw-Hill business software in BASIC should welcome news of the formation of an **Osborne Business Software Users Group**. The Group's aim is to promote the use, and ease the implementation, of those pioneering software packages. Membership is $10 per year domestic and $20 outside the US.

We expect that quite a few readers of *DDJ* are interested in the many 16-bit CPU chips. **Group Technology, Ltd.** is the publisher of *The 16-bit Microprocessor Handbook*, a work that sounds as if it should answer a lot of questions. The $15 book has a chapter on each of the 8086, Z8000, 68000, 9900, LSI-11, and 16032. Benchmark programs coded in each machine's language are included, with relative timings.

## A Gallimaufry of Software

A swarm of software notices came to us this month, telling of packages of every conceivable sort. Two Forth items were among this gallimaufry. **Colorware** has produced a Forth implementation for the CompuColor machine. Their release suggests that they followed the FIG-Forth standard. The first copy costs $79, extra diskettes are $69 (Ontario residents add 7% tax). **Timin Engineering** has published a diskette of sample application programs by Scott Pickett. They can be used with Timin- or FIG-Forth. The diskette is said to contain 100 screens each of documentation and of code. A standard 8" diskette costs $75; one in another diameter is $90.

Apple owners now have yet another choice in word processing packages. **Westico** is selling WORD /// (three slashes meaning "3", a logo designed to confuse typesetters nationwide). It is a full-screen editor with print formatting abilities. The editor includes the usual operations for scrolling, moving, and deleting text, and the less-common ability to insert or append text from other files. The program's formatting abilities are also conventional, if complete, except for one feature we've seen claimed by no other program: the ability to produce output in a two-column format. The $195 package can take advantage of the features of daisy and tulip printers.

Here's news of yet another spelling corrector (how quickly new ideas go from being marvels to being standard utilities!), this one for Flex and MiniFlex systems. Magic Spell (tm, and good name, guys) from **Star-Kits** is the usual sort of marvel. It runs a text file against a dictionary, noting unfound words for correction or insertion into the dictionary. The dictionary file can be edited with any text editor. That surprises us; if it can be edited then it isn't an ISAM or B-tree file. How do they get quick access to the dictionary for lookup? The program's price is $60 until November 10th, $90 thereafter. The number listed is answered by a computer, evenings; tell it to LIST MAGIC.DAT.

BASEX was an early, and fairly successful, attempt at a simple systems programming language for small micros. The BASEX user manual seems to crop up in most bookstores that carry computer books. **Interactive Microware** offers BASEX on cassette and diskette for several systems, most recently for the Heath H8. The manual, an H8 addendum, and a cassette cost $33, a reasonable price for what should be a good hobbyist tool.

The **Syntauri Corporation** has been selling their Apple-based alphaSyntauri keyboard synthesizer for some time. Now they've bundled all the supporting software into what appears to be a single, black-box operating system. It's hard to make out the details because the release was written to be read by musicians, not computists. But it seems that alphaPlus is a self-contained control program for the Syntauri-equipped Apple. The user needs to learn only the use of alphaPlus and may neglect all other aspects of Apple software. The system allows full control of the synthesizer's many features, and the user can store and play back an album of pieces from a single drive. The alphaPlus software costs $35, which is less than 2% of the price of the synthesizer. Why don't they just give it away?

**Microsoft** is pleased to inform us that they have produced Release 5.30 of their BASIC compiler. It includes CHAIN with COMMON, so that overlays can communicate through COMMON variables. The new compiler release is said to be suitable for Apple II Softcard systems as well. The link command can link programs some 16K larger than previously. Developers might take note that the rules for royalties on products developed under the compiler have been eased.

SuperCalc is what **Sorcim** call their spreadsheet calculator program for CP/M. After being previewed on several makes of hardware at industry shows, the package is now available for general release. We've tried it; it makes a V***C*** user feel right at home. A variety of terminals are supported, and SuperCalc has the ability to merge part or all of a filed worksheet into the active one.

## Build Strong Programs 3 Ways

Three notices came in that told of

application development tools, programs that help build programs. One is called Screenmaster, from **Marketing Essentials**. It is a file of code that, when included in a CBASIC-2 program, gives the programmer the means to manage a pre-formatted screen display. Utilities are provided to build and test the display definitions. Screenmaster's "introductory" price is $195; a demo disk can be had for $10.

Quic-n-Easy from **Standard MicroSystems** goes Screenmaster one, maybe two, better. Rather than building the application program in another language and calling on the display handler code, the Quic-n-Easy programmer defines the processing cycle as part of the definition of each display field. The Quic-n-Easy interpreter translates the processing statements at run time, managing the display, performing calcuations, and accessing indexed sequential files. The package price has been set at $395.

Finally, RADAR is the name of a system that makes a CP/M system into a data entry station and a suitable replacement for a keypunch station or IBM 3741 data entry station. RADAR is said to provide check digit verification, double key verification, and a number of other features that make it easy to program a data entry clerk. **Southern Computer Systems** sells it for $495.

Engineers
Scientists
Systems Programmers
Business Owners
All these read *Dr. Dobb's Journal*.
Reach them with cost-effective advertising.
(415) 323-3111.

## Percom Takes Aim

**Percom**, maker of diskette drives for SS-50 and TRS-80 systems, has built a mini-disk system for the AIM-65, KIM, and SYM machines. From one to three drives can be hung on an AIM; the drives may have 40 or 80 tracks and may have the "flippy" option. A disk operating system is provided. The price of a 1-drive system will fall around the $700 mark.

## Model III and CP/M

A complete, origin-zero, 64K, CP/M system in a TRS-80 Model III? That's what **Parasitic Engineering** claims that their Shuffleboard gives you. The $500 package consists of a circuit board, 16K of RAM, 2K of ROM, and those marvellous CP/M manuals. It installs by plugging into two existing sockets, they say, so that installation requires no permanent changes of any kind. DDJ

## Contact Points

**Irv Brechner Enterprises**, POB 265WOB, West Orange, NJ 07052.

**Colorware**, POB 6733, Station J, Ottawa, Ontario, K2A 3Z4 Canada.

**Commodore**, 300 Valley Forge Square, 681 Moore Road, King of Prussia, PA 19406; (215) 337-7100.

**Group Technology, Ltd.**, POB 87, Check, VA 24072; (703) 651-3153.

**Interactive Microware**, POB 771, State College, PA 16801; (814) 238-8294.

**Marketing Essentials**, Inc., 161 West 75th St., New York, NY 10023; (212) 580-3589.

**Micro-Serve**, 250 Cedar Hills Avenue, Nyack, NY 10960; (914) 358-1340.

**Microsoft**, 10800 N.E. Eighth, Suite 819, Bellevue, WA 98004; (206) 455-8080.

**Osborne Business Software Users Group**, 2256 Main Street, Suite 11, Otay, CA 92011; (714) 423-0538.

**Parasitic Engineering**, 1101 Ninth Ave., Oakland, CA 94606; (415) 839-2636.

**Pegasus Software**, POB 10014, Honolulu, Hawaii 96816.

**Percom Data Company**, 211 N. Kirby, Garland, TX 75042; (214) 272-3421.

**Sorcim**, 405 Aldo Avenue, Santa Clara, CA 95050; (408) 727-7634.

**Southern Computer Systems**, POB 3373A, Birmingham, AL 35255; (205) 933-1659.

**Standard MicroSystems**, 136 Granite Hill Court, Langhorne, PA 19047; (215) 968-5966.

**Star-Kits**, POB 209, Mt. Kisco, NY 10549; (914) 241-0287.

**Syntauri Corporation**, 3506 Waverly Street, Palo Alto, CA 94306; (415) 494-1017.

**Timin Engineering**, 9575 Genesee Ave., Suite E-2, San Diego, CA 92121; (714) 455-9008.

**Westico**, 25 Van Zant St., Norwalk, CT 06855; (203 853-6880.

# Adven-80

## An Advanced Adventure Development System

**by Peter D. Scargill**

*Peter D. Scargill, 24 Percy Park, Tynemouth, Tyne & Wear, Great Britain.*

For those not familiar with Adventure, it is a game which could simply not have existed before the age of the computer. Imagine reading a story tale where you follow the hero (or heroine) in a fantasy world where anything can happen. The difference between Adventure and a simple story book is that you can be the hero and be instrumental in affecting the outcome of the story, rather than being simply a passive reader.

More specifically, in most Adventures, you are in control of an "entity" which acts as your eyes and ears throughout the story. You can look, listen, manipulate objects, collect treasures, move in whatever direction you want, in fact do anything you want within the confines of a limited universe maintained by the computer.

Conceivably, some day, 3-dimensional tactile dream-worlds will become possible as technology progresses, perhaps coupling computer technology with some form of video disk system; but for the present, Adventure and similar games provide us with the first simple examples of this exciting concept.

This article is probably the first to be published of its kind, in that it provides a complete package for creating Adventures. The code presented here is a *completely general-purpose adventure "interpreter"* together with a BASIC program, designed to allow you, the reader, to create a database for your own Adventures with only the vaguest knowledge of assembly language programming. The code is not intended to be clever; experienced programmers can no doubt quite drastically improve upon the techniques used; however, it does work quite well and I would appreciate any feedback on possible improvements.

### In Brief

ADVEN-80 is a program wherein the user resides in one of many LOCATIONS. A LOCATION may have the property of being LIGHT or DARK, and usually is associated with several alternative EXITS. The player moves from one LOCATION to another, and in any LOCATION he may find an OBJECT or OBJECTS. Objects have DESCRIPTIONS, and may optionally have DETAILED DESCRIPTIONS as well. Unless specifically mentioned an object may be TAKEN or DROPPED or LOOKED-AT or otherwise manipulated within the confines of the facilities of the MATCHW table. Each object may only be in one location at any give time (or may be worn or carried as applicable). The object of the game is usually to obtain some form of TREASURE or achieve a SCORE or so many points.

Of special interest is the LINE PARSER, whose job it is to take in whatever the player enters on the keyboard, and attempt to make some sense out of it. In simplified form, the parser in ADVEN-80 takes in a line of user entry, and looks at each word in turn to see if that is a known word. If so, a token entry (somewhat like the one-byte tokens used in some BASICs) is made in a separate buffer; if not, no entry is made, and the next word is checked. When the entire line has been checked over, the token buffer is scanned for words such as "THEN" and "AND" and if found, the line is split, leaving all code after the "AND" for the next time round, in lieu of the player's next entry. This allows the use of multiple statement lines.

Generally, user input breaks down into simple two word sequences after irrelevant words have been discarded. A sentence such as "PLEASE GET THE LAMP AND LIGHT IT" breaks down into two operations, "GET LAMP" and "LIGHT LAMP." The program treats "IT" as the last object referred to. A table is scanned for such word sequences and appropriate action is taken, scores can be maintained, and the program has special facilities to accommodate scoring.

Adventure-type games for microcomputers have been around for some time and several Adventures have been made available for the more popular systems[1] although to date, very few have had anywhere near the capabilities to handle user input as had the original "BIG SYSTEMS" Adventure which many of the major Polytechnic computers have access to.

The Original Adventure for these systems was written in Fortran and required considerably more memory than that available on present-day microcomputers, and the program itself was inflexible (i.e., the creation of a new game would require the re-writing of the Fortran program as the database was an integral part of the overall structure).

ADVEN-80 is a new Micro-Adventure system, written in machine language. Consideration has been given as to obtaining the maximum usage out of the limited memory capacity of currently available small computers.

This Adventure system, written for the Z80 microprocessor, is a full-feature, machine-code Adventure utilizing the powerful functions of the above. It is partly because of this that the game runs at very high speed, with a minimum memory requirement (a simple Adventure could be written in *well* under 16K).

Because of the speed of the Z80, it was possible to use an extremely simple database layout, and the use of an intermediate translator accounts for high readability of the source listing. In reading the following, an elemental knowledge of assembly language programming in Z80 microprocessor is considered useful but not essential.

### Tables

**OBJLOC** The format of this table, containing <object no./location> pairs is simple. The pairs are contiguous with a zero to end of the table. Several location numbers have special meanings and these are listed below.

| | |
|---|---|
| Loc. 255 | Object is non-existent. Cannot be found. |
| Loc. 254 | Object is currently being carried. |
| Loc. 253 | Object is currently being worn. |

Finally, location 252 is special in that any object currently in this location is one of a pair of objects which together form a TWIN-STATE object. A typical example of this is a LAMP. The lamp may be LIT or UNLIT and hence it is necessary to have two versions of the same object.

This method is probably the simplest way of implementing a twin-state object, and in the table these two forms should always be concurrent. In the example of the lamp, the two states could be, say, LAMP and ULAMP. If the word LAMP is made accessible, then this must appear FIRST in the table. This is very important, first in ascending order; i.e.:

```
OBJLOC      /* Table name. These are mandatory.          */
LAMP 45     /* object 34, a lit lamp, is in location 45  */
ULAMP 252   /* object 35, an unlit lamp, is hidden from
               view.                                     */
ENDTABLE    /* end of table */
```

As you can see from the above table, if the scanner begins at the start of the table, and then finds a match with object LAMP, in order to blow out the lamp, the two LOCATION entries in the table are simply swapped over.

Object ULAMP was equated only and *must never* be found by the user-input encoding section, although CONDITIONS such as OC or OP can access both versions.

If a user says GET LAMP, and there are no conditions, the system will look for lamp. If this is at location 252 then the pointer will be incremented automatically to point at the ULAMP entry. Hence the user will be given whichever version of the object is *not* at location 252. (Assuming that the other version is PRESENT or being carried or whatever.)

Inventories simply look at the location numbers and if a match is found, then a description is printed. The same applies during the initial description, where objects contained within the current location are described.

**OBJDES** This table containing <object no./MSG no.> pairs is closely related to the above OBJLOC table. During "I can see . . ." type routines, the table OBJLOC is scanned in order to find objects contained within a given location, be it the current location or those reserved for objects being carried or worn . . .

When the object is found, the number of the object is used to scan through the left side of OBJDES, and when an object match is found, the description number, which refers to an entry in the UMESS table, is used to produce a description of the required object. The OBJDES table need be in no particular order, and no ZERO terminator is required as we are not searching for a non-existent object. In the sample database the ENDTABLE macro is shown, but this is just for neatness, it is not essential here.

**OBPROP** This table describes the things that can be done to an object, assuming you wish to limit that object to certain actions. I.e., you would not expect the player to be able to TAKE THE STATUE if the statue weighs a ton . . .

```
OBPROP
STATUE      LOOK,LISTEN,FEEL
DOOR        LOCK,UNLOCK,LOOK
ENDTABLE
```

Objects not in the table are considered to have universal attributes (within the limitations of the MATCHW table).

**OBDSK** This table finds use for those objects you would like to enable a close inspection of. Any object not appearing in the table will be described as ordinary looking.

```
OBJDSK
LAMP        'Looks like an old miner's lamp to me.'
ENDTABLE
```

**TABKEY** This table is used by the line parser. As the user input line is parsed, 4-byte word sections are put into a temporary buffer. (Words shorter than 4 characters long are left-justified and padded with spaces.) TABKEY is used to replace any known words with a single byte code. Unknown words are scrapped.

TABKEY is composed of the following. The first byte is KEY number. Following that are groups of 4 bytes with the last 4-byte group having the 8th bit set on the last character. If any of the 4-byte groups match that in the buffer, then the KEY byte is put into a buffer for later use. To signify the end of the table, ZERO is used as a KEY byte.

When the sentence has been completely scanned, a buffer called CODBUF will contain the single byte KEY replacements to all the words in the user input that were matched successfully. A ZERO will indicate the end of this buffer. Hence the sequence N "N NORT" will allow user inputs of "N" or "GO NORTH" to be placed in CODBUF as the single byte representing the key N.

A special byte which may end up in CODBUF is THEN. This is currently set at 250 decimal and is used for batching of commands, so that the user may say, for example "GET THE AXE THEN LOOK AT IT." The IT being taken care of automatically by a section of code that converts occurrences of IT into the last mentioned object.

KEYSETS are actually entered in macro form; i.e.:

```
BEGIN       'BEGISTAR'
```

Another example of this is LOOK "LOOKPERUEXAM" where LOOK is the word that will be equated, and will match LOOK, PERUSE and EXAMINE.

**MATCHW** This table is the one that looks at user input after it has been encoded into a byte sequence. The table is scanned until a pair of bytes corresponding to those in the CODBUF area are found. If there is no match then a default message is printed to indicate that the program found no sensible matches. Otherwise, a set of CONDITIONS is checked, and if all of these are TRUE, then a set of actions is taken.

The format is as follows:

<2 byte key><cond><255><actions><255><either ZERO or more keys>

Where cond = either no entry or any amount of conditions and arguments.

Where actions = any amount of actions.

Obviously from the above, 255 is used as a delimiter and so must not occur within either the conditions or the actions. However, 255 may be used as one of the two KEY bytes to tell the program to match up with *any* entry, i.e.:

<23,255> If CODBUF contains 23 followed by anything then the set will be accepted.

<34,0> In this case, CODBUF should contain 34 ONLY.

Again, simple equates are used to make life easy. Typical table entries are shown below, together with an explanation of each.

```
MATCH       GET,ANY
TEST        OH,INP
ACTION      TKE,INP,OK
```

In the above example, if the user enters, say, GET LAMP, then the system will test to see if the object is in the current location (OH) and if so, will allow the taking of the LAMP and will then print an OK message and exit the table, ready for more user input.

```
MATCH       LOOK,NULL
ACTION      LA,EX
```

If and only if the user enters the single word LOOK, the system will unconditionally Look Around, then EXit the

table.

```
MATCH     ANY,ANY
ACTION    PRM,37,EX
```

This entry should be put at the end of the MATCHW table. This will match ANY user input, and print message 37 from the MSG table, which would be, for example, "I don't know what you mean."

```
MATCH     ON,LAMP
TEST      OC,MATCH,OC,ULAMP
ACTION    SWP,LAMP,SWP,MATCH,OK
```

On the previous page we have a more complex example. If the user enters ON LAMP or LIGHT THE LAMP, etc., and then only if the MATCH and the ULAMP (unlit lamp) are being carried, then swap the MATCH for its USED partner, and swap the ULAMP with its LIT partner.

Notice that no attempt was made to execute the instruction SWP,ULAMP. This would have been disastrous as the ULAMP is the SECOND entry in the table, and you must only SWAP the FIRST entry (i.e., the main entry), as the command SWP FRED looks for the entry FRED in the OBJLOC table, and swaps its location with that of the *next* entry in the table. That is why the user-accessible version of the object *must* be entered into the table before the alternative form.

## Messages

**MSG** This is a general-purpose message table, containing messages such as "I can not see anything" and "ok.". These messages include directives and all general statements. The format is extremely simple:

```
0 message sequence 0 message sequence 0
=                  =                  =
```

The sequence is that of a zero byte followed by ASCII text until the next zero. The zero then acts as both delimiter and counter. As the maximum message number is known (i.e., there is no search), an end-of-table delimiter is not required, other than the zero at the end of the last message.

NOTE that the first 20 messages must *not* be altered in position as they are used by the system in some cases automatically. There is a limit of 255 messages, but this is far more than will normally be required.

In macro form, simply enter the message number and the message, i.e.:

```
45  'Hi there, this is a message, isn't it . .%'
```

Note that the standard ' symbol cannot be used *within text* and must be replaced with the alternative ` form. You will have noticed the use of one other special character, the percentage sign. This is used as a CRLF and hence must not be used for its original purpose (see below).

**UMESS** This table contains descriptions of objects and all general messages that are specific to a particular adventure.

```
7   'This is another message'
```

PLEASE NOTE: To make assembly listings more easily readable, CRLF sequence will be sent to the screen on encountering a "%" percentage sign in the text. This of course also saves a little space if you're tight with your bytes. Normally, CRLF's are not required in bulk text as the system automatically inserts these depending on the width of the screen.

It must also be noted here that the use of the character ">" anywhere in a sentence will not produce any output but will serve to destroy any pending BATCHING and so this may be used freely within error messages and any message which requires the user's immediate attention.

A final point concerning messages. It is possible to have conditional sections within messages (or indeed complete conditional messages). Flags 23 to 49 inclusive have been reserved for this purpose. These may be tested for being TRUE or FALSE only, in the following manner. Anywhere within a string, the game writer may insert parentheses as follows:

```
'This is a bit of {Auntidy } text {B as you can{C easily} see}.'
```

To explain. If flags 23 (i.e., A), 24 (i.e., B), and 25 (i.e., C), are all ZERO then the output of the message would be "This is a bit of text."; however if one were to set flag 25 then the output would be "This is a bit of untidy text."

If flag 24 was set, then you would get "This is a bit of text as you can see." It is obvious that if flag 25 were also set then the word "easily" would be inserted. Note, however, that this would not apply if flag 24 were reset, whereby all text inside the matching set of parentheses would be ignored.

Only the letters A-Z inclusive and the "/" character may be used after the opening parentheses (A-Z corresponding to flags 24-49) and the maximum NESTING depth is 10, although as many NON-NESTED sets as desired may be put into any section of text. The character "/" may be used *before* any of the alpha chars A-Z to indicate that the flag has *not* been set, i.e., is ZERO. The system flag that indicates that the "/" char is in use is only set and reset on OPENING braces so do not try any fancy multi-level conditional text while using this form.

In this version, two special flags have been reserved. Flags 21 and 22 corresponding to @ and ? are automatically set by the program after every user entry to be always opposite; i.e., if flag 21 is "1" then flag 22 will be "0" and vice-versa, but *randomly* so. Hence it is possible to write the following default message.

```
44  '{@I simply don't understand you.}{?What ????}'
```

Whenever this message is called, there is a 50-50 chance of either "What ????" or "I simply don't understand you." being output by the program.

These conditional facilities find their main use in circumstances where you want the program to differentiate between LOOK AT THE GLOBE and, say, LOOK AT THE GLOBE CAREFULLY. Obviously you can have umpteen levels of description.

The facilities may be used anywhere that text is found, either in place descriptions or in object descriptions, or even in default replies. A good use here is to allow for either random or deliberate variations in, say, the default "CANT HELP YOU" message, etc. etc.

## Descriptions

**LOC** Place descriptions begin with zero, a number 255 to 252 to indicate one of the following:

```
255   Normal place, full exit description
254   Normal place, no exit description
253   Dark place, full exit description (if lit by lamp)
252   Dark place, no exit description
```

In high level form the following words have the same effect as the numbers above:

NORMAL, NOEXIT, DARK, DARKNO

MESSAGE / KEY / LOCATION bytes, 255 to indicate end of directions, and then textual description, zero being the text terminator.

I.e.:

0 254 02 01 02 255 <TEXT> 0 253 03 32 12 255 <TEXT> 0

In macro form, a typical entry may be as follows:

```
1       NOEXIT,02,N,2
        'I am in a small room with no furniture. There '
        'appears to be nobody around at the moment.%'
```

Again, no end of table required other than the mandatory zero, (or ENDTABLE) as we are not looking for locations which do not exist. In the above example, the first text contains only one sequence of exits, (not printed) MSG2 being the description, i.e., NORTH. This corresponds to a user KEY code of 01, and in this case, if a match for the key is found, the player will be transferred to location number 03.

In the second example, MSG3 prints out SOUTH; if the user typed in GO SOUTH, or SOUTH, or S then this would be converted to KEY 03 and the user would then be transferred to location 12. Meanwhile, the present location description is printed out, and any attempt to LOOK at anything will produce the response "I can't see anything" as the 253 at the start of the sequence dictates that this is a dark location. (Of course, if the user has a LAMP, then that is a different matter. See description of LAMP.)

A special location number is 249 (RTN). When a transfer

### Flag Table

This is a 255-byte table numbered 1 to 255, zero not being used, as zero is used as a delimiter. Certain flags have special uses.

| | |
|---|---|
| 1 | Dark location indicator . . . If ZERO then dark. |
| 2 | Number of items carried. |
| 3 | Maximum permissible items to be carried. |
| 4 | Counted up once every turn. |
| 5 | " " |
| 6 | " " |
| 7 | Counted down once every turn if not zero. |
| 8 | " " |
| 9 | " " |
| 10 | Counted down if dark, lamp ON, every turn. |
| 11 | " " |
| 12 | " " |
| 13 | Present score. |
| 14 | Present location. |
| 15 | Store FF or FE (Internal). |
| 16 | 1 and 10 OR'd together. ZERO means can't see. |
| 17 | Lamp on/off. Zero when ON. 1 when OFF. |
| 18 | Number of items WORN. |
| 19 | Number of items HERE (present location). |
| 20 | Store of last location number (automatic). |
| 21-22 | Inclusive. For random text control (automatic). |
| 23-48 | Inclusive. Text controls A-Z. |

**Flags 49 to 250 are available to the user.**


**You Read**

**Dr. Dobb's Journal**

**And You Don't Subscribe?!**

**Save $13**
**off newsstand prices for 2 yrs.**
**Save $5 for 1 yr.**

Can you afford to miss an issue with information vital to your interests? As a subscriber you can look forward to articles on Small-C, FORTH, CP/M, S-100, Compiler optimization, Concurrent Programming and more, delivered right to your door. And you'll never miss the issue that covers your project.

**DR. DOBB'S JOURNAL**
**P.O. Box E**
**Menlo Park, CA 94025**

**Yes!** Sign me up for

____ 2 yrs. $47 ____ 1 yr. $25

____ I enclose a check/money order
____ Charge my Visa, MasterCard, American Express
____ Please bill me later

Name _____

Address _____

_____

_____ Zip _____

Credit Card _____ Exp. date _____

Account No. _____

Signature _____

request to this location is selected, the program will treat this in a special manner by returning the user to the previous location. Hence, you may enter a LARGE BOX and a RTN exit will bring you out, no matter where the box is situated. This avoids the need for an awful lot of location testing.

The macro version of this table makes life very simple as can be seen in the above example. No delimiters are required, and the final description is quite readable.

## Assembly Notes

To use these programs you will need access to the following utilities. A Z80 assembler (Zilog mnemonics) with conditional assembly. (If your assembler does not support the CONMSG pseudo-op and/or conditional assembly, miss out the messages and only select the sections of code that you require for your application.)

The BASIC used for the converter is Microsoft Version 5.

## Conditions/Actions

Here are the conditions and actions possible together with explanations for each. Those shown below are all available to the Adventure writer and there is no limit to the amount of conditions and/or actions that you may specify in any set.

### Conditions

| | | | |
|---|---|---|---|
| CL | xx | | Current location is xx. |
| NCL | xx | | Current location is NOT xx. |
| OP | xx | | Object xx is in present location or is worn or carried (xx=00 means obtain object number from user input). |
| OC | xx | | Object is being carried. |
| OW | xx | | Object is being worn. |
| OWC | xx | | Object is worn or carried. |
| OH | xx | | Object here but not carried or worn. |
| ONP | xx | | Object is not present. |
| ONC | xx | | Object is not being carried. |
| ONW | xx | | Object is not being worn. |
| NWC | xx | | Object is not worn and not carried. |
| ONH | xx | | Object not here, but MAY be worn or carried. |
| OE | xx | | Object exists at present. |
| ONE | xx | | Object does not exist at present. |
| @L | xx | yy | Object xx is at location yy. |
| N@L | xx | yy | Object is not at location yy. |
| RND | xx | | Random chance 0-254 . . . Where 254 is dead cert. |
| TF | xx | yy | Test flag No. xx for value yy. |
| NTF | xx | yy | Test flag No. xx for NOT value yy. |
| GE | xx | yy | Flag xx greater than or equal to value yy. |
| LT | xx | yy | Flag xx less than value yy. |
| C2 | xx | | Compare 2nd word with xx. |
| C3 | xx | | Compare 3rd word with xx. |
| NC2 | xx | | True if 2nd word is NOT = to xx. |
| NC3 | xx | | True if 3rd word is NOT = to xx. |

### Actions

| | | | |
|---|---|---|---|
| INV | | | Print an inventory of carried and worn objects. |
| TKE | xx | | Get object to carry it. |
| WR | xx | | Get object and wear it. |
| DR | xx | | Drop the object. |
| PR | xx | | Print message number xx from UMESS table. |
| PRM | xx | | Print message number xx from MESS table. |
| DL | xx | | Describe location. |
| SF | xx | yy | Set flag xx to value yy. |
| DSCOB | xx | | Describe the object. |
| SWP | xx | | Swap object xx with object xx+1. |
| MV | xx | | Move to location xx. |
| OK | | | Say 'OK' and exit table. |
| QT | | | Quit. |
| INF | xx | | Increment flag xx. |
| DEF | xx | | Decrement flag xx. |
| EX | | | Exit table. |
| SC | | | Print current score. |
| PU | xx | yy | Put object xx into location yy. |
| CR | xx | | Create object xx (Place here). |
| DES | xx | | Destroy object xx. |
| FLIP | xx | | Flip flag xx true to Zero and Vice Versa. |
| LA | xx | | Look around at items. |
| SMA | xx | | Set message A to UMESS number xx. |
| SMB | xx | | Set message B to UMESS number xx. |
| WT | xx | | Wait for xx seconds. |
| PF | xx | | Print the value of flag No. xx. |
| ATF | xx | yy | Add yy to the value of flag xx. If > 255 then set to 255. Make yy=0. |
| SFF | xx | yy | Subtract yy from flag xx. If makes xx < 0 then set xx=0. Make yy=0. |
| GRUN | xx | | Print description, directions etc. & exit. |

The above conditions and actions represent all of those implemented in the present version of ADVEN-80.

### Flag

This table is simply a collection of pairs of flag numbers and their corresponding initial data values. Entries *must* be contiguous.

```
35      22      /* means flag 35 is set to value 22 */
```

Certain flags are controlled automatically by the program and may be viewed only.

### TITL1

This final message is that which will be presented at the start of any game. It is comprised of ASCII text, with a ZERO delimiter at the end.

```
CHR$(any single ASCII number you like, or several to clear screen.)
'any title message'
'blah blah'
ENDTABLE
```

Peculiarities of this BASIC are few, the most important one is the INSTR. This returns a value of the 1st occurrence of one string or character within another string:

A=INSTR(Q$,"R") where Q$="ARE", A=2

Only sequential disk files are used, read line by line from the source file into the destination file in a single pass. The program is designed to allow reasonably free format of the source file.

## ADVEN-80 System Overall View

ADVEN-80 was a development of an original BASIC system[6] which relied heavily on disk storage. The BASIC program itself was suggested by a small program called GROW[3] (for those interested, the BASIC used was Microsoft Disk Extended Version 5.1 and the disks, Micropolis Metafloppy).

Anyone who wants to attempt to write their own Adventures in BASIC would be advised to start by reading Greg Hassett's article[8] and for those who want to start in machine code (any processor) and are capable of working from generalized macros, a good starting point would be Ken Reed's program[4].

Due mainly to the desire to re-do the system from scratch in Assembler, and with help from someone already familiar with the design of Adventure systems[4], ADVEN-80 was developed. The present version has several outstanding features, firstly the ability to handle multiple commands (i.e., GET LAMP THEN LIGHT IT). This example shows another feature, the system assumes that when you say "IT" you mean the item mentioned last.

ADVEN-80 is fairly flexible when it comes to entering English, (depending on the database). For example, you may enter GO SOUTH or simply S. You may say GET LAMP or TAKE THAT LAMP. If, for example, the player were to say RUB THE LAMP, a particular effect may occur. If it is necessary to immediately repeat such an action, he may either re-enter RUB THE LAMP, or if preferred, he may simply enter AGAIN. The idea for this was taken from a magazine article on ZORK[7]. On long inputs this can save a small amount of key pressing and so was considered to be a worthwhile addition to the system.

Due to a potentially fairly massive vocabulary, the system can make sense out of many sentences, but of course this must not be confused with real intelligence (as anyone actually creating the database could not fail to notice!!). So if you *try* to confuse the program, then you probably will succeed, so be sensible. . . . The present version was written under the M80 macro pre-processor[5] and assembled by ASMB, the Cromemco macro-assembler. The listing shown here has already been run through M80 and is suitable for most Z80 assemblers. (The macro pre-pass accounts for the excess labels.)

Readers who wish to use the programs in this article to create their own Adventures may find database construction made considerably easier by starting off with a very large piece of paper, drawing out all the locations as large "molecules." Connect them together, then make a list of all the interconnections, followed by all the items in the game, and where they are going to be at the start of the game. Finally, work through the game location by location, armed with as many sketches as possible, preferably with a friend known to be a whizz at finding faults in other people's programs . . . really, you won't get it right first time; you'll probably be able to work through a game by yourself with no problems, then along comes young Johnny, totally unaware that you're not *supposed* to try EATING the AXE. Hey, presto, another slight bug in your database discovered! Spend a lot of time thinking out the game, and twice as much time debugging it; but most of all, enjoy yourself. DDJ

**(LISTING ON PAGE 14)**

## References


[1] Scott Adams. Adventures available for TRS-80, Sorcerer, Apple, etc.

[2] Mostek Z80 manual.

[3] "Grow" by Jeff Levinsky. *Creative Computing* magazine.

[4] "Adventure II" by K. Reed. *Practical Computing* (Aug. 1980).

[5] *ML80: A Structured Machine-Oriented Microcomputer Language*. By L. R. B. Pedroso. AD/A-010 055.

[6] "Fantasia" by P. Scargill. A BASIC Adventure system. *Creative Computing* (1981).

[7] "How To Fit a Large Program Into a Small Machine" by Marc S. Blank and S. W. Galley. *Creative Computing* (July 1980).

[8] "How To Write an Adventure" by Gregg Hassett. *Creative Computing* (July 1980).



If you think you have an idea that would be of interest to *DDJ* readers, we would like to discuss it with you. Contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# Adven-80 (Text begins on page 8)

## Listing I

```
; ============================================================
;       ADVEN-80
;                   An Adventure Game System
;          Designed for and implemented in Z80 Microprocessor
;              Assembly Language.  AUGUST'80 TO APRIL'81
;
;                Released for NON-COMMERCIAL USE ONLY
;                      1981 Peter D. Scargill
;
; ============================================================
;
;     ASSEMBLY NOTES  :
;
;     The Assembler used to produce the object module was ASMB
;     the Cromemco Macro Assembler. The present  object module
;     runs to under 4K, and may be  originated at  any address
;     in system RAM memory.
;
; ============================================================
;
;     This program was inspired by Scott Adams Adventure series
;     and the design was influenced by several ideas  generated
;     with the help of K. Reed of S.P.L. International.
;
; ============================================================
;
;
        ORG     100H    ;Chosen for CP/M convenience.
        JP      RUNB    ;Skip reserved RAM area.
;
;CONDITION EQUATES       (Must not be changed or re-ordered)
;
CL      EQU     1       ;Current Location xx
NCL     EQU     2       ;Current location not xx
OP      EQU     3       ;Object xx present (carried/worn etc)
OC      EQU     4       ;Object xx carried
OW      EQU     5       ;Object xx worn
OWC     EQU     6       ;Object xx carried/worn
OH      EQU     7       ;Object xx here but not carried/worn
ONP     EQU     8       ;Object xx not present
ONC     EQU     9       ;Object xx not carried
ONW     EQU     10      ;Object xx not carried
NWC     EQU     11      ;Object xx not worn
ONH     EQU     12      ;Object xx not here, maybe carried/worn
OE      EQU     13      ;Object xx exists
ONE     EQU     14      ;Object xx does not exist
@L      EQU     15      ;Object xx is at location yy
N@L     EQU     16      ;Object xx not at location yy
RND     EQU     17      ;Random xx where xx 1 to 254 inclusive
TF      EQU     18      ;Flag xx = yy
NTF     EQU     19      ;Flag xx <> yy
GE      EQU     20      ;Flag xx >= yy
LT      EQU     21      ;Flag xx < yy
C2      EQU     22      ;2nd word = xx
C3      EQU     23      ;3rd word = xx


NC2     EQU     24      ;2nd word not = xx
NC3     EQU     25      ;3rd word not = xx
;
; ACTION EQUATES
;
INV     EQU     1       ;Print inventory of carried/worn items
TKE     EQU     2       ;Get xx to carry
WR      EQU     3       ;Get xx to wear
DR      EQU     4       ;Drop xx
PR      EQU     5       ;Print UMESS(xx)
PRM     EQU     6       ;Print MESS(xx)
DL      EQU     7       ;Describe the location xx
SF      EQU     8       ;Set flag xx= yy
DSCOB   EQU     9       ;Describe the object
SWP     EQU     10      ;Swap object xx with xx+1
MV      EQU     11      ;Go to location xx
OK      EQU     12      ;Say OK and leave table
QT      EQU     13      ;Quit, printing present score
INF     EQU     14      ;Increment flag xx
DEF     EQU     15      ;Decrement flag xx
EX      EQU     16      ;Exit table
SC      EQU     17      ;Print current score
PU      EQU     18      ;Put object xx into location yy
CR      EQU     19      ;Create object xx
DES     EQU     20      ;Destroy object xx
FLIP    EQU     21      ;Flip flag xx<>xx true-false
LA      EQU     22      ;Look around at items
SMA     EQU     23      ;Set up message xx
SMB     EQU     24      ;Set up message yy
WT      EQU     25      ;Wait for xx seconds (4MHZ)
PF      EQU     26      ;Print decimal value of flag xx
ATF     EQU     27      ;Add to flag then clear.
SFF     EQU     28      ;Subtract from flag then clear.
GRUN    EQU     29      ;Print Place, directions etc.
;
; These GLOBALS link this program module to the main database
;
        GLOBAL  OBJLOC  ;Object present location table.
        GLOBAL  OBJDES  ;Simple object description table.
        GLOBAL  MATCHW  ;Keyword match pair/Action table.
        GLOBAL  TITL1   ;Database title.
        GLOBAL  MSG     ;Standard system messages.
        GLOBAL  UMESS   ;Individual database messages.
        GLOBAL  TABKEY  ;Keyword/substitute table.
        GLOBAL  OBJDSK  ;Object special descriptions.
        GLOBAL  OBPROP  ;Properties of restricted objects.
        GLOBAL  LOC     ;Location table.
        GLOBAL  FLAG    ;Flag table
        GLOBAL  FL14    ;Refs within the table
        GLOBAL  FL20    ; ""       """""""""
        GLOBAL  FL21    ; ""       """"    "
;
;Storage areas and buffers
;
XNCO    DEFB    00      ;1st Optional Message
XBEFA   DEFB    00      ;2nd Optional Message
LSTOBJ  DEFB    00      ;Byte reserved for latest object name
BUFFER  DEFS    60      ;60 Byte Line Buffer
BLK4    DEFS    04      ;Temporary buffer
CODBUF  DEFB    00      ;Coded text store & batch buffer
        DEFS    14
CODB2   DEFB    00      ;2nd Buffer for 'AGAIN'
        DEFS    14
WAYOUT  DEFB    00      ;Temporary store for available exits.
        DEFS    30
RANSEED DEFS    01      ;Last random number store
CCOUNT  DEFB    00      ;Character count
SPEC    DEFB    00      ;Status of SPECIAL text.
        DEFS    16      ;STACK for 'SPECIAL' message system.
NSTORE  DEFS    4       ;Temporary store for BIN-ASCII
SAVIX   DEFB    00      ;Temporary store
TE1     DEFB    00      ;Negation flag for text
;
;General program equates. (Modify for different systems)
;
TRUE    EQU     -1
FALSE   EQU     00
;
CPM     DL      TRUE    ;Terminal currently set up for CP/M.
SPLIT   DL      FALSE   ;Split-screen system Active.
CECHO   DL      FALSE   ;Echo input chars on screen.
MMAP    DL      FALSE   ;Using a memory-mapped Device
VDWIDE  EQU     80      ;No. characters per line on my screen.
VDHT    EQU     24      ;No. lines on screen.
VSIZE   EQU     VDWIDE*VDHT
HEADLN  EQU     VDHT/2  ;Lines cleared for split top half.
CLEARS  EQU     26      ;Clear Screen on TVI920C
HOMEUP  EQU     30      ;Home up command on TVI920C
HOMEDN  EQU     31      ;Internal Memory mapped screen handler.
HALFCL  EQU     29      ; ""      """     "     "     "
MEMTOP  EQU     0F800H  ;For memory mapped screens, Top left.
SYSRET  EQU     0000    ;Return to CP/M
;
AGAIN   EQU     252     ;DO NOT change. For repeat line.
RTN     EQU     249     ;Special place, returns last location.
LINLEN  EQU     VDWIDE-10       ;Set for Maximum line length-10
THEN    EQU     250     ;Don't alter this. Duplicate in
                        ;database.
;
;     ********** ASSEMBLY-TIME CONSOLE MESSAGES **********
;
        CONMSG
        CONMSG  Configuring Adven-80 Adventure system.
        CONMSG
;
;     ************************************************************
;
;Start of ADVEN-80. First section prints out title info and
;sets up random number.
;


;
RUNB    LD      A,CLEARS        ;Initialise screen.
        CALL    COUT
        LD      HL,TITL1        ;Print heading blurb
        CALL    MPRINT          ;Routine to do the hard work
        CALL    CHRIN           ;Wait for any key
;
        LD      A,R             ;Completely random seed
        LD      (RANSEED),A     ;Save it in buffer
        LD      A,CLEARS        ;Clear my screen
        CALL    COUT
;
;
;Main program loop. Start by printing location description.
;
;
RUN     CALL    DESCR           ;Describe location & 'can see..
;
AFTDK   CALL    STABLE          ;Set up flags
        CALL    IBTCH           ;check if we are batching
        JR      Z,GTENTR
        CALL    BEFA            ;2nd optional message
GTENTR  CALL    ENTRY           ;Get user input.
        LD      A,R             ;Set up random number.
        LD      (RANSEED),A
;
SRTOUT  LD      IY,WAYOUT       ;Point to available exits
        LD      IX,FLAG         ;Point to flags
        LD      HL,CODBUF+1     ;Look @ 2nd word in code buffer
        LD      A,(HL)
        CP      251             ;Look for the code for 'IT'
        JR      NZ,QWOP
        LD      A,(LSTOBJ)      ;If found, get last object
        OR      A               ;Test Acc.
        LD      E,0
        JP      Z,ENDMAT        ;If no last object, Error.
        LD      (HL),A          ;otherwise save in buffer.
QWOP    DEC     HL              ;Point back to start of CODBUF
        LD      A,(HL)
        CP      13              ;See if this is a direction
        JR      NC,XMATCH
MOREDI  LD      A,(IY+0)        ;Compare with available exits
        OR      A
        JR      Z,XMATCH        ;Found a direction to go ???
        LD      A,(IY+1)
        CP      (HL)
        JR      Z,DFND
        INC     IY              ;Skip description of direction
```

```
        INC     IY
        INC     IY
        JR      MOREDI          ;Look for another exit
DFND    LD      A,(IY+2)
        CP      RTN             ;Is this a return
        JR      NZ,BASL
        LD      A,(IX+20)       ;If so, retrieve old location
BASL    PUSH    AF
        LD      A,(IX+14)
        LD      (IX+20),A       ;Do a swap.....
        POP     AF
        LD      (IX+14),A       ;Put new location in FLAG table
        JP      RUN             ;and go to it.
;
;
;Look to see if we are in the middle of a batch of commands
;and if so, set Zero flag else reset it.
;
IBTCH   PUSH    HL              ;Save HL
        LD      HL,CODBUF       ;Point to start of buffer
IBTPL   LD      A,(HL)
        OR      A               ;If you hit end, then no batch.
        JR      Z,NBTCH
        CP      THEN
        JR      Z,ZBTCH         ;If find THEN, is batch line.
        INC     HL
        JR      IBTPL
ZBTCH   LD      A,255           ;Set or Reset Zero flag as
NBTCH   INC     A               ;neccessary.
        POP     HL              ;Restore HL
        RET
;
;XMATCH
;This section looks at CODBUF to see if any sense can be made
;of the codes. (This is AFTER direction checks have been made.)
;if no sense then a request fo re-phraseing is made else action
;is taken.
;
XMATCH  LD      IX,CODBUF       ;Point IX to user input after
        EXX                     ;treatment, checking for
        LD      HL,OBPROP       ;legality in the case of an
FLNA    INC     HL              ;object. (Check OBPROP).
        LD      A,(HL)
        OR      A
        JP      Z,ENCHK         ;Not a special object ???
        CP      (IX+1)
        JP      Z,FNBYTE        ;Got as match.
FLNB    INC     HL              ;Otherwise skip to next.
        LD      A,(HL)
        OR      A
        JP      NZ,FLNB
        JP      FLNA
FNBYTE  INC     HL
        LD      A,(HL)
        OR      A
        JP      Z,ILEGL
        CP      (IX+0)          ;Check is legal.
        JP      Z,ENCHK
        JP      FNBYTE
ILEGL   EXX
        LD      HL,MSG          ;otherwise say not.


        LD      B,46
        JP      HOPM
ENCHK   EXX
;
        LD      A,(IX+1)        ;Save this byte
        LD      (SAVIX),A
        LD      IY,MATCHW       ;Set to start of table.
        LD      DE,00           ;Reset pointers
NEXTK   LD      A,(SAVIX)       ;Get byte back
        LD      (IX+1),A
        LD      A,(IY+0)
        OR      A
        JR      NZ,NOTENDM      ;If not end then skip.
ENDMAT  XOR     A
        OR      E               ;Check value in E.
;
        IF      SPLIT
        CONMSG  Split screen configuration
        JP      NZ,RUN          ;If command understood, simply
        ENDIF
;
        IF      NOT SPLIT
        CONMSG  Normal terminal configuration
        JP      NZ,AFTDK        ;Return for next input
        ENDIF
;
        IF      MMAP
        CONMSG  Memory-mapped output device.
        ENDIF
;
        LD      HL,CODBUF
        LD      A,(HL)
        CP      15              ;See if this was a direction.
        LD      HL,MSG
        JR      NC,DNFOL
        LD      B,21            ;Print "Not valid direction"
        JP      HOPM
DNFOL   LD      B,20            ; "Don't follow you"
HOPM    CALL    MFETCH
HOMU
;
        IF      SPLIT
        JP      RUN
        ENDIF
;
        IF      NOT SPLIT
        JP      AFTDK
        ENDIF
;
;First, check input pair. If either is 255 then this is treated
;as WILDCARD &  passed. Is necessary for certain CONDITION/
;ACTION sets that must be run after EVERY input.
```

```
;
NOTENDM LD      A,(IY+0)
        CP      255             ;Check 1st Char.

        JR      Z,CHSND
        CP      (IX+0)
        JR      NZ,FAILTHS      ;If no match then FAILED
CHSND   LD      A,(IY+1)
        CP      255             ;Another WILDCARD ???
        JR      Z,PASED
        CP      (IX+1)
        JR      Z,PASED
;
;This section sets the IY pointer to the start of a new set of
;conditions/actions.
;
FAILTHS INC     IY
FAILOOP INC     IY
        LD      A,(IY+0)
        CP      255
        JR      NZ,FAILOOP      ;Look for next section-ACTIONS
FAILLX  INC     IY
        LD      A,(IY+0)
        CP      255
        JR      NZ,FAILLX       ;Look for new set
        INC     IY
        JP      NEXTK           ;Look for another set to match
;
;This section looks at conditions in current list & determines
;whether or not these are all valid.
;
PASED   LD      E,1             ;Make flag non-zero
SKP1    INC     IY              ;Skip over Key set.
        INC     IY
        LD      A,(IY+0)
        CP      255     ;Check in case there are no more conds.
        JP      Z,ACTONIT       ;If not, go to action section.
;
;First of the conditions as outlined in the EQUATES section.
;
        CP      C2
        JR      NZ,LB0          ;Simple comparisons
        LD      A,(IY+1)
        CP      (IX+1)
HOPZ    JR      Z,SKP1          ;HOPZ is used as stepping stone
        JP      FAILTHS         ;Saves few bytes here & there.

LB0                             ;Forgive the excess labels,
                                ;turned out by the Macro Assm.
ISC3    CP      C3
        JR      NZ,LB1
        LD      A,(IY+1)
        CP      (IX+2)
        JR      HOPZ            ;Just saved 3 bytes ! (Gee)

LB1
ISNC2   CP      NC2
        JR      NZ,LB2
        LD      A,(IY+1)

        CP      (IX+1)
HOPNZ   JR      NZ,SKP1         ;Stepping stone for a different
        JP      FAILTHS         ;condition.
;
LB2
ISNC3   CP      NC3
        JR      NZ,LB3          ;All comparisons. No actions.
        LD      A,(IY+1)
        CP      (IX+2)
        JR      HOPNZ           ;Just saved another 3 bytes !!!
;
LB3
ISCL    CP      CL
        JR      NZ,LB4          ;Here comparing loc. number in
        LD      HL,FL14         ;pair buffer with present loc.
        LD      A,(HL)
        CP      (IY+1)          ;stored in flag 14.
        JP      HOPZ
;
LB4
ISNCL   CP      NCL
        JR      NZ,LB5          ;This as above but the outcome
        LD      HL,FL14         ;is the opposite.
        LD      A,(HL)
        CP      (IY+1)
        JP      HOPNZ
;
LB5
ISOPP   CP      RND             ;Are we talking here about
        JP      NC,IBRND        ;objects, or should we skip
        LD      A,(IY+1)        ;the next few comparisons.
        OR      A
        JR      NZ,CHZR
        LD      A,(IX+1)        ;Check if the user actually
        OR      A               ;did enter an object name.
        JR      NZ,NOTUSR
        LD      HL,MSG          ;If not, send a nasty message.
        LD      B,33
        JP      HOPM
;
CHZR    LD      (IX+1),A        ;He did, eh, let's go find it.
NOTUSR  LD      HL,OBJLOC
NTX1    LD      A,(HL)
        OR      A               ;If we reach end of the table
        JR      Z,CNFND         ;then no object.
        CP      (IX+1)
        JR      Z,OBFND         ;Found it !!
        INC     HL
        INC     HL
        JR      NTX1
CNFND   LD      HL,MSG          ;Can't find  thing. Tell him.
        LD      B,25
        JP      HOPM
OBFND   INC     HL              ;Put location in the D reg.
;
;If we are talking here of an object name entered by user, we
;must check to make sure that this is correct version of object
;in the case if a twin/state object such as a lamp.
;
        LD      A,(IY+1)        ;Get user input
        OR      A
        JR      NZ,ALAX
        LD      A,(HL)
        CP      252             ;Is it non-existant version ?
        JR      NZ,ALAX
        INC     HL              ;If so, go to existing version.
        INC     HL
;
ALAX    LD      D,(HL)
        LD      A,(IY+0)        ;Get the verb back.
;
ISOP    CP      OP
        JR      NZ,LB6          ;Continue looking as before.
        LD      HL,FL14
        LD      A,(HL)
        CP      D               ;Is it here ??
        JP      Z,SKP1
        LD      A,254
        CP      D               ;Is he carrying it ??
        JP      Z,SKP1
        LD      A,253           ;or wearing it ??
HOPZ1   CP      D
        JP      HOPZ
;
LB6
ISOC    CP      OC
        JR      NZ,LB7
        LD      A,254           ;Carrying only this time.
        JR      HOPZ1
;
LB7
ISOW    CP      OW
        JR      NZ,LB8
        LD      A,253           ;Wearing perhaps ???
        JR      HOPZ1
;
LB8
ISOH    CP      OH
        JR      NZ,LB9
        LD      HL,FL14
        LD      A,(HL)          ;Here only ??
        JR      HOPZ1
;
LB9
ISOWC   CP      OWC
        JR      NZ,LB10
        LD      A,254
        CP      D               ;Is object worn....
        JP      Z,SKP1
        DEC     A               ;or perhaps carried ???


        JR      HOPZ1
;
LB10
ISONP   CP      ONP
        JR      NZ,LB11         ;Now for opposites to above.
        LD      HL,FL14
        LD      A,(HL)
        CP      D
        JP      NZ,SKP1
        LD      A,254
        CP      D               ;Exactly as above but opposite
        JP      NZ,SKP1         ;result.
HOPNZ1  CP      D
        JP      HOPNZ
;
LB11
ISONC   CP      ONC
        JR      NZ,LB12         ;Is object NOT carried ??
        LD      A,254
        JP      HOPNZ1
;
LB12
ISONW   CP      ONW
        JR      NZ,LB13         ;Is it NOT worn ??
        LD      A,253
        JP      HOPNZ1
;
LB13
ISNWC   CP      NWC
        JR      NZ,LB14         ;NOT worn or carried ??
        LD      A,254
        CP      D
        JP      NZ,SKP1
        DEC     A
        JP      HOPNZ1
;
LB14
ISONH   CP      ONH
        JR      NZ,LB15         ;NOT here ??
        LD      HL,FL14
        LD      A,(HL)
        JP      HOPNZ1
;
LB15
ISOE    CP      OE
        JR      NZ,LB16         ;Does it even EXIST ????
        LD      A,255
        JP      HOPNZ1
```

```
;
LB16
ISONE   CP      ONE
        JR      NZ,LB17         ;and opposite question.
        LD      A,255
        JP      HOPZ1
;
LB17
ISAL    CP      AL
        JR      NZ,LB18         ;Is it anywhere in particular.
        LD      A,(IY+2)
        CP      D
        INC     IY
        JP      HOPZ
;
LB18
ISNAL   LD      A,(IY+2)        ;Or not.
        CP      D
        INC     IY
        JP      HOPNZ
;
ISRND   CP      RND
        JR      NZ,LB19         ;Looking for a random chance ??
RANDOM  CALL    RANM
        CP      (IY+1)
        JP      C,SKP1
        JP      FAILTHS
;
LB19
ISTF    PUSH    DE
        LD      HL,FLAG
        LD      E,(IY+1)
        LD      D,0
        ADD     HL,DE
        POP     DE
;
        CP      TF
        JR      NZ,LB20         ;Do we want to test a flag ??
        LD      A,(HL)
        CP      (IY+2)
        INC     IY
        JP      HOPZ
;
LB20
ISNTF   CP      NTF
        JR      NZ,LB21         ;Or test it for Not a value ??
        LD      A,(HL)
        CP      (IY+2)
        INC     IY
        JP      HOPNZ
;
LB21
ISGE    CP      GE
        JR      NZ,LB22         ;Greater than or equal test.
        LD      A,(HL)
        CP      (IY+2)
        INC     IY
        JP      Z,SKP1
        JP      NC,SKP1
        JP      FAILTHS
;
LB22


ISLT    CP      LT
        JR      NZ,LB23         ;Less than test.
        LD      A,(HL)
        CP      (IY+2)
        INC     IY
        JP      C,SKP1
        JP      FAILTHS
;
LB23
DANGER  LD      HL,MSG          ;reserved for emergencies.?
        LD      B,26
        CALL    MFETCH          ;and print an error message.
        JP      FAILTHS
;
;This is  ACTION section. This only reached assuming that
;all the above conditions (as applicable) have been met.
;
ACTONIT INC     IY              ;Skip over the delimiter.
        LD      A,(IY+0)
        CP      255             ;Get action key-check last one.
        JR      NZ,ACTO1        ;Ok, get next key set.
        INC     IY
        JP      NEXTK           ;Else finish.
;
ACTO1   CP      SMA
        JR      NZ,LB24         ;Set 1st optional message ???
        LD      A,(IY+1)
        LD      (XNCO),A
HOPC    INC     IY
        JP      ACTONIT
;
LB24
        CP      GRUN
        JR      NZ,LB25
        JP      RUN
;
LB25
APF     CP      PF
        JR      NZ,LB26         ;Print a flag value ??
        PUSH    HL
        PUSH    IX
        LD      E,(IY+1)
        LD      HL,FLAG
        LD      D,0
        ADD     HL,DE
        PUSH    HL
        POP     IX
        CALL    PNUMBER         ;This prints the decimal value.
        POP     IX
        POP     HL
        JR      HOPC
;
LB26
ASMB    CP      SMB
        JR      NZ,LB27         ;Set 2nd optional message ??

        LD      A,(IY+1)
        LD      (XBEFA),A
        JR      HOPC
;
LB27
AWT     CP      WT
        JR      NZ,LB28         ;Wait around, perhaps ??
        LD      B,(IY+1)
WTLP    PUSH    BC              ;A really simple delay loop.
        LD      B,255
WTLP1   PUSH    BC
        LD      B,255
LZLO
        PUSH    AF              ;Clumsy but it works.
        POP     AF
        PUSH    AF
        POP     AF
        DJNZ    LZLO
        POP     BC
        DJNZ    WTLP1
        POP     BC
        DJNZ    WTLP
        JP      HOPC
;
LB28
AINV    CP      INV
        JR      NZ,LB29         ;An inventory perhaps ??
        LD      C,0             ;Yes ?? well print carrying.
        CALL    ICANC
        LD      C,1             ;Also wearing.
        CALL    ICANC
        JP      ACTONIT
;
LB29
ALA     CP      LA
        JR      NZ,LB30         ;A look around maybe ??
        LD      C,2
        CALL    ICANC
        JP      ACTONIT
;
LB30
AEX     CP      EX
        JR      NZ,LB31         ;Exit the table maybe ??
        CALL    IBTCH
        JP      Z,AFTDK
;
        IF      SPLIT
        JP      RUN
        ENDIF
;
        IF NOT SPLIT
        JP      AFTDK
        ENDIF
;
LB31

AQT     CP      QT
        JR      NZ,LB32         ;Want to quit ...
        LD      HL,MSG          ;Quiting with...
        LD      B,27
        CALL    MFETCH
        PUSH    IX
        LD      IX,FLAG+13
        CALL    PNUMBER         ;decimal number
        POP     IX
        LD      B,28
        CALL    MFETCH          ;points.
;
        JP      SYSRET          ; EXIT HERE
;
LB32
AOK     CP      OK
        JR      NZ,LB33         ;Print OK and exit ??
        CALL    IBTCH
        JP      Z,AFTDK
        LD      HL,MSG
        LD      B,29
        JP      HOPM
;
LB33
ASC     CP      SC
        JR      NZ,LB34         ;Present score ??
        LD      HL,MSG
        PUSH    IX
        PUSH    IY
        PUSH    DE
        LD      B,30
        CALL    MFETCH
        POP     DE
        POP     IY
        POP     IX
        PUSH    IX
        LD      IX,FLAG+13
        CALL    PNUMBER         ;Just like quiting only
        POP     IX              ;different.....
        CALL    CRLF
        JP      ACTONIT
;
LB34
AMV     CP      MV
        JR      NZ,LB35         ;change locations, perhaps ??
        LD      HL,FL14
        LD      A,(IY+1)
        CP      RTN             ;Is this a return ???
        JR      NZ,AMV1
        LD      HL,FL20
        LD      A,(HL)
AMV1    PUSH    AF
        LD      HL,FL14
        LD      A,(HL)
        LD      HL,FL20
```

```
        LD      (HL),A
        LD      HL,FL14
        POP     AF
        LD      (HL),A          ;Simply enter up new location.
        JP      RUN
;
LB35
APR     CP      PR
        JR      NZ,LB36         ;Print a mess. from UMESS table
        LD      HL,UMESS
APRPL   LD      B,(IY+1)
        PUSH    IX
        PUSH    IY
        PUSH    DE
        CALL    MFETCH
        POP     DE
        POP     IY
        POP     IX
        JP      HOPC
;
LB36
APRM    CP      PRM
        JR      NZ,LB37         ;Or from MESS table. !!!!
        LD      HL,MSG
        JR      APRPL
;
LB37
AFLIP   LD      HL,FLAG         ;Talking about FLAGS now.
        PUSH    DE
        LD      D,0
        LD      E,(IY+1)
        ADD     HL,DE           ;HL points to relevant flag.
        POP     DE
;
        CP      FLIP
        JR      NZ,LB38         ;Flip a flag ??
        LD      A,(HL)
        OR      A
        JR      NZ,MAK0
        LD      A,1
        JR      MAK01
MAK0    XOR     A
MAK01   LD      (HL),A
        JP      HOPC
;
LB38
AIN     CP      INF
        JR      NZ,LB39         ;Increment it ??
        INC     (HL)
        JP      ACTONIT
;
LB39
ADEF    CP      DEF
        JR      NZ,LB40         ;Or decrement it ????
        DEC     (HL)

        JP      ACTONIT
;
LB40
ASF     CP      SF
        JR      NZ,LB41         ;Set to a new value ??
        LD      A,(IY+2)
        LD      (HL),A
        INC     IY
        JP      HOPC
;
LB41
        CP      ATF
        JR      NZ,LB42         ;Add to flag
        LD      A,(IY+2)
        LD      (IY+2),0        ;Wipeout value so it wont be
        ADD     (HL)            ;used again.
        JR      C,BRBXA
        LD      (HL),A
        JR      BXRL
BRBXA   LD      (HL),255        ;If >255 then make 255
BXRL    INC     IY
        JP      HOPC
;
LB42
        CP      SFF
        JR      NZ,LB43         ;Subtract from flag
        LD      A,(HL)
        SUB     (IY+2)
        LD      (IY+2),0
        JR      C,SEFAP
        LD      (HL),A
        JR      SAFAP
SEFAP   LD      (HL),0          ;If <0 then make 00
SAFAP   INC     IY
        JP      HOPC
;
LB43
ADL     CP      DL
        JR      NZ,LB44         ;Describe a location.
        LD      HL,FL14
        LD      B,(HL)
        LD      HL,LOC
        PUSH    IX
        PUSH    IY
        PUSH    DE
        CALL    MFETCH
        POP     DE
        POP     IY
        POP     IX
        JP      ACTONIT
;
LB44
ROBJ    LD      A,(IY+1)        ;PUT OBJ. NO. INTO (IX+1)
        OR      A
        JR      Z,BMISTHS
        LD      (IX+1),A
        LD      D,A
        JR      MISSTHS
;
BMISTHS LD      A,(IX+1)
        LD      D,A
        OR      A
        JR      NZ,MISSTHS
        LD      HL,MSG          ;Problems ?? print a message.
        LD      B,33
        JR      ACDANX
;
MISSTHS LD      HL,OBJLOC
MX5D    LD      A,(HL)
        OR      A
        JP      Z,ACDANGR       ;Can't find object ????
        CP      D
        JR      Z,FNDOBX
        INC     HL
        INC     HL
        JR      MX5D
FNDOBX  LD      A,(HL)          ;Save last object
        LD      (LSTOBJ),A
        INC     HL
;
;In  case of twin-state objects, check for the correct version.
;
        LD      A,(IY+1)
        OR      A
        JR      NZ,ALAXZ
        LD      A,(HL)
        CP      252
        JR      NZ,ALAXZ        ;If not correct, go on to
        INC     HL              ;the next one.
        INC     HL
;
ALAXZ   LD      D,(HL)          ;Put present place in D reg.
        JP      ANOTX
;
ACDANGR LD      HL,MSG          ;Problems-stop batching & print
        LD      B,31
ACDANX  JP      HOPM
;
ANOTX   LD      A,(IY+0)
;
        CP      DSCOB
        JR      NZ,LB45         ;Describe the object.
        LD      HL,OBJDSK
        INC     HL              ;Miss 1st zero
NOTHR   LD      A,(HL)
        OR      A
        JR      Z,NODSC
        CP      (IX+1)
        INC     HL              ;Wont set flags
        JP      NZ,NOXR

        CALL    MPRINT          ;Print relevant message
        JP      HOPC
NOXR    LD      A,(HL)
        INC     HL
        OR      A
        JR      Z,NOTHR
        JR      NOXR
NODSC   LD      HL,MSG
        LD      B,45
        JP      HOPM
;
LB45
        CP      SWP
        JR      NZ,LB46         ;Swap twin-state objects ??
        LD      A,(HL)
        PUSH    AF
        INC     HL              ;This is very clumsy.
        INC     HL
        LD      A,(HL)
        DEC     HL
        DEC     HL
        LD      (HL),A
        INC     HL
        INC     HL
        POP     AF
        LD      (HL),A
        DEC     HL
        DEC     HL
        JP      HOPC
;
LB46
AGET    CP      TKE
        JR      NZ,LB47         ;Want to take an object ???
        CALL    TOOMNY          ;Test to see if we are carrying
        LD      (HL),254        ;too much perhaps.
BRTSB   JP      HOPC
;
LB47
AWR     CP      WR
        JR      NZ,LB48         ;Wear an item, maybe..
        CALL    TOOMNY          ;Check again.
        LD      (HL),253
        JR      BRTSB
;
LB48
ADR     CP      DR
        JR      NZ,LB49         ;Drop something, thats easy.
ADR1    PUSH    IX
        LD      IX,FLAG
        LD      A,(IX+14)
        LD      (HL),A
        POP     IX
        JR      BRTSB
;
LB49
```

```
APU     CP      PU
        JR      NZ,LB50         ;Transfer an item to a new loc.
        LD      A,(IY+2)
        LD      (HL),A
        INC     IY
        JR      BRTSB
;
LB50
ACR     CP      CR              ;Create something, even.
        JR      Z,ADR1
;
        CP      DES
        JR      NZ,LB51
        LD      (HL),255
        JR      BRTSB
;
LB51
DANGA   LD      HL,MSG          ;Problems. Print error message.
        LD      B,32
        JP      HOPM
;
;Check here to see if we are carrying more items than entry in
;the flag table.
;
TOOMNY  PUSH    IX              ;Simple totalling.
        LD      IX,FLAG
        LD      A,(IX+2)
        ADD     A,(IX+18)       ;Total worn or carried.
        CP      (IX+3)
        JR      C,TOEMNY
        LD      HL,MSG
        LD      B,38
        CALL    MFETCH          ;If over, print 'Cant carry'
        POP     IX
        POP     IX
        DEC     IY
        JP      HOMU
TOEMNY  POP     IX
        RET
;
;Optional messages. If either is required, will be printed once
;only. If required again, may be set again using SMA and SMB
;commands in the action table.
;
ONCO    LD      A,(XNCO)        ;Load with last entry
        LD      B,A
        XOR     A
        LD      (XNCO),A
        JR      BEFAD
BEFA    LD      A,(XBEFA)
        LD      B,A
        XOR     A
        LD      (XBEFA),A
BEFAD   LD      A,B
        OR      A
```

```
        RET     Z               ;Return if no message.
        LD      HL,UMESS
        JP      MFETCH
;
;
;This routine prints out a decimal number pointed to by IX. The
;range is 0-255. This could probably be made a lot shorter.
;
;
PNUMBER PUSH    IX
        PUSH    IY
        LD      IY,NSTORE
        LD      (IY+0),0                ;put zeros in buffer
        LD      (IY+1),0
        LD      (IY+2),0
        LD      A,(IX+0)
        LD      (IY+3),A
BM1     SUB     100                     ;Hundreds
        JR      C,BM2
        INC     (IY+0)
        LD      (IY+3),A
        JR      BM1
BM2     LD      A,(IY+3)
        SUB     10                      ;tens
        JR      C,BM3
        INC     (IY+1)
        LD      (IY+3),A
        JR      BM2
BM3     LD      A,(IY+3)
        SUB     01                      ;units
        JR      C,BM4
        INC     (IY+2)
        LD      (IY+3),A
        JR      BM3
BM4     LD      A,(IY+0)
        OR      A
        JR      Z,BM5
        CALL    CCOUT           ;print with leading zero
BM5     LD      A,(IY+1)        ;suppressed on 1st digit.
        CALL    CCOUT
BM6     LD      A,(IY+2)
        CALL    CCOUT
        POP     IY
        POP     IX
        RET
;
CCOUT   OR      A               ;Simply add 30 to print.
        ADD     A,30H
        JP      COUT
;
;Random number gen. Nothing flash, but it works. Seeded from
;the refresh register at the start of the game and each
;move so it should be random.
;
RANM    LD      A,(RANSEED);number is returned in Acc.
        PUSH    BC
        LD      B,A
        LD      A,R
        ADD     B
        LD      (RANSEED),A
        POP     BC
        RET
;
;This routine describes location, and also all available exits
;together with a description of items that can be seen.
;
DESCR
        IF      SPLIT
        IF      NOT MMAP
        LD      A,HOMEUP                ;Clear 10 lines & home.
        CALL    COUT
        LD      B,HEADLN
RUNX0   PUSH    BC
        LD      A,' '
        LD      B,VDWIDE
RUNX1   CALL    CHROUT
        DJNZ    RUNX1
        POP     BC
        DJNZ    RUNX0
        LD      A,HOMEUP
        CALL    COUT
        ENDIF
        ENDIF
;
        IF      SPLIT
        IF      MMAP
        LD      A,HALFCL
        CALL    COUT
        ENDIF
        ENDIF
;
        LD      IX,FLAG         ;Point to Flag table
        CALL    CRLF            ;For neatness
        LD      HL,LOC          ;Set up pointer to place descr.
        LD      B,(IX+14)
        CALL    MFETCH          ;Print description of
                                ;current location
        LD      A,(IX+16)       ;Check for visibility
        OR      A               ;Test it
        JP      Z,EN1           ;and perhaps miss out dirs.
        LD      A,(IX+15)
        CP      OFEH            ;Do we print them ??
        JR      Z,EN1
        LD      HL,MSG          ;Ok, print the directions.
        LD      B,14
        CALL    MFETCH          ;Print :  EXITS ARE.....
        LD      DE,WAYOUT       ;Set up possibilities
WLOOP   LD      A,(DE)          ;and start looking
        INC     DE              ;Skip the rest.
        INC     DE
        INC     DE
```

```
        OR      A
        JR      Z,ENXX
        LD      B,A
        CALL    MFETCH          ;Print a possible direction
        JP      WLOOP           ;go back for some more
ENXX    CALL    CRLF            ;When finished, tidy up
EN1     LD      C,2
        CALL    ICANC           ;I can see.........
        CALL    ONCO            ;1st optional message;
;
        IF      SPLIT
        IF      NOT MMAP
        LD      A,HOMEUP
        CALL    COUT
        LD      B,VDHT-2
RUNX2   CALL    CRLF
        DJNZ    RUNX2
        ENDIF
        ENDIF
;
        IF      SPLIT
        IF      MMAP
        LD      A,HOMEDN
        CALL    COUT
        ENDIF
        ENDIF
;
        RET
;
;Here we take user input and convert into single byte tokens.
;Tokens placed in CODBUF.
;
ENTRY   LD      HL,CODBUF       ;Look first in case we
        LD      DE,CODBUF       ;are in the middle of a batch
        LD      A,(HL)          ;of commands.
        OR      A
        JR      Z,ENTRX
BUFFZ   LD      A,(HL)
        INC     HL
        OR      A
        JR      NZ,BUFFY
        LD      HL,CODBUF
        LD      (HL),A
        JR      ENTRY
BUFFY   CP      THEN
        JR      NZ,BUFFZ
BUFFY1  LD      A,(HL)
        INC     HL
        CP      THEN
        JR      Z,BUFFY1
        DEC     HL
        PUSH    HL
        LD      BC,0001
XXLDR   LD      A,(HL)
        OR      A
```

```
        JR      Z,XLDR
        INC     HL
        INC     BC
        JR      XXLDR
XLDR    POP     HL
        LDIR
        JP      SRTOUT
;
ENTRX   CALL    RANM            ;Set up specials (FL21 & 22)
        AND     01H             ;here. Make complimentary
        LD      HL,FL21         ;But random.
        LD      (HL),A
        XOR     01
        INC     HL
        LD      (HL),A
;
        LD      HL,CODBUF       ;Actual start. Zero 1st byte
        XOR     A               ;in code buffer.
        LD      (HL),A
        LD      HL,BUFFER       ;Point to entry buffer.
        LD      BC,0
        CALL    CRLF
        LD      A,'>'           ;Print a prompt.
        CALL    COUT
ENT1    CALL    CHRIN           ;Get a character.
        CP      0DH             ;End of line ??
        JR      Z,ENDIN
        CP      7FH             ;Backspace perhaps ??
        JR      Z,BACKS
        CP      5FH             ;BOTH types..
        JR      Z,BACKS
        CP      15H             ;CONT/U for scrap line.
        JR      NZ,CONVUP
        LD      A,'@'
        CALL    COUT
        JR      ENTRX           ;Try again.
CONVUP  OR      A
;
        IF      CECHO
        CALL    COUT            ;Not for CPM
        ENDIF
;
        BIT     6,A
        JP      Z,NOCONV        ;Check for upper case
        RES     5,A             ;If not, make it so.
NOCONV  CP      20H
        JR      Z,MISTRP
        CP      'A'
        JR      C,ENT1
MISTRP  LD      (HL),A          ;Save a good character.
        INC     HL
        INC     B
        LD      A,B
        CP      LINLEN          ;Get present count
        JR      NC,ENDIN


        JR      ENT1            ;Go back for another.
BACKS   LD      A,B
        OR      A
        JR      Z,ENT1
        DEC     HL              ;or perhaps backup one char.
        DEC     B
;
        IF      CECHO
        LD      A,08
        CALL    COUT            ;Print backspace.
        LD      A,20H
        CALL    COUT
        LD      A,08
        CALL    COUT
        ENDIF
;
        JP      ENT1
ENDIN   LD      (HL),0          ;End of line.
        CALL    CRLF
        LD      A,B
        OR      A
        JP      NZ,CON
;
        IF      SPLIT
        CALL    DESCR
        ENDIF
;
        JP      ENTRY
;
;This is where we convert from buffer into single byte tokens.
;
CON     LD      DE,BUFFER       ;Point to the input buffer
        LD      C,0
        LD      HL,CODBUF       ;and to the final code buffer.
MORC    LD      IY,TABKEY       ;Point to the lookup table.
MORX    CALL    FIND1ST
        JP      Z,ENDMESS
XX      LD      B,(IY+0)
        INC     IY
        LD      A,B
        OR      A
        JP      Z,MORC
MORXX   LD      A,(IX+0)
        CP      (IY+0)          ;Start the comparison.
        JR      NZ,MISS
        LD      A,(IX+1)
        CP      (IY+1)
        JR      NZ,MISS
        LD      A,(IX+2)
        CP      (IY+2)
        JR      NZ,MISS
        LD      A,(IY+3)
        AND     7FH             ;Reset 8th bit.
        CP      (IX+3)
        JR      NZ,MISS


GOTONE  LD      A,C             ;Is this 1st word ??
        OR      A
        JR      NZ,GHIR
        LD      A,B
;
        CP      AGAIN
        JR      NZ,LB52         ;Is the word 'AGAIN' ????
        LD      HL,CODB2        ;If so then retrieve old buffer
        LD      DE,CODBUF       ;and substitute into new buffer
        LD      BC,0015         ;and pretend you've just typed
        LDIR                    ;it all in again !!!!
;
LB52
GHIR    LD      (HL),B          ;If we get this far, success.
        INC     HL
        LD      C,1             ;Set flag to say we found
        JP      MORC            ;at least 1 word.
MISS    LD      A,(IY+3)        ;Check end of this block.
        INC     IY
        INC     IY
        INC     IY
        INC     IY
        OR      A
        JP      M,XX
        JP      MORXX
ENDMESS LD      (HL),0          ;Clear last 3
        INC     HL
        LD      (HL),0
        INC     HL
        LD      (HL),0
        LD      A,C
        OR      A
        JR      NZ,GOTALOT
        LD      HL,MSG
        LD      B,1
        CALL    MFETCH          ;Select 1st message - WHAT !!
;
        IF      SPLIT
        CALL    DESCR
        ENDIF
;
        JP      ENTRY           ;and try again
GOTALOT LD      HL,CODBUF       ;Save line in spare buffer.
        LD      DE,CODB2
        LD      BC,0015
        LDIR
        RET                     ;else return.
;
FIND1ST LD      A,(DE)          ;Get 1st 4 letters, and pad
        OR      A               ;if necessary.
        RET     Z
        CP      20H
        JR      NZ,BRP
        INC     DE
        JR      FIND1ST


;
BRP     PUSH    HL
        LD      IX,BLK4
        PUSH    IX
        POP     HL
        LD      (IX+0),20H      ;Firstly, empty buffer.
        LD      (IX+1),20H
        LD      (IX+2),20H
        LD      (IX+3),20H
        LD      B,4             ;4 chars.
PRP     LD      A,(DE)
        OR      A
        JR      Z,FX
        CP      20H             ;If space or zero, quit.
        JR      Z,FX
        LD      (HL),A
        INC     HL
        INC     DE
        DJNZ    PRP
BRT     LD      A,(DE)          ;Find end of word if > 4 chars.
        OR      A
        JR      Z,FX
        CP      20H
        JR      Z,FX
        INC     DE
        JR      BRT
FX      CP      0FFH            ;Zero flag must not be set
        POP     HL              ;on return.
        RET
;
;Character I/O section.
;
CRLF    EX      AF,AF'          ;Pretty obviously prints a CRLF
        XOR     A               ;sequence.
        LD      (CCOUNT),A      ;Also zeroes char. count.
        LD      A,0DH
        CALL    COUT
        LD      A,0AH
        CALL    COUT
        EX      AF,AF'
        RET
;
CHRIN
        IF      CPM             ;If using CP/M then will do
        CONMSG  This program uses CP/M I/O
        EXX
        LD      C,1
        CALL    0005
```

```
        EXX
        ENDIF
;
        IF      NOT CPM
        CONMSG  Non-CP/M version (Custom I/O)
CINN    IN      A,(00)          ;Typical input char in A.
        AND     02



        JR      Z,CINN
        IN      A,(01)
        AND     7FH
        ENDIF
;
        RET
;
COUT    CALL    CHROUT          ;Print char in Acc.
        OR      A
        PUSH    AF
        LD      A,(CCOUNT)
        INC     A               ;Inc. Char count.
        LD      (CCOUNT),A
        CP      LINLEN
        JR      C,BOX
        POP     AF
        PUSH    AF
        CP      21H             ;Space char + 1
        JR      NC,BODX
        CALL    CRLF
BODX    CP      08              ;Backspace
        JP      NZ,BOX
        LD      A,(CCOUNT)
        DEC     A
        DEC     A
        LD      (CCOUNT),A
BOX     POP     AF
        RET
;
CHROUT
        IF      CPM
        PUSH    AF
        EXX
        LD      C,2
        LD      E,A
        CALL    0005
        EXX
        POP     AF
        RET
        ENDIF
;
        IF NOT CPM
;SCREEN. A SIMPLE SCREEN HANDLER
;
        EXX                     ;Save all the registers
        PUSH    AF
        LD      HL,(VDMP)       ;Get current cursor position.
        CP      20H             ;Miss these checks if it's a
        JP      NC,ORCHAR       ;printing character...
        CP      0DH             ;Was input a C.R. ??
        JR      Z,CHART
        CP      08              ;or a BACKSPACE ??
        JR      Z,BKSP
        CP      CLEARS          ;or a CLEAR SCREEN ??
        JR      Z,CLEAR



        CP      HOMEUP          ;or a HOMEUP ??
        JR      Z,HMP
        CP      HOMEDN          ;or a HOMEDOWN ??
        JR      Z,HMDN
        CP      HALFCL          ;or a HALF SCREEN CLEAR
        JR      Z,HLFC
        CP      20H
        JR      C,SPEED         ;If a control character then
ORCHAR  LD      (HL),A          ;ignore it else print it.
        INC     HL
PTCOS   LD      (HL),0A0H       ;Increment pointer & place
        JR      LINOV           ;cursor there.
BKSP    LD      (HL),20H        ;For BS put in a space &
        DEC     HL              ;decrement the pointer.
        JR      PTCOS
CHART   LD      (HL),20H
        LD      A,L
        AND     0C0H
        ADD     40H
        LD      L,A
        LD      A,0
        ADC     H
        LD      H,A
LINOV   LD      (VDMP),HL
        LD      A,L
        AND     3FH
        JR      NZ,SPEED
        LD      A,H
        CP      0FCH            ;Next page after screen.
        JR      NZ,SPEED
        LD      HL,MEMTOP+VDWIDE
        LD      DE,MEMTOP
        LD      BC,VSIZE-VDWIDE
        LDIR
        LD      HL,MEMTOP+VSIZE-VDWIDE
        LD      (HL),20H
        LD      DE,MEMTOP+VSIZE-VDWIDE+1
        LD      BC,VDWIDE-1
        LDIR
SPORD   LD      HL,MEMTOP+VSIZE-VDWIDE
SPOOD   LD      (VDMP),HL
        LD      (HL),0A0H
SPEED   POP     AF
        EXX
        RET
;
HMP     LD      (HL),20H
        LD      HL,MEMTOP
        LD      (VDMP),HL
        JR      SPOOD
HMDN    LD      (HL),20H
        JR      SPORD
;
HLFC    LD      BC,VSIZE/2-1
        JR      CLEARQQ


;
CLEAR   LD      BC,VSIZE-1
CLEARQQ LD      HL,MEMTOP
        LD      (VDMP),HL
        LD      (HL),20H
        LD      DE,MEMTOP+1
        LDIR
        XOR     A
        OUT     (0CBH),A
        JP      SPEED
;
VDMP    DS      02
;
; ###       END OF HANDLER   ####
;
        ENDIF
;
; STABLE : This routine sets up the flags per the written
;          specification.
;
STABLE  LD      IX,FLAG         ;Point to flag table.
        INC     (IX+4)          ;These three are easy.
        INC     (IX+5)
        INC     (IX+6)
        LD      HL,FLAG+6
        LD      B,3
STAB1   INC     HL
        LD      A,(HL)
        OR      A
        JR      Z,STAB2
        DEC     (HL)
STAB2   DJNZ    STAB1
;
        LD      A,(IX+17)
        DEC     A
        AND     (IX+10)
        OR      (IX+1)
        LD      (IX+16),A       ;Save real light/dark.
        LD      A,(IX+17)
        DEC     A
        JR      NZ,STABQW
        LD      HL,FLAG+9
        LD      B,2
STAB4   INC     HL
        LD      A,(HL)
        OR      A
        JR      Z,STAB4A
        DEC     (HL)
STAB4A  DJNZ    STAB4
STABQW  LD      A,(IX+10)
        OR      A
        JR      NZ,STAB9
        INC     (IX+12)
        JR      STAB5
STAB9   LD      (IX+12),0



STAB5   LD      B,254
        CALL    STAB51
        LD      B,253
        LD      IX,FLAG+16
        CALL    STAB51
        LD      IX,FLAG+14
        LD      B,(IX+0)
        LD      IX,FLAG+17
STAB51  LD      (IX+2),0        ;Find out how many objects are
        LD      HL,OBJLOC       ;carried.
        INC     HL              ;Can't say OBJLOC+1 'cause it's
STAB5A  LD      A,(HL)          ;a Global.
        OR      A
        JR      Z,STAB6
        INC     HL
        INC     HL
        CP      B
        JR      NZ,STAB5A
        INC     (IX+2)
        JR      STAB5A
STAB6   LD      IX,FLAG
        RET
;
; MFETCH : This  routine selects and prints a message. HL must
;          point to the table, and B reg  contains a  number 1
;          to 255. The message is  selected and  printed. 0 is
;          the delimiter and '%' will echo as a CRLF sequence.
;          '>' acts to terminate batches but will not print.
;          ALSO HANDLES DESCRIPTIONS, STORING DIRECTION DATA.
;
; MPRINT : This may be used as a  seperate routine. HL  points
;          to first byte. delimiters as above.
;
MFETCH  PUSH    HL              ;Save registers
        PUSH    AF
        LD      A,0             ;Load Acc. with terminator
LOOPF   PUSH    BC              ;Save the count
        LD      BC,0000         ;Up to 64k !!!!
        CPIR                    ;Look for zeros..FAST.
        POP     BC              ;Get count back.
        DJNZ    LOOPF           ;Again ???
;
        LD      A,(HL)
        CP      0FCH            ;Is this location description.
        JP      C,MPRX          ;4 Different varieties.
        CP      0FEH            ;255=normal, 254=no directions
        JR      NC,MPML         ;253=dark
```

```
        INC     A               ;252=dark & no directions ever.
        INC     A
        LD      (IX+1),0        ;Save either dark....
        JR      MPM
MPML    LD      (IX+1),1        ;or light status.
MPM     PUSH    DE
        LD      DE,WAYOUT-1
        LD      (IX+15),A       ;Store SEE/NOT SEE directions


BRPX    INC     HL              ;in flag table + 15.
        INC     DE
        LD      A,(HL)
        CP      OFFH
        JR      Z,BLTX
        LD      (DE),A
        JR      BRPX
BLTX    PUSH    AF
        XOR     A
        LD      (DE),A
        POP     AF
        POP     DE
        INC     HL
        LD      A,(IX+17)
        DEC     A
        AND     (IX+10)
        OR      (IX+1)
        LD      (IX+16),A
        JP      Z,MPREND
        JR      MPRX
;
;Standard (well nearly) HL-POINTS-TO-MESSAGE type of routine.
;
MPRINT  PUSH    HL              ;Save Registers
        PUSH    AF
MPRX    LD      A,(HL)          ;Get a character
        OR      A               ;Test it
        JP      Z,MPREND        ;No more, perhaps ??
        INC     HL              ;ready for next char.
;
        CP      3EH
        JR      NZ,LB53         ; '>' Terminate batch ???
        XOR     A
        LD      (CODBUF),A
        JR      MPRX
;
LB53
        CP      25H
        JR      NZ,LB54         ; ' %' Looking for a CRLF ???
        CALL    CRLF
        JR      MPRX
;
LB54
        CP      7BH
        JR      NZ,LB55         ; '('Starting new conditional??
        LD      A,(HL)
        CP      '/'             ;Negative form ???
        JR      Z,FRTHR         ;Section added Feb
        XOR     A               ;'81 for negative
        LD      (TE1),A         ;flags.
        JR      FRTHR1
FRTHR   LD      A,1
        LD      (TE1),A
        INC     HL
FRTHR1  LD      A,(HL)



        PUSH    HL
        PUSH    BC
        PUSH    AF
        LD      HL,SPEC         ;Point to base
        INC     (HL)            ;Effectively PUSH stack
;
        LD      B,(HL)
LZL1                            ;Get new count in B
        INC     HL
        DJNZ    LZL1            ;Find new entry
;
        POP     AF              ;Retrieve FLAG No.
        SUB     62
        LD      (HL),A          ;Save it.
        POP     BC
        POP     HL
        INC     HL              ;Skip over No.
        JP      MPRX
;
LB55
        CP      7DH
        JR      NZ,LB56         ; ')' Ending a conditional ????
        LD      A,(SPEC)
        SUB     1
        JR      NC,SRXX
        XOR     A
SRXX    LD      (SPEC),A        ;Effectively POP stack.
        JP      MPRX
;
LB56
;Now to check flags pointed to by the various 'STACK' levels.
;To do this it is necessary to find offset into the FLAG table
;and then test the location, for EVERY level currently active.
;
;Oh, for a 6809 !!
;
        PUSH    AF              ;Check All levels for true
        LD      A,(SPEC)
        OR      A               ;Test DEPTH
        JP      Z,LOVEL         ;Miss out altogether if ZERO
        PUSH    HL              ;Else start testing !!!
        PUSH    BC
        LD      C,1             ;Assume true until otherwise.
        LD      B,A             ;Set no of levels in counter.
        LD      HL,SPEC         ;Set HL to bottom level-1.
LOOPQ   INC     HL
        LD      A,(HL)          ;Put FLAG offset in A for now.
        PUSH    HL              ;Inner loop.
        PUSH    BC
        LD      HL,FL20         ;HL to the flags-1.
        LD      B,A
LZL2                            ;Get LEVELS parameter from A
        INC     HL
        DJNZ    LZL2            ;Got correct flag now.
        LD      A,(HL)          ;Hold condition in A



        POP     BC
        POP     HL
        OR      A
        JR      NZ,ARX
        LD      C,A             ;If Zero, Kill C.
ARX     DJNZ    LOOPQ           ;Outer loop
        LD      A,C             ;Save C...
        POP     BC
        POP     HL              ;End outer loop
        OR      A               ;Test final condition
        JR      NZ,LOVELX
        LD      A,(TE1)         ;Check negation flag
        OR      A
        JR      Z,LOVELY
        JR      LOVEL
LOVELX  LD      A,(TE1)
        OR      A
        JR      Z,LOVEL
LOVELY  POP     AF              ;If zero, don't print.
        JP      MPRX
LOVEL   POP     AF
MPRZA   CALL    COUT            ;Actually print the char.
        JP      MPRX            ;then back for more.
MPREND  POP     AF
        POP     HL              ;Restore registers
        RET                     ;and return.
;
;This section deals with objects. The routine ICANC should be
;entered with the C register containing 0, 1 or 2 & the routine
;will print out what it can see according to the required mode.
;All registers are returned intact.
;The display will insert '&' between objects if more than 1.
;
;  Enter with the C register containing :-
;
;0 - I am carrying the following....
;1 - I am wearing the following.....
;2 - I can see the following........
;
;If no objects are found then 'nothing special' well be sent out.
;
ICANC   EX      AF,AF'          ;Save all registers.
        LD      A,C
        EXX
        LD      C,A
        PUSH    IX
        PUSH    IY
        LD      HL,MSG          ;Initial message.
        LD      IX,FLAG
        LD      A,(IX+16)
        OR      A
        JR      NZ,ICANQ
        LD      A,C
        CP      01
        JR      Z,ICAN3         ;Don't want 2 lots of dark!!!


        LD      B,15
        CALL    MFETCH          ;Send dark message.
        JR      ICAN3
ICANQ   LD      IY,OBJLOC
        PUSH    BC
        LD      A,16            ;Set to 'I am carrying..'
        ADD     A,C             ;and add offset.
        LD      C,A
        LD      B,C
        CALL    MFETCH          ;Print as appropriate.
;
        LD      HL,UMESS
        POP     BC
        DEC     C
        DEC     C
        DEC     C               ;Return true value.
        LD      A,C
        CP      255
        JR      NZ,ICANY
        LD      C,(IX+14)       ;Then load present location.
ICANX   LD      DE,0000
ICAN1   LD      D,(IY+0)
        LD      A,(IY+1)
        OR      A
        JR      Z,ICANZ
        INC     IY              ;On to next two
        INC     IY
        CP      C
        JR      NZ,ICAN1
;
        LD      A,E
        OR      A
        JR      Z,NOAND
        LD      A,' '
        CALL    COUT
        LD      A,'&'
        CALL    COUT
        LD      A,' '
```

```
        CALL    COUT
NOAND   PUSH    DE
        CALL    FINDMS
        POP     DE
        INC     E               ;Increment items found.
        JR      ICAN1
ICANZ   LD      A,E
        OR      A
        JR      NZ,ICAN2        ;If not zero then CRLF
        LD      HL,MSG
        LD      B,19
        CALL    MFETCH          ;else 'nothing special'
ICAN2   LD      A,'.'
        CALL    COUT
        CALL    CRLF
ICAN3   POP     IY
        POP     IX


        EXX
        EX      AF,AF'
        RET
;
ICANY   CP      254             ;This corrects a bit of bad
        JR      NZ,ICANZ1       ;design in the first place.
        LD      C,253
        JR      ICANX
ICANZ1  LD      C,254
        JR      ICANX
;
;
;This routine (FINDMS) looks thru the object table for object
;in D register and prints the correct message.
;
FINDMS  PUSH    IX
        LD      IX,OBJDES
FIND1   LD      A,(IX+0)
        LD      B,(IX+1)
        CP      D
        JR      Z,FIND2
        INC     IX
        INC     IX
        JR      FIND1
FIND2   POP     IX
        CALL    MFETCH
        RET
;
;
;The end. The database file is linked here.
```

End Listing

## Listing II

```
/* Released into public domain 1981 P. Scargill */

*********************************************************************
* Beginning of main database section, encoded entirely in          *
* the ADVEN language. File is run thru MAKEADV.COM then            *
* assembled and linked to the main program.                        *
*********************************************************************



*********************************************************************
*   OBJECT NAME/LOCATION  and OBJECT NAME/DESCRIPTION PAIRS         *
*   also PROPERTIES OF OBJECTS and OBJECT DETAILED INFO             *
*********************************************************************

OBJLOC     /* Object locations. The number following the object */
           /* keyword is the location number where the object   */
           /* will reside at the beginning of the game.         */

STATUE  4
KEY     8
MATCH   5
LAMP  252
ULAMP  11

ENDTABLE


OBJDES     /* Object descriptions. The number following the     */
           /* object keyword is the description number from the */
           /* UMESS table. All objects MUST have both LOCATION  */
           /* entries and DESCRIPTION entries.                  */

KEY     1
MATCH   2
LAMP    3
ULAMP   4
STATUE  6

ENDTABLE

OBPROP     /* Legal verbs associated with objects.                  */
           /* Where no properties shown, objects considered to have */
           /* universal properties. ie get, drop etc all work.      */
           /* Where properties are shown, only those are valid.     */

STATUE LOOK

ENDTABLE

OBJDSK     /* Where required, an object may have a special          */
           /* description. This is optional and may be as long as   */
           /* required, and may include conditionals as in the      */
           /* location tables.                                      */

KEY     'It's a shiny BRASS key with the name SHED on it...%'
MATCH   'It's just an ordinary match.'
        ' Quite useful for lighting lamps.%'
LAMP    'Looks like an oil lamp to me..%'
STATUE  'This is a statue of Aphrodite. '
        'It's covered in cobwebs.%'

ENDTABLE

*********************************************************************
*              WORDMATCH CONDITION/ACTION SETS                      *
*********************************************************************

MATCHW  /* Sets of match pairs, ie pairs of keywords, where      */
        /* either word may be the mask ANY, and the second word  */
        /* may be NUL (no word) or INP (replaced at runtime with */
        /* user input.)                                          */
        /* An optional TEST section may precede the action entry */
        /* following the rules of the CONDITION/ACTION tables.   */
        /* An ACTION entry is mandatory, and may allow dropping  */
        /* thru or termination as desired.                       */

MATCH  GET,ANY
TEST   OH,INP
ACTION TKE,INP,OK

MATCH  GET,NUL
ACTION PRM,39,EX

MATCH  GET,ANY
TEST   OWC,INP
ACTION PRM,44,EX

MATCH  GET,ANY
ACTION PRM,36,EX

MATCH  INVEN,NUL
ACTION INV,EX

MATCH  LOOK,AROUND
ACTION GRUN

MATCH  DROP,ANY
TEST   OWC,INP
ACTION DR,INP,OK

MATCH  DROP,ANY
ACTION PRM,35,EX

MATCH  LOOK,NUL
ACTION LA,EX

MATCH  LOOK,ANY
TEST   OP,INP
ACTION DSCOB,INP,EX

MATCH  QUIT,ANY
ACTION QT

MATCH  HELP,ANY
ACTION PRM,40,EX

MATCH  WHO,ANY
ACTION PRM,41,EX

MATCH  ON,LAMP
TEST   OC,ULAMP,OC,MATCH
ACTION SWP,LAMP,FLIP,17,OK

MATCH  ON,LAMP
TEST   OC,ULAMP
ACTION PR,05,EX

MATCH  OFF,LAMP
TEST   OC,LAMP
ACTION SWP,LAMP,FLIP,17,OK

MATCH  ON,LAMP
TEST   OC,LAMP
ACTION PRM,42,EX

MATCH  OFF,LAMP
TEST   OC,ULAMP
ACTION PRM,42,EX

MATCH  UNLOCK,ANY
TEST   TF,23,01,OC,KEY,CL,9
ACTION SF,23,00,OK

MATCH  LOCK,ANY
TEST   TF,23,00,OC,KEY,CL,9
ACTION SF,23,01,OK

MATCH  IN,ANY
TEST   TF,23,00,CL,9
ACTION MV,11

MATCH  YES,ANY
ACTION PRM,43

MATCH  SCORE,ANY
ACTION SC,EX

MATCHALL
ACTION PRM,37

ENDTABLE


*********************************************************************
*      Standard messages for system use mainly                      *
*********************************************************************

MSG  /* The positions of these messages MUST NOT be altered */
```

# Adven-80 (Listing continued, text begins on page 8)

```
1  '>That means nothing to me(@, I'm afraid)(?.) !!%'
2  ' North'
3  ' South'
4  ' East'
5  ' West'
6  ' Northeast'
7  ' Northwest'
8  ' Southeast'
9  ' Southwest'
10 ' Up'
11 ' Down'
12 ' In'
13 ' Out'
14 'Visible exits are '
15 '>It's dark, I can't see anything(? at all).'
   'It is (@very )dangerous to move in the dark.%'
16 'I am (@presently )carrying '
17 'I am wearing '
18 'I can see '
19 'nothing special'
20 '>I don't (@quite )follow you. Could you try '
   're-phraseing that....%'
21 '>I can(@no)(?')t go (@in that direction.)(?there.)%'
22 ' right'
23 ' left'
24 '>(?Ehh!! )That doesn't make sense(@, buddy) !!!!%'
25 '>(@I'm sorry, but )I can see no such object here.%'
26 '>PROGRAM ERROR :-INVALID CONDITION ;;;; %'
27 '>Quitting with '
28 ' points.%'
29 'Ok.%'
30 'Your(@ present) score is '
31 '>PROGRAM ERROR :- CANT FIND OBJECT FOR ACTION ;;;; %'
32 '>PROGRAM ERROR :- INVALID CONDITION %'
33 '>Try again...I'm not sure which object you mean.%'
34 '>I am not wearing this item.%'
35 '>This is not amongst the items I am carrying.%'
36 '>I can't see the object (@you mean )here....%'
37 '>(?Huh)(@Ehh) !!!!!!%'
38 '>Sorry, I can't (@possibly )carry anything else.%'
39 '>(?Sorry, I'm not sure from that what '
   'you want me to get)(@You want what) !!%'
40 'I can't help you (?at this time)(@I'm afraid)%'
41 '>Don't ask silly questions(@, please) !!%'
42 '>(@I believe I already did that !!%)'
   '(?It is already !!!!!!!!%)'
43 '>Sorry, no chance !!%'
44 '>But this is something I'm already carrying !!!%'
45 'Looks (@distinctly )ordinary to me !!!!%'


 6 '(?Sorry, that's quite impossible.)'
   '(@I'm sorry but I can't do that.)%'

 * You may expand here if necessary, but preferably */
 * use the UMESS table for user messages (max 246   */

ENDTABLE

/*********************************************************************
*    User defined messages, expandable up to 246                     *
*********************************************************************/

UMESS

1 'a set of keys'
2 'some matches'
3 'an oil lamp with a BLUE glow'
4 'an old worn out lamp'
5 'What with...(@ my fingernails ????)'     /* note use of */
  '(? I'm not carrying any matches !!!)%'   /* random func.*/
6 'an old dirty statue'

ENDTABLE
        ;



/*********************************************************************
*          LOCATION DESCRIPTIONS AND EXIT CODES                      *
*********************************************************************/

LOC     /* Locations with available exit table at start */

1  NORMAL,2,N,2,3,S,10
   'I am in the grounds of a derelict country mansion'
   ', it is a fine day and I can hear birds singing in'
   ' the trees.%'

2  NORMAL,12,IN,3,3,S,1,5,W,9
   'I am at the front of the mansion. There is no door'
   ' and I can see a long hallway inside.%'

3  NORMAL,22,RI,4,23,LE,5,10,U,6,13,EXIT,2
   'I am in a long Hallway. The light is dim here as there'
   ' has been no power here for a long time.%'

4  NOEXIT,13,EXIT,RTN
   'I am in an old study. There are cobwebs all around.%'

5  NOEXIT,13,EXIT,RTN
   'I've entered some sort of kitchen, by the look of it. '
   'There's a cooker of a kind, and a sink, but everything '
   'is broken and decayed.%'

6  NORMAL,11,DO,3,3,S,8,22,RI,7,23,LE,7,2,N,7,6,NE,7
   'I am on a wierd sort of landing, with exits in many '
   'directions. The floor doesn't look too safe to me!!%'

7  NOEXIT,13,EXIT,RTN
   'I am facing a boarded-up door.%'

8  NOEXIT,13,EXIT,RTN
   'I am in a pink bathroom. The bath has a big crack in it'
   ' and all the pipes have been long disconnected.%'

9  NORMAL,4,EA,2,5,W,10
   'I am standing outside an old (Alocked-up )wooden '
   'shed. (AThe windows have been boarded-up.)%'

10 NORMAL,2,N,1,4,EA,9,11,DO,12
   'I am standing in front of an old well. It looks like '
   'I could go down there if I wanted to.(A Of course I'd '
   'need some kind of lamp as it's probably very dark '
   'down there).....%'

11 NORMAL,13,EXIT,RTN
   'I am inside an old wooden shed.%'

12 DARK,4,W,13,5,EA,14,10,U,10
   'I am standing in a low, damp passageway.%'

13 DARK,5,EA,12,2,N,15
   'I am in a narrow passage, to the North is a dead end.% '

/* Expand up to 246 locations maximum */

ENDTABLE

/*********************************************************************
* KEYS. FIRST 15 ARE RESERVED...MUST NOT BE ALTERED. 'IT'=251 *
*********************************************************************/

TABKEY

N       'N   NORT'
S       'S   SOUT'
EA      'E   EAST'
W       'W   WEST'
NE      'NE  '
SE      'SE  '
NW      'NW  '
SW      'SW  '
U       'U   UP  '
DO      'D   DOWN'
LE      'L   LEFT'
RI      'R   RIGH'
IN      'IN  ENTE'
EXIT    'EXITLEAVOUT '


LOOK    'LOOKPERUEXAM'
INVEN   'INVE'
QUIT    'QUITEND STOPBYE ABOR'



GET     'GET GRABTAKECARRPICK'
DROP    'DROPREMO'
OPEN    'OPEN'
LAMP    'LAMP'
ULAMP
WEAR    'WEAR'
LIFT    'LIFTPUSHMOVEPULLYANK'
MATCH   'MATC'
SCORE   'SCOR'
AROUND  'AROUHERE'
SAY     'SAY YELLSHOUCALLSCRE'
OFF     'OFF UNLI'
ON      'ON  LIGHSTRIBURNALIGFIRE'
HELP    'HELPASSI'
BEGIN   'BEGISTAR'
CLOSE   'CLOSSHUT'
DOOR    'DOOR'
READ    'READ'
THROW   'THROHURLCHUC'
LOCK    'LOCK'
UNLOCK  'UNLO'
JUMP    'JUMP'
FORWARD 'FORW'
BACKWARD 'BACK'
BOARD   'NOTIBOARSIGN'
WHO     'WHO WHENWHATIS  WHERHOW WHY '
LISTEN  'LISTHEAR'
NOTE    'NOTE'
STUDY   'STUD'
KITCHEN 'KITC'
STAIRS  'UPSTSTAI'
ROOM    'ROOM'
BATH    'BATH'
KEY     'KEY KEYS'
SHED    'HUT SHED'
FRONT   'FRON'
GROUNDS 'GROU'
WELL    'WELL'
YES     'YES '
CARE    'CAREQUIECLOS'
STATUE  'STATFIGU'

 /* IMPORTANT - No more than 246 keys (As many alternatives */
 /*             as required, up to one line.                */

 ENDTABLE

 /*********************************************************************
 * SIGN-ON MESSAGE. THIS WILL NEED MINOR CHANGE FOR EACH ADV.  *
 *********************************************************************/
```

```
:ITL1

CHR$(26)
'Welcome to ADVE-80, a new type of ADVENTURE game. I will'

' be your eyes and ears in this game as we travel '
'together through an amazing fantasy world. '
'If you have any problems in getting me to '
'understand you, please re-phrase your '
'sentence. I cannot usually undertake to answer '
'questions but if you like you may ask '
'me for help. '
'Enjoy playing ADVEN-80, for further details '
'contact Pete Scargill on 0632-573905 (Great Britain) after 6 p.m. %'
'ADVEN-80 V.1.1  1981 P. Scargill. %'
'Hit any key to begin..... '

ENDTABLE

FLAG

1       01       /* If 00 then dark location */
2       00       /* No. of items carried. (AUTO) */
3       02       /* Max No. of objects can carry (AUTO) */
4       00      /* 3 Count up every turn (AUTO) */
5       00
6       00
7       00      /* 3 Count down every turn (AUTO) */
8       00
9       00
10      80      /* 2 Count down if dark & lamp on. (AUTO) */
11      00
12      00      /* Counts up if dark & lamp Empty (AUTO) */
13      00      /* Present Score (MANUAL UPDATE) */
14      01      /* Present Location (After 1st loc., AUTO) */
15      00      /* FF (normal) or FE (No directions) (AUTO) */
16      00      /* Determines if we can see. 00=BLIND.(AUTO) */
17      01      /* LAMP ON/OFF.  ON=00, OFF=01 (MANUAL) */
18      00      /* No. of items WORN (AUTO) */
19      00      /* No. of items HERE (AUTO) */
20      01      /* Last location store (AUTO) */
21      00      /* 2 Special Test Flags (AUTO) */
22      00      /* are set complimentary at random. */
23      01      /* 26 Flags (A-Z) for MANUAL text processing.*/
24      00
25      00      /* Will require init. for individual */
26      00      /* Adventures. */
27      00      /* Active any value other than 00.*/
28      00
29      00
30      00
31      00
32      00
33      00
34      00
35      00
36      00
37      00
38      00
39      00

40      00
41      00
42      00
43      00
44      00
45      00
46      00
47      00
48      00

/* maximum number is 246 - do not skip any numbers. */

ENDTABLE

ENDGAME
```

**End Listing**

## Listing III

```
10 PRINT:PRINT:PRINT:PRINT:PRINT
20 PRINT "### ADVEN    an ADVENTURE database conversion utility ###"
30 PRINT "............................................................"
40 DEFINT A-Z
50 CPM=0
60 PRINT
70 INPUT "Enter name of Database File (assumed ext .ADV)";FC$
80 IF FC$="" THEN PRINT "Invalid name":GOTO 70
90 IF INSTR(FC$,".")=0 THEN 110
100 FC$=LEFT$(FC$,INSTR(FC$,".")-1)
110 F$=FC$+".ADV"
120 MC$=FC$+".Z80"
130 TEST=0
140 PRINT "Input file = ";F$
150 PRINT "Output file= ";MC$
160 PRINT
170 OPEN "I",1,F$
180 OPEN "O",2,MC$
200 PRINT£2,"; ADVENTURE Z80 DATABASE SYSTEM"
210 PRINT£2,"; Released into the public domain 1981 P. Scargill"
220 PRINT£2,"; This file called ";MC$
230 PRINT£2,"; For ZILOG/MOSTEK Assemblers."
231 READ W$:IF W$<>"EOF" THEN PRINT£2,W$:GOTO 231
240 REM
250 LINE INPUT£1,Q$
260 GOSUB 1580
```

*(Continued on next page)*

Don't let your hot utilities
or computing ideas
sit on the shelf.
*DDJ* is a great place
for bright authors
to get a start.
Contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

```
270 IF Q$="" THEN 250
280 IF LEFT$(Q$,1)=" " THEN Q$=MID$(Q$,2):GOTO 280
290 IF LEFT$(Q$,1)=CHR$(9) THEN Q$=MID$(Q$,2):GOTO 280
300 IF LEFT$(Q$,1)="$" THEN 250
310 REM
320 IF RIGHT$(Q$,1)=" " THEN Q$=LEFT$(Q$,LEN(Q$)-1):GOTO 320
330 IF LEFT$(Q$,6)="OBJLOC" THEN TEST=TEST+1:GOTO 450
340 IF LEFT$(Q$,6)="OBJDES" THEN TEST=TEST+2:GOTO  510
350 IF LEFT$(Q$,6)="MATCHW" THEN TEST=TEST+4:GOTO 540
360 IF LEFT$(Q$,3)="MSG"    THEN TEST=TEST+8:GOTO 640
370 IF LEFT$(Q$,5)="UMESS"  THEN TEST=TEST+16:GOTO 740
380 IF LEFT$(Q$,3)="LOC"    THEN TEST=TEST+32: GOTO 780
390 IF LEFT$(Q$,6)="TABKEY" THEN TEST=TEST+64:GOTO 870
400 IF LEFT$(Q$,5)="TITL1"  THEN TEST=TEST+128:GOTO 1100
410 IF LEFT$(Q$,4)="FLAG" THEN TEST=TEST+256:GOTO 1170
415 IF LEFT$(Q$,6)="OBJDSK" THEN TEST=TEST+512:GOTO 1261
416 IF LEFT$(Q$,6)="OBPROP" THEN TEST=TEST+1024:GOTO 1091
420 IF LEFT$(Q$,8)="ENDGAME" AND TEST=2047 THEN PRINT"[EOF]  ALL CORRECT
    :CLOSE:GOTO 1620
430 IF LEFT$(Q$,8)<>"ENDGAME" THEN PRINT "BAD LINE":PRINT Q$:CLOSE
    :GOTO 1620
440 PRINT "MISSING SECTION MET AT END OF GAME":CLOSE:GOTO 1620
450 PRINT "Processing OBJLOC table."
460 PRINT£2,"OBJLOC";
470 GOSUB 1480
480 IF LEFT$(Q$,8)="ENDTABLE" THEN PRINT£2," DEFB 00,00":GOTO 250
485 MID$(Q$,INSTR(Q$," "),1)=","
490 PRINT£2," DEFB ";Q$
500 GOTO 470

510 PRINT "Processing OBJDES table."
520 PRINT£2,"OBJDES";
530 GOTO 470
540 PRINT "Processing MATCHW table."
550 PRINT£2,"MATCHW";
560 GOSUB 1480
570 IF LEFT$(Q$,8)="ENDTABLE" THEN GOTO 480
580 IF Q$="MATCHALL" THEN PRINT£2," DEFB 255,255":GOTO 560
590 IF LEFT$(Q$,5)<>"MATCH" AND LEFT$(Q$,4)<>"TEST" THEN 610
600 PRINT£2," DEFB ";MID$(Q$,INSTR(Q$," ")+1):GOTO 560
610 IF LEFT$(Q$,6)<>"ACTION" THEN CLOSE:PRINT "COND/ACTION FAULT"
    :PRINT Q$:GOTO 1620
620 PRINT£2," DEFB 255,";MID$(Q$,INSTR(Q$," ")+1);",255"
630 GOTO 560
640 PRINT "Processing MSG table."
650 PRINT£2,"MSG";
660 WE$="MES"
670 GOSUB 1480
680 IF LEFT$(Q$,8)="ENDTABLE" THEN GOTO 480
690 IF LEFT$(Q$,1)<>"'" THEN 710
700 PRINT£2,  " DEFB ";Q$:GOTO 670
710 PRINT£2," DEFB 00"
720 PRINT£2," DEFB ";MID$(Q$,INSTR(Q$," ")+1)
730 GOTO 670
740 PRINT "Processing UMESS table."
750 PRINT£2,"UMESS";
760 WE$="UMES"
770 GOTO 670
780 PRINT "Processing LOC table."
790 PRINT£2,"LOC";
800 GOSUB 1480
810 IF LEFT$(Q$,8)="ENDTABLE" THEN GOTO 480
820 IF LEFT$(Q$,1)<>"'" THEN 840
830 PRINT£2," DEFB ";Q$:GOTO 800
840 PRINT£2," DEFB 00"
850 PRINT£2," DEFB ";MID$(Q$,INSTR(Q$," ")+1);",255"
860 GOTO 800
870 PRINT "Processing TABKEY table."
880 CNT=0
890 PRINT£2,"TABKEY"
900 GOSUB 1480
910 IF LEFT$(Q$,8)="ENDTABLE" THEN 1030
920 IF INSTR(Q$," ")=0 THEN  CNT=CNT+1:PRINT£2,Q$;" EQU ";CNT
    :GOTO 900
930 Q1$=LEFT$(Q$,INSTR(Q$," ")-1)
940 Q$=MID$(Q$,INSTR(Q$," ")+1)
950 IF LEFT$(Q$,1)=" " THEN Q$=MID$(Q$,2):GOTO 950
960 CNT=CNT+1
970 PRINT£2,Q1$;" EQU ";CNT
980 PRINT£2,"  DEFB ";CNT
990 IF LEFT$(Q$,1)="'" AND RIGHT$(Q$,1)="'" THEN 1010
1000 CLOSE:PRINT "INCORRECT KEYS ";Q$:GOTO 1620
1010 PRINT£2,"  DEFM ";Q$
1020 GOTO 900
1030 PRINT£2,"THEN EQU 250":PRINT£2," DEFB 250"

1040 PRINT£2," DEFM 'THENAND ALSO'"
1050 PRINT£2,"IT EQU 251":PRINT£2," DEFB 251"
1060 PRINT£2," DEFM 'IT  THEM'"
1070 PRINT£2,"AGAIN EQU 252":PRINT£2," DEFB 252"
1080 PRINT£2," DEFM 'AGAINREPE'"
1090 GOTO 480
1091 PRINT "Processing OBPROP table."
1092 PRINT£2,"OBPROP";
1093 GOTO 1263
1100 PRINT "Processing TITL1 table."
1110 PRINT£2,"TITL1";
1120 GOSUB 1480
1130 IF LEFT$(Q$,8)="ENDTABLE" THEN GOTO 480
1140 IF LEFT$(Q$,5)<>"CHR$(" THEN 1160
1150 Q$=MID$(Q$,6):Q$=LEFT$(Q$,LEN(Q$)-1)
1160 PRINT£2," DEFB ";Q$:GOTO 1120
1170 PRINT "Processing FLAG table."
1180 PRINT£2,"FLAG DEFB 00"
1190 GOSUB 1480
1200 IF LEFT$(Q$,8)="ENDTABLE" THEN GOTO 480
1210 IF VAL(Q$)=0 THEN CLOSE:PRINT Q$:PRINT "INVALID FLAG"
    :GOTO 1620
1220 IF LEFT$(Q$,2)="14" THEN PRINT£2,"FL14";
1230 IF LEFT$(Q$,2)="20" THEN PRINT£2,"FL20";
1240 IF LEFT$(Q$,2)="21" THEN PRINT£2,"FL21";
1250 PRINT£2," DEFB ";MID$(Q$,INSTR(Q$," ")+1)
1260 GOTO 1190
1261 PRINT "Processing OBJDSK table."
1262 PRINT£2,"OBJDSK";
1263 GOSUB 1480 :IF LEFT$(Q$,1)<>"'" THEN PRINT£2," DEFB 00"
1265 IF LEFT$(Q$,8)="ENDTABLE" THEN GOTO 480
1266 MID$(Q$,INSTR(Q$," "),1)=","
1267 PRINT£2," DEFB ";Q$
1268 GOTO 1263
1270 DATA " GLOBAL OBJLOC"," GLOBAL OBJDES"," GLOBAL MATCHW"
1271 DATA " GLOBAL OBPROP"," GLOBAL OBJDSK"
1280 DATA " GLOBAL TITL1"," GLOBAL MSG"," GLOBAL UMESS"
1290 DATA " GLOBAL LOC"," GLOBAL TABKEY"," GLOBAL FLAG"
1300 DATA " GLOBAL FL14"," GLOBAL FL20"," GLOBAL FL21"
1310 DATA "RTN: EQU 249","ANY: EQU 255","INP: EQU 00"
1320 DATA "NUL: EQU 00","NORMAL: EQU 255","NOEXIT: EQU 254"
1330 DATA "DARK: EQU 253","DARKNO: EQU 252","CL: EQU 1"
1340 DATA "NCL: EQU 2","OP: EQU 3","OC: EQU 4","OW: EQU 5"
1350 DATA "OWC: EQU 6","OH: EQU 7","ONP: EQU 8","ONC: EQU 9"
1360 DATA "ONW: EQU 10","NWC: EQU 11","ONH: EQU 12","OE: EQU 13"
1370 DATA "ONE: EQU 14","ØL: EQU 15","NØL: EQU 16","RND: EQU 17"
1380 DATA "TF: EQU 18","NTF: EQU 19","GE: EQU 20","LT: EQU 21"
1390 DATA "C2: EQU 22","C3: EQU 23","NC2: EQU 24","NC3: EQU 25"
1400 DATA "INV: EQU 1","TKE: EQU 2","WR: EQU 3","DR: EQU 4"
1410 DATA "PR: EQU 5","PRM: EQU 6","DL: EQU 7","SF: EQU 8"
1420 DATA "DSCOB: EQU 9","SWP: EQU 10","MV: EQU 11","OK: EQU 12"
    ,"QT: EQU 13"
1430 DATA "INF: EQU 14","DEF: EQU 15","EX: EQU 16","SC: EQU 17"
1440 DATA "PU: EQU 18","CR: EQU 19","DES: EQU 20","FLIP: EQU 21"
1450 DATA "LA: EQU 22","SMA: EQU 23","SMB: EQU 24","WT: EQU 25"

1460 DATA "PF: EQU 26","ATF: EQU 27","SFF: EQU 28","GRUN: EQU 29"
1470 DATA "EOF"
1480 LINE INPUT£1,Q$
1490 GOSUB 1580
1500 IF LEFT$(Q$,1)=" " THEN Q$=MID$(Q$,2):GOTO 1500
1510 IF LEFT$(Q$,1)=CHR$(9) THEN Q$=MID$(Q$,2):GOTO 1500
1520 IF LEFT$(Q$,1)="$" THEN 1480
1530 IF INSTR(Q$,CHR$(9))=0 THEN 1550
1540 Q$=LEFT$(Q$,INSTR(Q$,CHR$(9))-1)+" "+MID$(Q$,INSTR(Q$,CHR$(9))+1)
    :GOTO 1530
1550 IF RIGHT$(Q$,1)=" " THEN Q$=LEFT$(Q$,LEN(Q$)-1):GOTO 1550
1560 IF Q$="" THEN 1480
1570 RETURN
1580 IF INSTR(Q$,"/$")=0 THEN RETURN
1590 IF INSTR(Q$,"$/")=0 THEN CLOSE:PRINT "BAD COMMENT"
    :PRINT Q$:GOTO 1620
1600 Q$=LEFT$(Q$,INSTR(Q$,"/$")-1)+MID$(Q$,INSTR(Q$,"$/")+2)
1610 GOTO 1580
1620 PRINT "Returning to CP/M operating system"
1630 END
```

**End Listing**

If you want to present your ground-breaking work to some of the most innovative and influential people in the microcomputing industry, *DDJ* can help. Contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# Terminal Programs for North Star

by Richard Blessing

Although I have had my North Star Horizon for almost three years, only recently have I had the opportunity to use it to communicate with another system. A change of employers has provided a Prime computer which may be accessed, via a modem, at 300 or 1200 baud. This was enough motivation for me to borrow a modem and write a routine that would allow me to use my North Star system as a terminal with which I could communicate with the Prime computer. At first, the results were very disappointing because characters were being dropped in almost every line received. On a hunch, I modified the North Star's initialization routine so it programmed the 8251 USART for one stop bit instead of two, and the problem disappeared.

The program listing titled MODEM.ASM (Listing I, p. 30) is an expanded version of my original program. At the suggestion of *DDJ*, a feature was added which computes the address of the user area I/O table and calls the various I/O routines using the jumps in this table. The user area address is computed by adding 700H to the CP/M BIOS address fetched from location 0001H. All of the I/O and status calls are passed to the subroutine JUMPS in MODEM.ASM. This base address for the user area is passed on the stack and the offsets for the I/O routines are passed by the calls to JUMPS using a technique I learned while working with the TI9900 and the RCA 1802. What would normally be the return address on the stack, instead points to a byte which is the offset into the jump table for the I/O routine being requested. The address of the user area is preserved as the first entry on the stack and the correct return address is PUSHed onto the stack by JUMPS. The offset is added to the base address in HL and a PCHL is executed to get to the correct I/O routine. When the I/O routine does a RET it will return to the address immediately following the byte which contained the offset.

The program MODEM.ASM loops between status checks of the console and the modem input ports and services either port when a character is presented. Communication is in the full duplex mode and since the Prime echos everything back to the sender, only the modem input is displayed on the console. The console input routine traps two characters which can be used to request special functions. An ESCAPE entered from the console will cause a JMP 0000H and reset the system, returning to CP/M. An FF (form feed), or control-L will toggle the printer on/off.

I used MODEM.ASM for about two weeks before I decided I needed a method for transferring disk files between the Prime and my North Star system. I wanted to be able to use the editors on my home system, which are much superior to the editor available on the Prime. Also, Pascal is supposed to soon be available on the Prime and I hope to develop programs on the North Star for use on the Prime. These reasons led to the development of the program MODEM.PAS (Listing II, p. 30).

MODEM.PAS is written in Pascal/MT+ and functions like MODEM.ASM with the additional ability to transfer disk files. The program relies on the interactive nature of the Prime to transfer disk files, in that it uses X_ON and X_OFF to control communications between the North Star and the Prime. Also, the Prime's DELAY command is used to request extra NULL's be sent at the end of each line to allow the X_OFF command to take effect. This enables MODEM.PAS to write to the disk file without missing any characters that might have been sent before the X_OFF was seen by the Prime.

To receive a file from the Prime, invoke the Prime's editor as though you were going to edit the file to be sent. Then enter an EOT at the console which informs MODEM.PAS a file is to be received and written to a disk file. MODEM.PAS will request a filename to use. Once you enter the filename everything received from the modem or console will be written to this file until another EOT is entered at the console. This requires some editing of the file once it is received to remove the editor commands used to get the file. However, it also allows you to send commands to the Prime or MODEM.PAS program and to concatenate files or abort the command. As I mentioned above, it is necessary to request extra NULL's at the end of each line by using the Prime's DELAY command. A <DELAY 20 40> seems to work very well.

Files are transmitted to the Prime using both MODEM.PAS and a small FORTRAN program called GET_F. A listing of GET_F appears in the top portion of the listing for MODEM.PAS. First start the execution of GET_F on the Prime and respond to the request for the filename with the Prime filename you want to write to. Next enter a SUB, or control-Z, at the console and answer the prompt for a filename with the name of the file you wish to send from the North Star. MODEM.PAS will display a "," for each line ACKed by GET_F and will display an "*" for each line not ACKed by GET_F. Lines which consist of only a CR LF will usually not get ACKed even though they are okay. Both programs will display a message when the EOF has been transmitted and received. Control is returned to the console so do *not* enter another SUB. This routine is much slower than the routine used to receive files because of the time required to process each character echoed by the Prime. A 40K file took two hours to transmit at 300 BAUD but it was a lot faster and less work that if I had to retype the file.

In addition to using MODEM.PAS in my work I have been using MODEM.PAS to communicate with RCPM's (Remote CP/M systems) and CBB's (Computer Bulletin Boards) around the country. I used it to transmit a copy of the 36K assembly listing of XMODEM from an RCPM in Chicago to my North Star system. XMODEM is a utility written by Keith Peterson for transmitting files between remote CP/M systems. I now use XMODEM to transfer files when talking to RCPM's, since it is available and more efficient than the routines I use with the Prime. I was unaware of the large amount of software available free on the RCPM's; this alone makes the entire project worthwhile.

I borrowed the modem used to develop the programs presented in this article and later purchased a used Omnitec 701B acoustic modem for $50. The used modem cost a sixth of what the borrowed modem cost and works just as well. Also, the Pascal/MT+ I used to write MODEM.PAS is an excellent product and I recommend it to anyone considering purchasing Pascal. DDJ

*Richard Blessing, R.R.4, Box 5, Geneseo, IL 61254.*

# Terminal Programs for N* (Text begins on page 28)

## Listing I

```
1:
2:
3:                      ;
4:                      ;               MODEM.ASM
5:                      ;
6:                      ;       A SIMPLE ROUTINE TO ALLOW THE N*
7:                      ;       TO ACT AS A DUMB TERMINAL USING A
8:                      ;       MODEM.
9:                      ;
10:                     ;
11: 0100                        ORG     0100H
12:                     ;
13:                     ;       OFFSETS INTO USER AREA OF CP/M FOR
14:                     ;       JUMPS TO EACH ROUTINE.
15:                     ;
16: 0000 =      CSTAT   EQU     0       ; CONSOLE STATUS
17:                     ;
18: 0003 =      CONIN   EQU     3       ; CONSOLE INPUT
19:                     ;
20: 0006 =      CONOUT  EQU     6       ; CONSOLE OUTPUT
21:                     ;
22: 0009 =      LIST    EQU     9       ; PRINTER
23:                     ;
24:                     ;
25:                     ;       THE MODEM IS CONNECTED TO N* SECOND SERIAL
26:                     ;       PORT.  YOU WILL HAVE TO SET UP YOUR USER
27:                     ;       AREA TO PROVIDE STATUS AND I/O ROUTINES
28:                     ;       IF THEY DO NOT ALREADY EXIST.
29:                     ;
30:                     ;
31: 000C =      MODOUT  EQU     12      ; MODEM OUTPUT
32:                     ;                 CP/M'S PUNCH
33:                     ;
34: 000F =      MODIN   EQU     15      ; MODEM INPUT
35:                     ;                 CP/M'S READER
36:                     ;
37: 0012 =      MSTAT   EQU     18      ; MODEM STATUS
38:                     ;                 CP/M'S PRINTER STATUS
39:                     ;
40: 0700 =      USER    EQU     0700H   ; OFFSET FROM BIOS TO USER AREA
41:                     ;
42:                     ;
43: 001B =      ESC     EQU     1BH     ; ESCAPE TO EXIT
44:                     ;
45: 000C =      FF      EQU     0CH     ; CONTROL-L TO TOGGLE PRINTER
46:                     ;                 ON / OFF
47:                     ;
48:                     ;       SET UP STACK AND TURN OFF PRINTER
49:                     ;       FLAG.
50:                     ;
51: 0100 318201 START:  LXI     SP,STACK
52: 0103 AF             XRA     A       ; CLEAR PRINT FLAG
53: 0104 326201         STA     PFLAG   ; SAVE PRINT FLAG
54:                     ;
55: 0107 2A0100         LHLD    0001H   ; GET ADDRESS OF BIOS
56: 010A 110007         LXI     D,USER  ; ADD OFFSET INTO USER
57: 010D 19             DAD     D       ; AREA FOR I/O ROUTINES
58: 010E E5             PUSH    H       ; SAVE ON STACK
59:                     ;
60:                     ;


61:                     ;       KEYBOARD INPUT ROUTINE
62:                     ;       TRAPS CONTROL-L TO TURN PRINTER
63:                     ;       FLAG ON/OFF OR AN ESCAPE
64:                     ;       TO EXIT AND RETURN TO CP/M
65:                     ;
66: 010F CD5101 CIN0:   CALL    JUMPS   ; CHECK FOR CHARACTER
67:                     ;                 FROM CONSOLE
68: 0112 00             DB      CSTAT   ; OFFSET TO CONSOLE STATUS
69:                     ;
70: 0113 B7             ORA     A
71: 0114 CA3401         JZ      CIN1    ; DON'T WAIT
72:                     ;                 ELSE
73: 0117 CD5101         CALL    JUMPS   ; GET CHARACTER
74:                     ;                 FROM CONSOLE
75: 011A 03             DB      CONIN   ; OFFSET TO CONSOLE INPUT
76:                     ;
77: 011B FE1B           CPI     ESC     ; TEST FOR ESCAPE
78: 011D CA5D01         JZ      EXIT    ; EXIT TO CP/M
79:                     ;
80: 0120 FE0C           CPI     FF      ; TEST FOR CONTROL-L
81:                     ;                 IF NOT A CONTROL-P
82: 0122 C22F01         JNZ     COUT1   ; SEND CHARACTER OUT MODEM
83:                     ;                 ELSE
84: 0125 3A6201         LDA     PFLAG   ; TOGGLE PRINTER FLAG
85: 0128 2F             CMA             ; ON/OFF
86: 0129 326201         STA     PFLAG   ; AND SAVE
87: 012C C33401         JMP     CIN1    ; CHECK FOR INPUT FROM MODEM
88:                     ;
89:                     ;       MODEM OUTPUT
90:                     ;

91: 012F 4F     COUT1:  MOV     C,A     ; SEND CHARACTER OUT
92: 0130 CD5101         CALL    JUMPS   ; MODEM PORT.
93: 0133 0C             DB      MODOUT  ; OFFSET TO MODEM OUTPUT
94:                     ;
95:                     ;       MODEM INPUT
96:                     ;
97: 0134 CD5101 CIN1:   CALL    JUMPS   ; CHECK FOR CHARACTER
98:                     ;                 FROM MODEM
99: 0137 12             DB      MSTAT   ; OFFSET TO MODEM STATUS
100:                    ;
101: 0138 B7            ORA     A
102: 0139 CA0F01        JZ      CIN0    ; DON'T WAIT FOR ONE !
103:                    ;
104: 013C CD5101        CALL    JUMPS   ; GET CHARACTER
105: 013F 0F            DB      MODIN   ; OFFSET TO MODEM INPUT
106:                    ;
107: 0140 4F            MOV     C,A     ; SAVE CHARACTER
108:                    ;
109:                    ;       CONSOLE OUTPUT
110:                    ;
111: 0141 3A6201 COU0:  LDA     PFLAG   ; TEST PRINTER FLAG
112: 0144 FE00          CPI     00H     ; TEST FLAG <0 = OFF, -1 = ON>
113:                    ;
114: 0146 C45101        CNZ     JUMPS   ; IF PRINTER FLAG IS ON
115:                    ;                 CALL PRINT ROUTINE.
116: 0149 09            DB      LIST    ; OFFSET TO LIST
117:                    ;
118:                    ;                THEN
119:                    ;
120: 014A CD5101        CALL    JUMPS   ; WRITE TO CONSOLE


121: 014D 06            DB      CONOUT  ; OFFSET TO CONSOLE OUTPUT
122:                    ;
123:                    ;
124: 014E C30F01        JMP     CIN0    ; LOOP
125:                    ;
126:                    ;
127:                    ;       THIS ROUTINE JUMPS INTO THE USER AREA
128:                    ;       JUMP TABLE.  IT LEAVES THE RETURN ADDRESS
129:                    ;       ON THE STACK FOR THE ROUTINE WHICH CALLED
130:                    ;       JUMPS.
131:                    ;
132: 0151 E1    JUMPS:  POP     H       ; GET ADDRESS OF OFFSET IN
133: 0152 7E            MOV     A,M     ; CALLING ROUTINE AND MOVE TO A
134:                    ;
135: 0153 23            INX     H       ; INCREMENT PAST OFFSET BYTE TO
136: 0154 EB            XCHG            ; CORRECT RETURN ADDRESS
137:                    ;
138: 0155 E1            POP     H       ; GET BASE ADDRESS OF USER TABLE
139: 0156 E5            PUSH    H       ; SAVE FOR NEXT TIME THRU
140:                    ;
141: 0157 D5            PUSH    D       ; PUT RETURN ADDRESS ON TOP
142:                    ;                 OF THE STACK
143:                    ;
144: 0158 1600          MVI     D,00H   ; COMPUTE JUMP ADDRESS BY ADDING
145: 015A 5F            MOV     E,A     ; OFFSET TO BASE ADDRESS
146: 015B 19            DAD     D
147:                    ;
148: 015C E9            PCHL            ; JUMP INTO USER TABLE
149:                    ;
150:                    ;
151: 015D 0E00  EXIT:   MVI     C,00H   ; RESET SYSTEM
152: 015F C30000        JMP     0000H   ; GOT AN ESCAPE FROM CONSOLE
153:                    ;
154:                    ;       STORAGE AREA
155:                    ;
156: 0162 =     PFLAG   EQU     $       ; PRINTER FLAG
157:                    ;
158: 0182 =     STACK   EQU     $+20H   ; A LITTLE SPACE FOR STORAGE
159:                    ;
160: 0162               END
```

End Listing

## Listing II

```
Pascal/MT+  Release 5.2  Copyright (c) 1980 MT MicroSYSTEMS  Page #   1
Compilation of: B:MODEM


Stmt  Nest    Source Statement

   1     0    (*
   2     0            Modem program
   3     0                 by
   4     0            Richard Blessing
   5     0            May 13,1981
   6     0
   7     0            This program is written in Pascal/MT+ and
   8     0            is being used on a North Star Horizon to
```

```
 9    0          communicate with a Prime computer.  In
10    0          addition to the operation as a terminal
11    0          the following commands are available:
12    0                  FF,
13    0                  Control-L - Turns printer listing ON/OFF.
14    0
15    0                  EOT,
16    0                  Control-D - Turns disk save ON/OFF.
17    0
18    0                  SUB,
19    0                  Control-Z - XMIT DISK <TEXT> FILE to Prime.
20    0
21    0                  CAN,
22    0                  Control-X - Returns to CP/M.
23    0   *)
24    0   (*$P*)


Stmt  Nest    Source Statement

25    0   (*
26    0
27    0          Note: The following FORTRAN program is executed
28    0               on the Prime when sending files to it.
29    0
30    0   ************************************************************************
31    0   *       PROGRAM GET_F                                                  *
32    0   *                                                                      *
33    0   *       PROGRAM TO RECEIVE FILES FROM N*                               *
34    0   *                                                                      *
35    0   *       CHARACTER FILENAME*16, RECORD*132, ACK*1                       *
36    0   *                                                                      *
37    0   *       DATA ACK/'006/                                                 *
38    0   *                                                                      *
39    0   *       WRITE(1,'(''ENTER NAME OF FILE TO BE WRITTEN'')')              *
40    0   *       READ(1,'(A16)') FILENAME                                       *
41    0   *                                                                      *
42    0   *       OPEN(7,FILE=FILENAME,ERR=900,IOSTAT=IER7)                      *
43    0   *                                                                      *
44    0   *       WRITE(1,(''START XMIT'')')                                     *
45    0   *                                                                      *
46    0   *  ******************* M A I N   L O O P ****************************** *
47    0   *                                                                      *
48    0   *  WHILE NOT EOF DO                                                    *
49    0   * 1     CONTINUE                                                       *
50    0   *                                                                      *
51    0   *         READ(1,'(A132)',END=2) RECORD  /* GET RECORD FROM N*        *
52    0   *                                                                      *
53    0   *         WRITE(7,'(A132)') RECORD  /* WRITE TO DISK FILE             *
54    0   *                                                                      *
55    0   *         WRITE(1,'(A1)') ACK  /* SEND ACKNOWLEDGE TO N*               *
56    0   *                                                                      *
57    0   *       GO TO 1                                                        *
58    0   *                                                                      *
59    0   *       RECEIVED EOF FROM N*                                           *
60    0   * 2     CONTINUE                                                       *
61    0   *       CLOSE (7)                                                      *
62    0   *       STOP                                                           *
63    0   *                                                                      *
64    0   * 900   CONTINUE                                                       *
65    0   *       WRITE(1,'(''ERROR OPENING UNIT 7 '',I8)') IER7                 *
66    0   *       STOP                                                           *
67    0   *                                                                      *
68    0   *       END                                                            *
69    0   ************************************************************************
70    0
71    0   *)
72    0   (*$P*)


Stmt  Nest    Source Statement

73    0   program MODEM_PAS;
74    0
75    0   const
76    1
77    1          CR = $0D;       LF = $0A;
78    1          FF = $0C;       EOT = $04;
79    1          CAN = $18;      X_OFF = $13;
80    1          X_ON = $11;     SUB = $1A;
81    1          ACK = $06;
82    1
83    1   (*     Port addresses and masks        *)
84    1          PORT0_STATUS = $03;   PORT1_STATUS = $05;   PORT2_STATUS = $06;
85    1          PORT0_OUT = $02;      PORT1_OUT = $04;      PORT2_OUT = $00;
86    1          PORT0_IN = $02;       PORT1_IN = $04;
87    1          IN_MASK = $02;        OUT_MASK = $01;
88    1
89    1          END_OF_FILE = $03;
90    1
91    1          NULL = $00;
92    1
93    1          ERROR = 255; (* Value returned by unsuccessful
94    1                          disk procedures and functions          *)
95    1
96    1   var
97    1          DISK1,
98    1          DISK2 : text;
99    1          CRLF : string[2];
100   1          BUFFER : string[132];
101   1          FILENAME : string[16];
102   1
103   1          KEY : byte;
104   1
105   1
106   1          PRINT_FLAG,
107   1          DISK_FLAG,
108   1          QUIT       : boolean;
109   1
110   1
111   1
112   1   (*$P*)

Stmt  Nest    Source Statement

113   1
114   1   procedure OUT_CONSOLE(KEY : byte);
115   1
116   1   begin
117   2     wait(PORT0_STATUS, OUT_MASK, false);
118   2
119   2     out[PORT0_OUT] := KEY
120   2   end;
121   1
122   1
123   1
124   1   procedure MODEM_IN(var KEY : byte);
125   1
126   1   begin
127   2     if (inp(PORT1_STATUS) & IN_MASK) <> 0 then
128   2         KEY := inp(PORT1_IN) & $7F
129   2     else
130   2       KEY := NULL
131   2   end;
132   1
133   1
134   1
135   1   procedure MODEM_OUT(KEY : byte);
136   1
137   1   begin
138   2     wait(PORT1_STATUS, OUT_MASK, false);
139   2
140   2     out[PORT1_OUT] := KEY;
141   2   end;
142   1   (*$P*)

Stmt  Nest    Source Statement

143   1
144   1
145   1   (* This procedure writes any partial buffers to the disk file
146   1      and then closes the file.
147   1   *)
148   1   procedure FLUSH_AND_CLOSE(var RESULT : integer);
149   1
150   1   var
151   2     I : integer;
152   2     WRITE_ERROR : boolean;
153   2
154   2   begin
155   2     MODEM_OUT(X_OFF); (* Stop xmit from Prime *)
156   2     BUFFER := concat(BUFFER,CRLF);
157   2
158   2     for I := 2 to length(BUFFER) do
159   2       begin
160   3         WRITE_ERROR := wnb(DISK1, BUFFER[I]);
161   3         if WRITE_ERROR  then
162   3           writeln('Error writing to disk');
163   3       end;
164   2
165   2     close(DISK1, RESULT);
166   2     if RESULT = ERROR then
167   2       writeln('Error closing ', FILENAME);
168   2     FILENAME := '';
169   2     MODEM_OUT(X_ON)  (* Restart xmit from Prime *)
170   2   end;
171   1   (*$P*)


Stmt  Nest    Source Statement

172   1   procedure DISK_RESET(var RESULT : integer);
173   1
174   1   begin
175   2     MODEM_OUT(X_OFF);
176   2       writeln('Enter name of file to be sent');
177   2       readln(FILENAME);
178   2       assign(DISK2, FILENAME);
179   2       reset(DISK2);
180   2       RESULT := IORESULT;
181   2     MODEM_OUT(X_ON)
182   2   end;
183   1
184   1
185   1
186   1   procedure DISK_REWRITE(var RESULT : integer);
187   1
```

```
188     1   begin
189     2         MODEM_OUT(X_OFF);
190     2           writeln('Enter filename');
191     2           readln(FILENAME);
192     2           assign(DISK1, FILENAME);
193     2           rewrite(DISK1);
194     2           RESULT := IORESULT;
195     2         MODEM_OUT(X_ON)
196     2   end;
197     1   (*$P*)
```

```
Stmt  Nest    Source Statement

198     1   procedure DISK_READ(var RESULT : integer);
199     1
200     1   var
201     2
202     2     DOT   : CHAR;
203     2
204     2     BYTE_OUT,
205     2     BYTE_IN : byte;
206     2     LINE_COUNT : integer;
207     2
208     2
209     2
210     2   (* This procedure is used to validate each byte as it is
211     2      ECHOed by the Prime.                                    *)
212     2
213     2       procedure GET_ECHO(var BYTE_IN : byte;
214     2                          BYTE_OUT : byte);
215     2       var
216     3        I : integer;
217     3
218     3       begin
219     3
220     3   (* Loop and give the echoed character a chance to get here *)
221     3         for I := 1 to 10000 do
222     3           begin
223     4             MODEM_IN(BYTE_IN);
224     4             if BYTE_IN <> NULL then
225     4               if (BYTE_IN = ACK) or (BYTE_IN = BYTE_OUT) then
226     4                 exit;
227     4           end;
228     3       end;
229     2
230     2
231     2   begin  (* DISK_READ *)
232     2     LINE_COUNT := 0;
233     2
234     2
235     2     repeat  (* For each line *)
236     3
237     3       repeat  (* For each character *)
238     4         BYTE_OUT := inb(DISK2);
239     4         if (BYTE_OUT <> $1A) then  (* Check for CP/M EOF *)
240     4             MODEM_OUT(BYTE_OUT);
241     4           GET_ECHO(BYTE_IN, BYTE_OUT);
242     4       until (BYTE_OUT = LF) or (BYTE_OUT = $1A);
243     3
244     3       GET_ECHO(BYTE_IN, BYTE_OUT);
245     3
246     3       (* If Prime ACK's the line sent print a '.', else print a '*'
247     3          to indicate a possible error in the line received
248     3       *)
249     3       if BYTE_IN = ACK then
250     3         DOT := '.'
251     3       else
```

```
Stmt  Nest    Source Statement

252     3           DOT := '*';
253     3
254     3       LINE_COUNT := LINE_COUNT + 1;
255     3       if (LINE_COUNT mod 50) = 0 then
256     3         writeln(DOT)
257     3       else
258     3         write(DOT);
259     3
260     3     until BYTE_OUT = $1A;
261     2
262     2     writeln('End of file reached ');
263     2     MODEM_OUT(END_OF_FILE);
264     2
265     2     close(DISK2, RESULT);
266     2     if RESULT = ERROR then
267     2       writeln('Error closing ',FILENAME)
268     2   end;
269     1
270     1
271     1
272     1
273     1   (*$P*)
```

```
Stmt  Nest    Source Statement

274     1
275     1   procedure DISK_WRITE(KEY : byte);
276     1
277     1   var
278     2     I : integer;
279     2     WRITE_ERROR : boolean;
280     2
281     2   begin
282     2     if KEY = CR then
283     2       begin
284     3           (* The CR is used as a flag and never gets concat to BUFFER.
285     3              Also, the LF is dropped while sending the X_OFF so both are
286     3              added to the line received.
287     3           *)
288     3         MODEM_OUT(X_OFF);
289     3         BUFFER := concat(BUFFER,CRLF);  (* We add them back here *)
290     3
291     3         for I := 2 to length(BUFFER) do
292     3           WRITE_ERROR := outb(DISK1, BUFFER[I]);
293     3           if WRITE_ERROR  then
294     3             writeln('Error writing to disk');
295     3         BUFFER := '';
296     3         MODEM_OUT(X_ON)
297     3       end
298     3     else
299     2         BUFFER := concat(BUFFER, chr(KEY));
300     2   end;
301     1
302     1
303     1
304     1   procedure PRINTER(KEY : byte);
305     1
306     1   begin
307     2     wait(PORT2_STATUS, OUT_MASK, false);
308     2
309     2     out[PORT2_OUT] := KEY;
310     2     out[PORT2_STATUS] := $20  (* Reset parallel port *)
311     2   end;
312     1   (*$P*)
```

```
Stmt  Nest    Source Statement

313     1   procedure IN_CONSOLE(var KEY : byte;
314     1                        var QUIT : boolean);
315     1
316     1   var
317     2     RESULT : integer;
318     2
319     2
320     2
321     2     begin
322     2     KEY := NULL;
323     2     if (inp(PORT0_STATUS) & IN_MASK) <> 0 then
324     2       KEY := inp(PORT0_IN) & $7F;
325     2
326     2     case KEY of
327     2
328     2       NULL            : ; (* NO INPUT *)
329     3
330     3       FF      : begin  (* Toggle print flag ON/OFF *)
331     4                        PRINT_FLAG := not(PRINT_FLAG);
332     4                        KEY := NULL
333     4                      end;
334     3
335     3       EOT     :        (* Open a disk file to receive incoming data
336     3                           or Flush and Close a disk file after receiving
337     3                           a file.                                    *)
338     3                    begin
339     4                      KEY := NULL;
340     4
341     4                      if DISK_FLAG then
342     4                        FLUSH_AND_CLOSE(RESULT)
343     4
344     4                      else
345     4
346     4                        begin
347     5                          DISK_REWRITE(RESULT);
348     5                          if RESULT = ERROR then
349     5                            writeln('Error rewriting ', FILENAME);
350     5
351     5                        end;
352     4                      DISK_FLAG := not(DISK_FLAG);
353     4                    end;
354     3
355     3
356     3
```

```
357   3       SUB :       (* Send a disk file to another system *)
358   3             begin
359   4               KEY := NULL;
360   4               DISK_RESET(RESULT);
361   4               if RESULT <> ERROR then
362   4                 DISK_READ(RESULT)
363   4
364   4               else
365   4
366   4                 writeln('Could not reset ',FILENAME)

Stmt  Nest    Source Statement

367   4             end;
368   3
369   3       CAN : QUIT := true;  (* Exit to CP/M *)
370   3
371   3     end
372   2   end;
373   1
374   1
375   1   (*$P*)
```

```
392   2
393   2         if QUIT then   (* QUIT set in IN_CONSOLE *)
394   2           exit;
395   2
396   2         if KEY <> NULL then
397   2           MODEM_OUT(KEY);
398   2
399   2         MODEM_IN(KEY);
400   2         if KEY <> NULL then
401   2           OUT_CONSOLE(KEY);
402   2
403   2         if KEY <> NULL then
404   2           begin
405   3             if DISK_FLAG then
406   3               DISK_WRITE(KEY);
407   3             if PRINT_FLAG then
408   3               PRINTER(KEY);
409   3           end;
410   2       until 1 <> 1;
411   1   end.
411   0   ---------------------------
411   0   Normal End of Input Reached
```

## End Listing

```
Stmt  Nest    Source Statement

376   1
377   1   (********** M A I N  P R O G R A M **********)
378   1
379   1   begin
380   1
381   1   (*  Load <cr>,<lf> into CRLF    *)
382   1
383   1           CRLF := concat(CRLF, chr(CR), chr(LF));
384   1
385   1
386   1     DISK_FLAG := false;   PRINT_FLAG := false;
387   1     QUIT := false;
388   1
389   1     repeat
390   2
391   2        IN_CONSOLE(KEY, QUIT);
```


Do you have a microcomputing manuscript
that seems too advanced for most magazines,
but too exciting for an academic journal?
*DDJ* may be the niche
you have been looking for.
Let us hear from you at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# Small VM

## Nucleus of a Portable Software Development Environment

by J. E. Hendrix

Portability is, or should be, a major concern of everyone developing software for microcomputers. The present variety of computers and operating systems, and the rate of new product announcements makes it highly desirable to write software which can be ported easily to foreign environments.

One approach to limiting the impact of porting software is to create a virtual operating system (VOS); i.e., a pseudo operating system which runs under and takes advantage of the services provided by a real operating system. This VOS would look the same to the user and to the programs running under it regardless of the particular, real operating system being used.

Application software is then written in a high level language under the VOS. Porting any amount of software is just a matter of developing another VOS for the new operating system and recompiling. If the new computer uses a different processor chip, then the compiler, assembler, and loader software must be revised or exchanged for different packages. Having a self-compiler is a real advantage.

To the user, the programs and the virtual operating system look the same. This limits the amount of retraining needed, as well as the amount of revision to system documentation.

A VOS may be viewed on three levels. At the lowest level is the interface to the real machine; this is called the virtual machine (VM). It is a collection of primitives for handling program control, memory management, and file management functions. The VM is written in assembly language and is the only part that must be rewritten when the VOS is ported. The second level is the command processor. It is written in a high level language and interfaces the user to the VM. The third level is the collection of programs needed to round out a complete software development environment. This includes text editors, compilers, loaders, file manipulation programs, text formatters, etc.

The VOS is invoked as an option so that the full capabilities of the real operating system are always available. This approach has been used to create UNIX-like (UNIX is a trademark of Bell Labs) environments on all sorts of systems from micros to mainframes. For a full discusssion of the VOS approach to software portability see "A Virtual Operating System" by Dennis E. Hall, Deborah K. Scherrer, and Joseph S. Sventak (*Communications of the ACM*, September 1980, volume 23, number 9).

*J. E. Hendrix, Rt. 1, Box 74-B-1, Oxford, MS 38655.*

### Overview

Small/VM is the first step in a project to develop a portable software development environment based on the VOS concept. The listing published with this article is copyrighted and is intended for private, non-commercial use only.

Small/VM is written for North Star DOS. It occupies 3K bytes (plus 537 bytes for every open file on DD systems, 281 for SD) and may be loaded with a program or operated co-resident with DOS at any convenient address. Small/VM provides program control, memory management, and file management services for programs written in Small/C or assembly language.

Small/VM is written for Allen Ashley's PDS macro assembler, and is compatible with North Star DOS 5.1 and later releases. It performs all I/O through standard entry points, making it compatible with whatever device drivers may be in use. It supports the following features:

- UNIX-like file management primitives
- automatic file creation and sizing
- random file access
- dynamic memory allocation
- UNIX-like command line argument passing
- UNIX-like redirection of standard I/O
- control of program execution

### Invoking Small/VM

Ordinarily, Small/VM resides on disk as a North Star "GO" file. It is invoked by a "GO VM" command either manually or automatically at boot time. After being invoked, it does some housekeeping and returns to DOS. Then it is ready to service programs which expect its presence in memory at a given address. I call this a resident copy of Small/VM. It may also be kept as a file of relocatable code to be loaded and linked with a program forming a self-contained unit. I call this a local copy of Small/VM.

### Memory Map

The only constraint on memory utilization is that some amount of free memory must follow the program for use by the machine stack and for allocation to the program. The location of the end of the program is passed to Small/VM at BOJ time, and the location of the end of free memory is a variable within Small/VM. Free memory is allocated from the end of the program toward high memory, and stack space starts at the end of free memory and grows toward the program.

### Calling Small/VM Routines

Except for CCBOJ, all Small/VM routines are called by pushing parameters onto the stack in the order they appear in a "C" function call, then executing a CALL to the appropriate routine. Each routine's calling sequence is documented in the listing. Some routines return a value in HL; others return zero for success. Minus 2 is an error flag indicating failure.

### File Concepts

Files are referenced by means of small integer values called file descriptors. They are assigned as shown in Table 1. The user of Small/VM views a file as simply a string of bytes. Blocking and deblocking is handled automatically, so there is no need to be concerned about the disk format. Files on disk look the

TABLE I

| fd | UNIX standard file | North Star device | default open mode | comment |
|---|---|---|---|---|
| 0 | stdin | 0 | read | console (redirectable) |
| 1 | stdout | 0 | write | console (redirectable) |
| 2 | stderr | 0 | read/write | console (fixed) |
| 3-9 | | 1-7 | read/write | 2nd serial I/O, etc. |
| 10-17 | | | closed | disk files |

same as I/O ports while reading and writing.

## Small/VM Entry Points

A jump table is located near the beginning of Small/VM to standardize entry points in resident copies. When Small/VM is loaded with a program, calls are made directly to the routines. CCEOM at the front of the table locates the end of free memory.

The following outline lists the Small/VM entry points alphabetically and summarizes the operation of each. The syntax for calling each routine is given as it would appear in a "C" program.

### A. CCALOC

1. Call
   address = CCALLOC(size)
   address: pointer to allocated block of memory
   size: number of bytes requested
2. Description
   CCALLOC reserves a block of free memory of the size requested. If insufficient space exists between the last allocated block and the stack, the program aborts with an allocation error indication (A?) on the console.

### B. CCAVAIL

1. Call
   size = CCAVAIL()
   size: number of bytes of free memory remaining
2. Description
   CCAVAIL calculates how many bytes of free memroy existed between the end of the last allocated block of memory and the top of the stack. If a negative value results an overlap condition exists, and the program aborts with a memory allocation error indication (A?) on the console.

### C. CCBOJ

1. Call
   This routine is called automatically at the beginning of the program. It is not called by the program explicitly. On entry, CCBOJ expects the address of the first byte following the program (first byte of free memory) to be in the DE register pair. The address at which CCBOJ is to return to the program must be on top of the stack. The following code brackets every program or sub-program compiled by Small/C:

```
        LXI   D,$+6
        JMP   CC9998
        .
        .
        .
CC9997: JMP   CCBOJ
CC9998: DS    6
        PUSH  D
        LXI   D,$+6
        JMP   CC9997
        ORG   CC9998
        JMP   $+6
        END
```

Any number of modules bracketed by this code and loaded together will call CCBOJ properly even though none of them knows where the program resides in memory or which position it occupies in the sequence of modules which make up the program. Every module but the last has the code following "JMP $+6" which loads at CC9998 overlayed by the prefix of the following module.

2. Description
   CCBOJ first decides whether the program is being started or continued. If it is being continued, CCBOJ restores all registers and reenters the program at the point of interruption. If it is being started, CCBOJ sets the machine stack to the nominal end of memory. Memory above this point may be dedicated to drivers or whatever.

   Next, CCBOJ scans the command line in DOS in order to establish I/O redirection and set up command line parameters to be passed to the program. The standard input file may be redirected from the default keyboard assignment by entering a "<" followed immediately by a file name or a file descriptor of an auxiliary I/O port. The standard output file may be redirected from the default console output device assignment by entering a ">" followed immediately by a file name or a file descriptor for an auxiliary I/O port. If the standard output file does not exist it is automatically created. If the standard input file does not exist the program is aborted with a redirection error indication (R?) on the console.

   Every string of non-blank characters following "GO" (except for I/O redirection specifications) are taken for parameters to the program. Each one is terminated with a null byte. A pointer to each one is pushed onto the stack. Finally, a count of the number of parameters is pushed, followed by a pointer to the first parameter pointer of the list. The first function of the program should be coded as follows in order to reference the command line parameters:

```
main(argc, argv) int argc, *argv;
```

   Finally CCBOJ pushes the address of CCEXIT onto the stack and reenters to the program. The final RET of the program transfers control to CCEXIT. A program may also explicitly call CCEXIT.

### D. CCCLOSE

1. Call
   status = CCCLOSE(fd)
   status: zero for success, else -2
   fd: file descriptor of file to be closed
2. Description
   CCCLOSE terminates access to a file by:
   a. flushing its buffer if necessary.
   b. setting the file's new size in the file directory, if the file is opened in write mode, is the last file on the diskette physically, and zero was given for a file size at open time.
   c. freeing memory used for the file control block.
   d. flagging the file descriptor block to indicate a closed status.

### E. CCEXIT

1. Call
   CCEXIT(errorcode)
   errorcode: a value indicating the nature of the exit
2. Description
   CCEXIT closes any open files. If errorcode is non-zero, its low order byte is displayed on the console followed by a question mark. Control is then returned to DOS.

### F. CCFILE

1. Call
   address = CCFILE(fd)
   address: address of the file control block associated with fd
   fd: file descriptor for the file in question
2. Description
   CCFILE tells the caller where to find the file control block for a specified file. The first 16 bytes are an in-core copy of the file's disk directory entry.

### G. CCFLUSH

1. Call
   status = CCFLUSH(fd)
   status: zero for success, else -2
   fd: file descriptor
2. Description
   CCFLUSH allows the program to force buffer flushing if changes have been made to the file. Nothing happens if the buffer contains no changes.

### H. CCFREE

1. Call
   status = CCFREE(address, size)
   status: zero for success, else address
   address: location of the block to be freed
   size: number of bytes to free
2. Description
   CCFREE frees the block of memory described by address and size only if it matches the end of allocated memory; otherwise, it takes no action and returns address.

### I. CCHALT

1. Call
   CCHALT()
2. Description
   This routine allows a program to interrupt itself. It saves the registers (including stack pointer) and returns to DOS. Processing may be resumed by entering a CTL-R (resident copy of Small/VM) or issuing a "JP" to the start of the program (local copy).

### J. CCMODE

1. Call
   mode = CCMODE(fd)
   mode: 0 if closed
   1 if read only
   2 if write only
   3 if read/write
   fd: file descriptor
2. Description
   CCMODE indicates the open-mode of a file.

### K. CCOPEN

1. Call
   status = CCOPEN(fd, name, mode, size, type, tdi)
   status: zero for success, else -2
   fd: file descriptor to be assigned to the file
   name: address of null terminated character string containing the name in North Star format of the file to be opened
   mode: 0 – allocate a file control block only
   1 – read mode
   2 – write mode
   3 – read/write mode
   size: size of new file in disk blocks
   type: North Star file type
   tdi: North Star type dependent information (first 2 bytes)
2. Description
   CCOPEN opens disk files, making them accessible to the program. If write mode is specified, non-existing files are created. If size is zero, write mode is given, and the file is either new or the last file on the diskette physically, then its size is set automatically at close time. Type and tdi go into the file directory entry of a new file. These North Star dependent parameters do not create a significant portability problem since in the Small/C environment, this routine is usually called indirectly through the fopen routine in the standard I/O library. Fopen supplies zeros for them. Their presence permits some programs greater freedom in performing file manipulation functions.

### L. CCPOLL

1. Call
   CCPOLL(flag)
   flag: 0 – disable operator interrupts
   1 – enable operator interrupts and poll for one
2. Description
   Two types of interrupts are recognized by CCPOLL. A CTL-C causes the program to return to DOS after preserving its registers; it may be resumed at a later time. Any other unsolicited character from the keyboard causes the program to pause until a second CTL-C character is entered. One CCPOLL(0) call makes the program uninterruptable. CCPOLL(1) calls should be placed in major loops within the program to permit full operator control.

### M. CCPURGE

1. Call
   status = CCPURGE(name)
   status: zero for success, else -2
   name: address of null terminated character string containing the name of the file in North Star format
2. Description
   CCPURGE locates the specified file on the specified or default drive and removes its entry from the directory.

### N. CCREAD

1. Call
   count = CCREAD(fd, recaddr, recsz)
   count: number of bytes actually read or -2 on failure
   fd: file descriptor of an open file
   recaddr: address of first byte of program's record buffer
   recsz: number of bytes requested
2. Description
   CCREAD fetches the next string of bytes of the length specified from the file indicated. If fewer than the requested number of bytes remain in the file, then as many as remain are transferred to recaddr. The actual number of bytes transferred is returned to the caller – zero at end of file. Successive reads transfer data from consecutive byte locations in the file. CCSEEK may be used to alter the sequence of reading.

### O. CCSEEK

1. Call
   status = CCSEEK(fd, disp, base)
   status: zero for success, else -2
   fd: file descriptor
   disp: displacement (32 bits unsigned)
   base: determines type of displacement
   0 – bytes from current position
   1 to 65535 – size in bytes of records to displace from the beginning of the file
2. Description
   CCSEEK relocates the current position within a file to a new byte address. If the new position is outside the current buffer, the target sector is read into memory. If the current buffer contains changed data it is flushed before reading the new sector. Presently Small/C does not support "long" variables, so the displacement may be passed to CCSEEK as two 16-bit values; e.g., a zero followed by a 16-bit displacement. CCSEEK may be used only with files opened for read or read/write; i.e., previously existing files.

### P. CCWRITE

1. Call
   count = CCWRITE(fd, recaddr, recsz)
   count: number of bytes actually written
   fd: file descriptor
   recaddr: address of the record to be written

recsz: number of bytes to write

2. Description
   CCWRITE writes the number of bytes specified by recsz from recaddr to the file indicated by fd. Successive writes transfer data to consecutive byte locations in the file. CCSEEK may be used to alter the sequence of writing, but only if the file is open in read/write mode.

## Conclusion

Small/VM adds file management, memory management, and program management functions to North Star DOS. The ability to alter standard input and output assignments at execution time and to pass command line arguments to programs makes it possible to develop programs in Small/C which give the North Star computer a definite UNIX flavor. To date, I have implemented most of the programs described in the book *Software Tools* by Kernighan and Plauger. In fact, this article was prepared with the software tools editor and text formatter. Small/C required some revision to its expression analyzer to make it properly scale values added to and subtracted from pointers and array names; but has performed flawlessly since. I am well pleased with the performance of programs written in Small/C and with the ease with which they are developed and maintained. We owe a debt of gratitude to Ron Cain for placing his compiler in the public domain.

This has been a very abbreviated description of Small/VM. For those who wish to pursue it further, complete documentation (58 pages) is available for $10. The package is available in double density format, including documentation, for $35. Small/C is included at no charge. Single density users do not have enough disk space to compile programs as large as the compiler itself, but I will make the package available to them for $40 (two diskettes) if they are interested. Since I cannot afford the time to properly support the package, all source code is included. Experienced assembly language programmers should have no trouble providing their own support. ▶▶J

*DDJ's* responsive readership
can give you invaluable feedback
on your ideas and innovations.
Find out how you can reach them
by contacting us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**


**You Read Dr. Dobb's Journal And You Don't Subscribe?!**

**Save $13 off newsstand prices for 2 yrs. Save $5 for 1 yr.**

Can you afford to miss an issue with information vital to your interests? As a subscriber you can look forward to articles on Small-C, FORTH, CP/M, S-100, Compiler optimization, Concurrent Programming and more, delivered right to your door. And you'll never miss the issue that covers your project.

**DR. DOBB'S JOURNAL**
**P.O. Box E**
**Menlo Park, CA 94025**

**Yes!** Sign me up for ___2 yrs. $47 ___1 yr. $25

___ I enclose a check/money order
___ Charge my Visa, MasterCard, American Express
___ Please bill me later

Name _____
Address _____
_____
Credit Card _____ Exp. date _____
Acct No. _____
Signature _____

# Small VM (Text begins on page 34)

```
;
;*************************************************************
;*  Small/VM:  version 1.0                                   *
;*             A virtual-machine interface to North Star DOS*
;*             for Small/C and assembly language programs.  *
;*                                                          *
;*  Copyright  1981  J. E. Hendrix.                         *
;*             All rights reserved.                         *
;*                                                          *
;*  Written for PDS MAKRO assembler.                        *
;*************************************************************
;
;
;----- ENTRY POINTS
;
        JMP     TRAP            ;;SKIPPED IF LOADED WITH PROGRAM
        JMP     CC9998          ;;EXECUTED ONLY IF LOADED WITH PROGRAM
CCEOM:  ENTRY                   ;;NOMINAL END OF MEMORY
        DW      0B000H
        JMP     CCBOJ
        JMP     CCEXIT
        JMP     CCPOLL
        JMP     CCHALT
        JMP     CCOPEN
        JMP     CCCLOSE
        JMP     CCREAD
        JMP     CCWRITE
        JMP     CCSEEK
        JMP     CCMODE
        JMP     CCFILE
        JMP     CCPURGE
        JMP     CCFLUSH
        JMP     CCALLOC
        JMP     CCFREE
        JMP     CCAVAIL
;
; DIVERT CIN CALLS TO CINTRAP
;
TRAP:   LHLD    CIN+1
        SHLD    CINV            ;;SAVE CIN VECTOR
        LXI     H,CINTRAP
        SHLD    CIN+1
        JMP     DOS
;
;----- DOS ENTRY POINTS
;
DOS:    EQU     2028H           ;;WARM START
COUT:   EQU     200DH           ;;CHAR OUTPUT
CIN:    EQU     2010H           ;;CHAR INPUT
CONTC:  EQU     2016H           ;;CTL-C DETECT ROUTINE
DLOOK:  EQU     201CH           ;;DIRECTORY LOOKUP
DWRIT:  EQU     201FH           ;;DIRECTORY REWRITE
DCOM:   EQU     2022H           ;;RD, WRT, OR VERIFY
CMDBF:  EQU     2031H           ;;DOS CMD BUFFER VECTOR
CONFG:  EQU     2034H           ;;Q1,Q2,Q3,Q4,F1,F2,F3,F4 -- Q=QUAD, F=FAST STEP
;
;----- MISC ASSEMBLY CONSTANTS
;
AMAX:   EQU     7               ;;# OF AUX I/O PORTS
FMAX:   EQU     8               ;;# OF OPEN FILES
BLKSZ:  EQU     256             ;;DISK BLOCK SIZE
DDEN:   EQU     2               ;;DRIVE DENSITY -- 1=SINGLE, 2=DOUBLE
DSKSZ:  EQU     350             ;;SECTORS PER SIDE OF DISKETTE
BPAD:   EQU     0               ;;BUFFER PAD BYTE
FCBONLY: EQU    0               ;;OPEN MODE = ALLOC FCB ONLY
RD:     EQU     1               ;;OPEN MODE = READ
WRT:    EQU     2               ;;OPEN MODE = WRITE
RDWRT:  EQU     3               ;;OPEN MODE = READ/WRITE
DCOMR:  EQU     1               ;;DCOM READ COMMAND
DCOMW:  EQU     0               ;;DCOM WRITE COMMAND
;
;----- FILE DESCRIPTOR BLOCKS (FDB's)
;
FDBSZ:  EQU     4               ;;SIZE OF FDB ENTRY
FMODE:  EQU     0               ;;OFFSET TO OPEN MODE
FDEV:   EQU     1               ;;OFFSET TO N* DEVICE
FFCB:   EQU     2               ;;OFFSET TO ADDR OF CORR FCB
;----- FD 0
FDB:
        DB      RD              ;;STDIN OPEN FOR RD
        DB      0               ;;STDIN = DEV 0 -- CONSOLE
        DW      0               ;;STDIN HAS NO FCB
;----- FD 1
        DB      WRT             ;;STDOUT OPEN FOR WRT
        DB      0               ;;STDOUT = DEV 0 -- CONSOLE
        DW      0               ;;STDOUT HAS NO FCB
;----- FD 2
        DB      RDWRT           ;;STDERR OPEN FOR RD/WRT
        DB      0               ;;STDERR = DEV 0 -- CONSOLE (ALWAYS)
        DW      0               ;;STDERR HAS NO FCB
;----- FD 3 THRU 2+AMAX
        REPT    AMAX
        DB      RDWRT           ;;AUX PORTS OPEN FOR RD/WRT
        DB      1+#             ;;DEV 1,2,3....
        DW      0               ;;NO DCB's
        REPND
;----- FD 3+AMAX THRU 2+AMAX+FMAX
        REPT    FMAX
        DB      0               ;;DISK FILES CLOSED
        DB      0               ;;UNUSED
        DW      0               ;;ADDR OF CORR FCB
        REPND
;
FDBV:   DW      0               ;;CURRENT FDB VECTOR
;
;
;----- FILE CONTROL BLOCKS (FCB's)
;
;----- N* FILE-DIRECTORY ENTRY (FDE) OFFSETS
;
NAM:    EQU     0               ;;FILE NAME
DA:     EQU     8               ;;DISK ADDRESS (SECTORS 0 - 349/699)
FSZ:    EQU     10              ;;FILE SIZE IN SECTORS
FTY:    EQU     12              ;;FILE TYPE (HIGH BIT SET IF DOUBLE DENSITY FILE)
TDI:    EQU     13              ;;TYPE-DEPENDENT INFO
;
;----- OTHER OFFSETS
;
DRV:    EQU     16              ;;DRIVE
NBB:    EQU     17              ;;NXT BYTE IN BUFFER (0,1,2,...,BSZ-1)
CBF:    EQU     19              ;;CUR BUFFER IN FILE (0,1,2,...,32/67)
BSZ:    EQU     21              ;;BUFFER SIZE (256 OR 512 BYTES)
CHG:    EQU     23              ;;0=NO CHANGE TO BUFFER, 1=BUFFER CHANGED
BUF:    EQU     24              ;;BUFFER
;
; FCB'S ARE ALLOCATED BY ALLOC
;
FCBSZ:  EQU     BUF+BLKSZ*DDEN  ;;SIZE OF AN FCB
;
FCBV:   DW      0               ;;CURRENT FCB VECTOR
;
;----- CALL PARAMETERS
;
CPTDI:  DS      2               ;;OPEN - TYPE DEP. INFO
CPTYPE: DS      2               ;;OPEN - FILE TYPE
CPBASE: DS      0               ;;SEEK - BASE
CPSIZE: DS      2               ;;OPEN - FILE SIZE (256 BYTE BLOCKS); FREE - MEM SIZE
CPMODE: DS      0               ;;OPEN - MODE
CPRECSZ: DS     0               ;;READ/WRITE - RECORD SIZE
CPDISP: DS      0               ;;SEEK - DISPLACEMENT (32 BITS)
CPADDR: DS      2               ;;FREE - MEM ADDR
CPNAME: DS      0               ;;OPEN - FILE NAME ADDR
CPREC:  DS      2               ;;READ/WRITE - RECORD ADDR
CPFD:   DS      2               ;;FILE DESCRIPTOR
;
;
;
;----- MISC STORAGE
;
UPCSW:  DB      1               ;;IGNORE CTL-C'S IF ZERO
EOP:    DW      0               ;;END OF PROGRAM
FREEMEM: DW     0               ;;CURRENT FREE MEM PTR
SOB:    DW      0               ;;START OF CMD LINE BUFFER IN DOS
ARGC:   DW      0               ;;ARGUMENT COUNT FOR MAIN
RFD:    DW      2+AMAX+FMAX     ;;NEXT REDIR FD
BOJRET: DW      0               ;;ADDR FOR RET TO PROG
FDBVTMP: DW     0               ;;HOLD FDBV
CINV:   DW      0               ;;CIN VECTOR (ZERO IF LOADED WITH PROGRAM)
FILNAME: DS     8               ;;TEMP FILE NAME STORAGE
SPACE:  DB      20H
NCBF:   DW      0
NNBB:   DW      0
SAVSP:  DW      0               ;;NON-ZERO ON PROGRAM HALT ONLY (CTL-C)
TEMP:   DW      0
CMD:    DB      0
DATA:   DB      0               ;;HOLD DATA BYTE
;
;
;----- MACRO DEFINITIONS
;
        NOEXP
RARD:   MACRO   HI,LO
        MOV     A,HI
        RAR
        MOV     HI,A
        MOV     A,LO
        RAR
        MOV     LO,A
        MACND
SBCM:   MACRO   HI,LO
        MOV     A,L
        SBB     LO
        MOV     L,A
        MOV     A,H
        SBB     HI
        MOV     H,A
        MACND
CADM:   MACRO   HI,LO
        MOV     A,L
        ADC     LO
        MOV     L,A
        MOV     A,H
        ADC     HI
        MOV     H,A
        MACND
ADRHL:  MACRO   BLOCK,FIELD
        LHLD    BLOCKV
        LXI     B,FIELD
        DAD     B
        MACND
GETA:   MACRO   BLOCK,FIELD
        ADRHL   BLOCK,FIELD
        MOV     A,M
        MACND
PUTA:   MACRO   BLOCK,FIELD
        ADRHL   BLOCK,FIELD
        MOV     M,A
        MACND
GETHL:  MACRO   BLOCK,FIELD
        ADRHL   BLOCK,FIELD
        CALL    GINT
        MACND
PUTHL:  MACRO   BLOCK,FIELD
        XCHG
        ADRHL   BLOCK,FIELD
        XCHG
        CALL    PINT
        MACND
;
;
;----- CCBOJ()  stack = return addr,  DE = free mem addr
;           set up return to EOJ
;           scan for command line parameters
;           establish I/O redirection
;
CCBOJ:  ENTRY
        LHLD    CINV            ;;LOADED WITH PROG?
        MOV     A,H
        ORA     L
        JNZ     BOJO            ;;NO
        LHLD    SAVSP           ;;YES, THEN IS PROG HALTED?
        MOV     A,H
        ORA     L
        JNZ     REENTER         ;;YES, THEN CONTINUE IT
```

```
233 BOJO:   XTHL
234         SHLD    BOJRET      ;;BOJRET=RETURN ADDR FOR PROGRAM START
235         LHLD    CCEOM
236         SPHL                ;;SET STACK PTR
237         XCHG                ;;HL -> END OF PROGRAM
238         SHLD    EOP
239         SHLD    FREEMEM     ;;INIT FREE MEM PTR
240         XRA     A
241         STA     FDB+FDEV    ;;UNDIRECT STDIN
242         STA     FDB+FDBSZ+FDEV;;UNDIRECT STDOUT
243         LXI     H,2+AMAX+FMAX
244         SHLD    RFD         ;;RESET 1ST REDIR FD
245 ;
246         MVI     A,AMAX+3    ;;1ST FILE FD
247 BOJ1:   PUSH    PSW
248         CALL    LOCFDB      ;;HL=FDB VECTOR (CAN'T FAIL)
249         SHLD    FDBV
250         XRA     A
251         PUTA    FDB,FMODE   ;;MARK FD CLOSED
252         MOV     H,A
253         MOV     L,A
254         PUTHL   FDB,FFCB    ;;NO FCB ALLOCATED
255         POP     PSW
256         INR     A
257         CPI     AMAX+FMAX+3 ;;LAST FILE FD + 1
258         JNZ     BOJ1
259 ;


260         LXI     H,0
261         SHLD    SAVSP       ;;MAKE CONT CMD (^C) INVALID
262         SHLD    ARGC        ;;RESET ARGC
263 ;
264 ; PROCESS CMD LINE PARAMETERS
265 ;
266         LXI     H,CMDBF
267         CALL    GINT
268         SHLD    SOB         ;;-> DOS CMD LINE
269 ENDLP:  INX     H
270         MOV     A,M
271         CPI     0DH         ;;END OF LINE?
272         JNZ     ENDLP       ;;NO
273 ;
274 SOBCK1: CALL    GTSOB       ;;RET IF SOB<PTR
275         DCX     H
276         MOV     A,M
277         CPI     20H         ;;SPACE?
278         JZ      SOBCK1      ;;YES
279 ;
280         XRA     A
281         INX     H
282         MOV     M,A         ;;ZERO FOLLOWING BYTE
283         DCX     H
284 ;
285 DECR2:  DCX     H
286         MOV     A,M
287         CPI     20H         ;;SPACE?
288         JNZ     CKRIN
289         INX     H
290         PUSH    H           ;;NEXT ARGV PTR


291         DCX     H
292         CALL    ARGS
293         JMP     SOBCK1
294 ;
295 CKRIN:  CPI     3CH         ;; "<" ?
296         JZ      RIN         ;;YES
297         CPI     3EH         ;; ">" ?
298         JZ      ROUT        ;;YES
299         CALL    GTSOB       ;;RET IF SOB<PTR
300         JMP     DECR2
301 ;
302 ; PUSH PTR+1 & BUMP ARGC
303 ;
304 ARGS:   XCHG
305         LHLD    ARGC
306         INX     H
307         SHLD    ARGC
308         XCHG
309         RET
310 ;
311 ; REDIR STDIN


312 ;
313 RIN:    MVI     C,RD
314         JMP     RDIR
315 ;
316 ; REDIR STDOUT
317 ;
318 ROUT:   MVI     C,WRT
319 ;
320 ; REDIRECT ROUTINE
321 ;
322 RDIR:   PUSH    H           ;;SAVE CMD LINE PTR
323         MOV     A,C         ;;FETCH RD/WRT FLAG
324         SUI     1           ;;CONVERT TO STDIN/STDOUT FD
325         CALL    LOCFDB      ;;HL=FDB VECTOR (CAN'T FAIL)
326         SHLD    FDBVTMP     ;;<---
327 ;


328         POP     H           ;;FETCH CMD LINE PTR
329         PUSH    H           ;;AND SAVE IT AGAIN
330         INX     H
331         INX     H
332         MOV     A,M         ;;LOAD 2ND BYTE
333         DCX     H           ;;HL -> 1ST BYTE
334         ANA     A
335         JNZ     RFILE       ;;NOT A SINGLE CHAR TOKEN
336 ;
337         MOV     A,M
338         CPI     33H         ;;LT 1ST AUX PORT?
339         JM      RFILE       ;;YES
340         CPI     50+AMAX     ;;GT LAST AUX PORT?
341         JP      RFILE       ;;YES
342         SUI     30H         ;;CONVERT TO BINARY
343 ;
344 RFLAG:  LHLD    FDBVTMP
345         SHLD    FDBV
346         PUTA    FDB,FDEV    ;;FLAG REDIRECTION <---
347         POP     H           ;;RESTORE CMD LINE PTR
348         JMP     SOBCK1


349 ;
350 ; FILE REDIRECTION
351 ;
352 RFILE:  XCHG                ;;DE -> NAME
353         LHLD    RFD         ;;HL=NEXT REDIR FD
354         PUSH    H
355         XCHG                ;;HL -> NAME
356         PUSH    H
357         MVI     B,0         ;;BC=OPEN MODE
358         PUSH    B
359         LXI     H,0         ;;HL=SIZE, TYPE, TDI
360         PUSH    H
361         PUSH    H
362         PUSH    H
363         CALL    CCOPEN      ;;OPEN REDIR FILE <---
364         POP     B
365         POP     B
366         POP     B
367         POP     B
368         POP     B
369         POP     B
370         MOV     A,L         ;;SUCCESSFUL?
371         ANA     A
372         JZ      RFILE2      ;;YES


373 ;
374         MVI     B,52H       ;;"R" CLUE
375         JMP     ABORT
376 ;
377 RFILE2: LHLD    RFD
378         MOV     A,L         ;;A=REDIR FD
379         DCX     H           ;;DECR FOR NEXT REDIR
380         SHLD    RFD
381         JMP     RFLAG
382 ;
383 GTSOB:  XCHG
384         LHLD    SOB
385         XCHG                ;;DE=SOB, HL=PTR
386         PUSH    H
387         CALL    UCMP
388         POP     H
389         RC                  ;;RET IF SOB<PTR
390         POP     B           ;;WAST RET ADDR
391 ;
392         PUSH    H
393         POP     D
394         LXI     H,2
395         DAD     D
396         MOV     A,M
397         ORA     A           ;;ZERO?
398         XCHG
399         JZ      SKPA        ;;YES
400         INX     H
401         PUSH    H           ;;NEXT ARGV PTR
402         DCX     H
403         CALL    ARGS
404 SKPA:   LHLD    ARGC
405         PUSH    H           ;;PUSH ARGC
406         LXI     H,2


407         DAD     SP
408         PUSH    H           ;;PUSH ARGV
409 ;
410         LXI     H,EXITO
411         PUSH    H           ;;SET UP PROG EXIT TO EXITO
412         LHLD    BOJRET
413         PCHL                ;;RETURN TO PROGRAM
414 ;
415 ;
416 ;----- CCEXIT(errcode)
417 ;           close files, reset controls, jmp to DOS
418 ;errcode: 0=normal exit
419 ;         1-255 identifies specific error
420 ;               non-zero byte is displayed on console


421 ;
422 CCEXIT: ENTRY
423         LXI     H,2         ;;ERROR CODE?
424         DAD     SP
425         MOV     A,M
426         ANA     A
427         JZ      EXITO       ;;NO
428 ;
429         MOV     B,A
430 ABORT:  XRA     A
431         CALL    COUT
432         MVI     B,3FH       ;;"?"
433         XRA     A
434         CALL    COUT
435         MVI     B,0DH       ;;CR
436         XRA     A
437         CALL    COUT
438         MVI     B,0AH       ;;LF
439         XRA     A
440         CALL    COUT
441 ;
442 EXITO:  LXI     H,AMAX+3    ;;1ST FILE FD
443 EXIT1:  PUSH    H
444         CALL    CCCLOSE     ;;CLOSE IT IF OPEN
445         LXI     H,0
446         PUTHL   FDB,FFCB    ;;ZERO FCB ADDR
447         POP     D
448         PUSH    D
449         LXI     H,AMAX+FMAX+2;;LAST FILE FD
450         CALL    UCMP        ;;DONE?
451         JZ      DOS         ;;YEP
452         POP     H
453         INX     H
454         JMP     EXIT1
455 ;
456 ;
457 ;----- NORMAL RETURN TO USER PROGRAM
458 ;
459 DONE:   LXI     H,0
460         RET
461 ;
462 ;
463 ;----- ERROR RETURNS
464 ;
```

```
ERR2:   POP     B
ERR1:   POP     B
ERR:    LXI     H,-2
        RET
;
;
;----- BLOCK MOVE
;       HL=SOURCE ADDR, DE=DEST ADDR, C=LENGTH
;
BMOVE:  MOV     A,M
        INX     H
        XCHG
        MOV     M,A
        INX     H
        XCHG
        DCR     C
        JNZ     BMOVE
        RET
;
;----- GET CALL PARMS FROM STACK
;       DE=DEST ADDR, C=# BYTES TO FETCH
;
GETP:   LXI     H,4             ;;TWO CALLS DEEP
        DAD     SP
GETP1:  MOV     A,M
        STAX    D
        DCR     C
        RZ
        INX     H
        INX     D
        JMP     GETP1
;
;----- FIND FDB & FCB ENTRIES
;



FINDF:  LDA     CPFD            ;;LOAD FD
        CALL    LOCFDB          ;;HL=ORIGINAL FDB VECTOR
        LDA     CPFD
        CPI     2               ;;REDIRECTABLE?
        JP      FNDF1           ;;NO
        XCHG                    ;;DE=FDB VECTOR
        LXI     H,FDEV
        DAD     D               ;;HL=FDB DEV VECTOR
        MOV     A,M             ;;LOAD DEV
        XCHG                    ;;HL=FDB VECTOR
        ANA     A               ;;REDIRECTED?
        JZ      FNDF1           ;;NO
        CALL    LOCFDB          ;;HL=REDIR FDB VECTOR
FNDF1:  SHLD    FDBV            ;;SAVE FDB VECTOR <---
        LXI     D,FFCB          ;;OFFSET TO FCB VECTOR
        DAD     D               ;;HL=ADDR OF FCB VECTOR
        CALL    GINT            ;;HL=FCB VECTOR
        SHLD    FCBV            ;;SAVE FCB VECTOR <---
        RET
;
;----- LOCATE FDB ENTRY (A=FD)
;
LOCFDB: CPI     3+AMAX+FMAX     ;;OUT OF RANGE?
        JP      ERR2            ;;YES



        ANA     A               ;;CY=0
        JM      ERR2            ;;YES
        RAL
        RAL                     ;;A = FD*4
        MOV     E,A
        MVI     D,0
        LXI     H,FDB
        DAD     D
        RET
;
;----- FILL BUFFER, BUMP CBF, ZERO NBB & CHG
;
FILL:   CALL    UBUF
        CALL    ZNBB
        MVI     A,DCOMR
        STA     CMD
        CALL    DSKIO           ;; <---
        RET
;
;----- RETURN IF CHANGE FLAG ZERO
;----- FLUSH BUFFER, BUMP CBF, ZERO NBB & CHG
;----- PAD NEW BUFFER
;
BFLUSH: GETA    FCB,CHG         ;;CHG ZERO?
        ANA     A
        RZ                      ;;YES
;
        MVI     A,DCOMW
        STA     CMD
        CALL    DSKIO           ;;FLUSH IT <---
        CALL    UBUF            ;;ADVANCE TO NEXT BUFFER
;
IBUF:
;
; IS THIS A ZERO LENGTH FILE?
;
        GETHL   FCB,FSZ         ;;HL=FSZ
        MOV     A,H
        ORA     L
        JNZ     IBUF1           ;;NO
;
; YES -- WILL NEW BUFFER FIT ON THE DISK?
;
        GETHL   FCB,CBF
        XCHG
        GETHL   FCB,DA          ;;DISK ADDR OF FILE
        DAD     D
        CALL    CDSZ            ;;(CBF+DA)<DRIVE SIZE?
        JC      IBUF2           ;;YES
        RET                     ;;NO, RETURN WITH CY=0
;
; WILL NEW BUFFER FIT IN ALLOCATED FILE SPACE?
;
IBUF1:  XCHG                    ;;DE=FSZ
        GETHL   FCB,CBF         ;;HL=CBF
        XCHG
        CALL    UCMP            ;;CBF<FSZ?
        RNC                     ;;NO, RETURN WITH CY=0
;
IBUF2:  GETHL   FCB,BSZ
        XCHG                    ;;DE=BUFFER SIZE
        ADRHL   FCB,BUF         ;;HL -> BUFFER
IBUF3:  MVI     A,BPAD          ;;A=PAD BYTE
        MOV     M,A
        INX     H
        DCX     D
        MOV     A,D
        ORA     E
        JNZ     IBUF3
;
;----- ZERO NBB
;
ZNBB:   LXI     H,0
        PUTHL   FCB,NBB         ;;ZERO NBB
        STC                     ;;FLAG NEW BUFFER FITS
        RET
;
;----- UPDATE BUFFER CONTROLS
;
UBUF:   XRA     A
        PUTA    FCB,CHG         ;;CLEAR CHANGE FLAG
        GETHL   FCB,CBF         ;;BUMP CBF
        INX     H
        PUTHL   FCB,CBF
        RET
;
;----- SET UP DCOM CALL FOR READ OR WRITE
;
DSKIO:  GETHL   FCB,DA
        XCHG
        GETHL   FCB,CBF
        DAD     D
        PUSH    H               ;;SAVE DA+CBF
        ADRHL   FCB,BUF
        PUSH    H               ;;SAVE BUFFER ADDR
        GETA    FCB,FTY         ;;BIT 7 => DD
        ANI     80H
        MOV     E,A
        GETA    FCB,DRV
        ORA     E
        MOV     C,A             ;;C=DRIVE (BIT 7 => DD)
        LDA     CMD
        MOV     B,A             ;;B=DCOM COMMAND
        MVI     A,1             ;;A=NBR OF SECTORS
        POP     D               ;;DE=BUFFER ADDR
        POP     H               ;;HL=DA+CBF=DSK ADDR
        CALL    DCOM            ;;DOS CALL <---
        JC      ERR2
        RET
;
;----- COMPARE HL TO DRIVE SIZE IN SECTORS
;       SET CY IF HL<DRIVE SIZE
;
CDSZ:   XCHG                    ;;DE=(HL)
        GETA    FCB,DRV
        MOV     C,A             ;;C=DRIVE



        LDA     CONFG           ;;A=DOS CONFIGURATION BYTE
CDSZ6:  RAL                     ;;SELECT BIT FOR THIS DRIVE
        DCR     C               ;;AND SET CY IF DOUBLE SIDED
        JNZ     CDSZ6
        LXI     H,DSKSZ         ;;HL=SECTORS PER SIDE
        JNC     CDSZ7           ;;SINGLE SIDED
        MOV     A,L             ;;DOUBLE HL GIVING QUAD DRIVE SIZE
        ANA     A
        RAL
        MOV     L,A
        MOV     A,H
        RAL
        MOV     H,A
CDSZ7:  CALL    UCMP            ;;<HL><DRIVE SIZE?
        RET
;
;----- SET NEW CBF & NBB VALUES AND ZERO CHG
;
NBUF:   LHLD    NCBF
        PUTHL   FCB,CBF         ;;NEW CBF
        LHLD    NNBB
        PUTHL   FCB,NBB         ;;NEW NBB
        XRA     A
        PUTA    FCB,CHG         ;;ZERO CHG
        RET
;
; TRAP CALL TO CIN
;
CINTRAP:SHLD    ARGC
        CPI     0               ;;CONSOLE INPUT?
        JNZ     CINTRAP2        ;;NO - LET IT GO THEN
CINTRAP1:LXI    H,CINTRAP3      ;;YES - SET UP RET HERE
        PUSH    H
CINTRAP2:LHLD   CINV
        PUSH    H
        LHLD    ARGC            ;;RESTORE HL
        RET                     ;;CALL CIN ***
;
CINTRAP3:CPI    12H             ;;CTL-R?
        JZ      UPR             ;;YES
;
CINTRAP4:LHLD   ARGC            ;;RESTORE HL
        RET                     ;;RET BYTE TO CALLER
;
UPR:    LHLD    SAVSP
        MOV     A,H
        ORA     L
        MVI     A,12H
        JZ      CINTRAP4        ;;REENTER CMD INVALID - RET CTL-R TO CALLER
        JMP     REENTER
;
;
;----- CCPOLL(flag)
;               flag=0 -- inhibit interrupt checking
;               flag>0 -- enable interrupt checking & take a look
;
CCPOLL: ENTRY
        LXI     H,2             ;;FETCH LOW BYTE OF FLAG
```

```
697             DAD     SP
698             MOV     A,M
699             STA     UPCSW
700             ANA     A
701             RZ
702             CALL    CONTC
703             RNC                     ;;NO CONSOLE INPUT
704             JZ      CCHALT          ;;CTL-C FROM CONSOLE
705 ;
706             MVI     A,0             ;;PAUSE TILL NEXT CHR FROM CONSOLE
707             CALL    CIN
708             CPI     3               ;;CTL-C?
709             RNZ                     ;;NO, RESUME PROGRAM
710 ;
711 ;
712 ;----- CCHALT()
713 ;           save context and return to DOS
714 ;
715 CCHALT: ENTRY
716             PUSH    PSW             ;;YES, SAVE CONTEXT
717             PUSH    B
718             PUSH    D
719             PUSH    H
720             LXI     H,0
721             DAD     SP
722             SHLD    SAVSP           ;;SAVE SP
723             JMP     DOS
724 ;
725 REENTER: LHLD   SAVSP           ;;RESTORE SP
726             SPHL
727             LXI     H,0
728             SHLD    SAVSP           ;;FLAG CONT CMD INVALID
729             POP     H
730             POP     D
731             POP     B
732             POP     PSW
733             RET                     ;;CONTINUE
734 ;
735 ;
736 ;----- CCOPEN(fd, name, mode, size, type, tdi)
737 ;
738 ;   fd: file descriptor
739 ; name: address of N* file name ( and drive) terminaded with NULL char
740 ; mode: 1=rd, 2=wrt, 3=rd/wrt, 0=allocate FCB and exit
741 ; size: file size in blocks for open, 0=use space in file or space avail
742 ; type: file type for new file (0-127)
743 ;  tdi: type-dependent-info for new file (0-65535)
744 ;
745 CCOPEN: ENTRY
746             LXI     D,CPTDI         ;;FETCH CALL PARMS
747             MVI     C,12
748             CALL    GETP
749             CALL    FINDF           ;;SET FDBV & FCBV
750             LXI     D,0
751             LDA     CPFD            ;;I/O PORT?
752             CPI     3+AMAX
753             JM      ERR             ;;YES
754             GETA    FDB,FMODE       ;;OPEN?
755             ANA     A
756             JNZ     ERR             ;;YES
757 ;
758             LHLD    FCBV            ;;FCB ALREADY ALLOCATED?
759             MOV     A,H
760             ORA     L
761             JNZ     OPNO            ;;YES
762             LXI     H,FCBSZ         ;;GET MEM FOR NEW FCB
763             PUSH    H
764             CALL    CCALLOC
765             POP     B
766             SHLD    FCBV
767             PUTHL   FDB,FFCB
768 ;
769 OPNO:   LDA     CPMODE
770             CPI     FCBONLY
771             JZ      DONE            ;;ALLOC FCB ONLY
772 ;
773             MVI     A,1             ;;SET DEFAULT DRIVE NUMBER
774             PUTA    FCB,DRV
775 ;
776 ; VALIDATE FILE NAME
777 ;
778             LHLD    FCBV
779             XCHG                    ;;DE -> NAME IN FCB
780             LHLD    CPNAME          ;;HL -> NAME
781             MVI     C,8             ;;CHARS TO SEARCH
782 OPN1:   MOV     A,M             ;;LOAD NXT BYTE
783             INX     H               ;;NXT BYTE
784             ANA     A               ;;NULL?
785             JZ      OPN3            ;;YES
786             CPI     2CH             ;;COMMA?
787             JZ      OPN2            ;;YES
788             CPI     20H             ;;SPACE?
789             JZ      OPNERR          ;;YES
790             XCHG
791             MOV     M,A             ;;STORE IN FCB NAME--MAY HARMLESSLY OVERWRITE DA BYTE
792             INX     H
793             XCHG
794             DCR     C
795             JP      OPN1            ;;NOT TRAILING BYTE YET
796             JMP     OPNERR          ;;NINTH BYTE MUST BE NULL OR COMMA
797 ;
798 OPN2:   MOV     A,M             ;;DRIVE = 1 THRU 8?
799             CPI     31H             ;;1
800             JM      OPNERR          ;;NO
801             CPI     39H             ;;9
802             JP      OPNERR          ;;NO
803             SUI     30H             ;;CONVERT TO BINARY
804             PUSH    B
805             PUSH    H
806             PUTA    FCB,DRV         ;;STORE IN FCB OVER DEFAULT
807             POP     H
808             POP     B
809             INX     H
810             MOV     A,M             ;;NEXT BYTE NULL?
811             ANA     A
812             JNZ     OPNERR          ;;NO
813 ;
814 ; CALL DLOOK TO FIND FILE
815 ;
816 OPN3:   XCHG
817             MVI     A,20H           ;;FORCE TRAILING SPACES
818 OPN4:   MOV     M,A             ;;WILL OVERRITE DA BYTE
819             INX     H
820             DCR     C
821             JP      OPN4
822             GETA    FCB,DRV         ;;A=DRV
823             LHLD    FCBV            ;;HL -> FILENAME,SP,SP,...
824             CALL    DLOOK           ;;LOOK FOR IT <---
825             JC      NEW             ;;NOT FOUND
826 ;
827 ; COPY N* FDE INTO FCB
828 ;
829 OLD:    LXI     D,-8            ;;BACK UP 8 BYTES
830             DAD     D
831             XCHG
832             LHLD    FCBV
833             XCHG                    ;;DE -> DEST
834             MVI     C,16            ;;FDE SIZE
835             CALL    BMOVE
836 ;
837             GETA    FCB,FTY         ;;MINUS IF DD
838             ANI     80H             ;;ISOLATE DENSITY BIT
839             RLC
840             ADI     1               ;;1=SD, 2=DD
841             MOV     H,A             ;;HL = 0100H/0200H (256/512)
842             MVI     L,0
843             PUTHL   FCB,BSZ         ;;SET BUFFER SIZE <---
844 ;
845             LDA     CPMODE          ;;LOW BYTE OF MODE
846             CPI     RD              ;;RD?
847             JZ      OLD2            ;;YES
848             CPI     RDWRT           ;;RD/WRT?
849             JZ      OLD2            ;;YES
850             CPI     WRT             ;;WRT?
851             JNZ     OPNERR          ;;NO
852 ;
853 ; YES, THEN FORCE ZERO LENGTH IF LAST FILE ON DISKETTE
854 ;
855             LXI     H,20FFH         ;;NEVER, NEVER FILE NAME
856             SHLD    FILNAME
857             GETA    FCB,DRV
858             LXI     H,FILNAME
859             CALL    DLOOK           ;; <---
860             JNC     OLD2            ;;SHOULD HAVE FAILED BUT DIDN'T
861 ;
862             PUSH    H               ;;SAVE ADDR OF 1ST FREE SECTOR
863             GETHL   FCB,DA
864             PUSH    H               ;;SAVE ADDR OF CURRENT FILE
865             GETHL   FCB,FSZ
866             XCHG                    ;;DE = CUR FILE SZ
867             POP     H               ;;HL = CUR FILE ADDR
868             DAD     D               ;;HL = NEXT SECTOR ADDR
869             POP     D               ;;DE = 1ST FREE SECTOR ADDR
870             CALL    UCMP            ;;DE=HL?
871             JNZ     OLD2            ;;NO, MUST BE IMBEDDED FILE
872 ;
873             LHLD    CPSIZE          ;;YES, MUST BE LAST FILE
874             MOV     A,H
875             ORA     L
876             JNZ     OPNB            ;;NON-ZERO SIZE ARG MEANS LEAVE SAME SIZE
877             PUTHL   FCB,FSZ         ;;ELSE ZERO FSZ <---
878             JMP     OPNB
879 ;
880 ; MODE = RD, RD/WRT, OR (WRT & IMBEDDED FILE)
881 ;
882 OLD2:   GETHL   FCB,FSZ
883             MOV     A,H
884             ORA     L
885             JZ      OPNERR          ;;FAIL IF ZERO LENGTH FILE
886             JMP     OPNB
887 ;
888 NEW:    PUTHL   FCB,DA          ;;FIRST FREE DSK ADDR <---
889             MVI     A,DDEN
890             DCR     A
891             RRC
892             MOV     B,A
893             LDA     CPTYPE
894             ANI     7FH
895             ORA     B               ;;SIGN=>DD
896             PUTA    FCB,FTY         ;;<---
897             LHLD    CPSIZE
898             ANA     A               ;;CY=0
899             JP      NEW1            ;;SD
900 ;
901             INX     H               ;;FORCE FILE SIZE UP TO NXT EVEN NBR
902             MOV     A,H             ;;DIV BY 2 TO GET SECTORS
903             RAR
904             MOV     H,A
905             MOV     A,L
906             RAR
907             MOV     L,A
908 ;
909 NEW1:   PUTHL   FCB,FSZ         ;;FILE SIZE (SECTORS) <---
910             LXI     H,DDEN*BLKSZ
911             PUTHL   FCB,BSZ         ;;SET BUFFER SZ <---
912             LHLD    CPTDI
913             PUTHL   FCB,TDI         ;;TDI BYTES 1&2
914             XRA     A
915             PUTA    FCB,TDI+2       ;;TDI BYTE 3
916 ;
917             LDA     CPMODE
918             CPI     WRT             ;;WRT?
919             JNZ     OPNERR          ;;NO
920 ;
921 NEW2:   GETHL   FCB,DA
922             XCHG
923             GETHL   FCB,FSZ
924             DAD     D               ;;HL=DA+FSZ
925             LXI     D,DSKSZ         ;;DE=DSKSZ
926             CALL    UCMP
927             JC      OPNERR          ;;DA+FSZ > DSKSZ -- NO ROOM!!!
928 ;
929 ; DISALLOW IF PREVIOUS OPEN WITH FSZ=0 ON SAME DRIVE
930 ;
931             MVI     B,FDBSZ
932             MVI     C,FMAX
933             LXI     H,FDB+FDBSZ*(3+AMAX)    ;;FIRST FILE-FDB
934 ;
935 NEW3:   MOV     A,M             ;;LOAD OPEN-MODE
```

```
936             ANA     A               ;;OPEN?
937             JZ      NEW4            ;;NO -- SKIPS CUR FDB TOO
938             PUSH    H               ;;SAVE CUR FDB ADDR
939             LXI     D,FFCB
940             DAD     D               ;;HL -> FCB ADDR
941             CALL    GINT            ;;HL -> FCB
942             SHLD    TEMP
943             LXI     D,FSZ
944             DAD     D               ;;HL -> FSZ
945             CALL    GINT            ;;HL=FSZ
946             LXI     D,0
947             CALL    UCMP            ;;FSZ=0?
948             POP     H               ;;RESTORE CUR FDB ADDR
949             JNZ     NEW4            ;;FSZ<>0
950             PUSH    H
951             LHLD    TEMP            ;;HL -> FCB
952             LXI     D,DRV
953             DAD     D
954             CALL    GINT
955             XCHG                    ;;DE=DRV (GARBAGE IN HIGH BYTE)
956             PUSH    B
957             GETA    FCB,DRV         ;;A=DRV THIS CALL
958             POP     B
959             POP     H
960             XRA     E               ;;SAME DRIVE?
961             JZ      OPNERR          ;;YEP! -- SAME DRV, FSZ=0, OPEN
962 ;
963 NEW4:       MVI     D,0
964             MOV     E,B
965             DAD     D               ;;BUMP TO NXT FDB
966             DCR     C               ;;ANOTHER?
967             JNZ     NEW3            ;;YES
968 ;
969 ; FIND NULL FDE
970 ;
971             GETA    FCB,DRV
972             LXI     H,SPACE
973             CALL    DLOOK           ;;EMPTY FDE?
974             JC      OPNERR          ;;NO
975 ;
976 ; COPY FCB FIELDS TO FDE
977 ;
978             LXI     D,-8            ;;BACKUP 8 BYTES
979             DAD     D
980             XCHG                    ;;DE -> FDE
981             ADRHL   FCB,NAM         ;;FDE FIELDS IN FCB
982             MVI     C,16            ;;FDE SIZE
983             CALL    BMOVE
984 ;
985 ; WRITE FDE TO DISK DIRECTORY
986 ;
987             CALL    DWRIT           ;;NEW FDE  <---
988 ;
989 OPNB:       LXI     H,0
990             PUTHL   FCB,CBF         ;;CBF=0 <---
991             CALL    IBUF            ;;INIT NEW BUFFER, ZERO NBB, CHECK FIT
992             JNC     OPNERR          ;;WON'T FIT
993             XRA     A
994             PUTA    FCB,CHG         ;;CHG=0 <---
995             LDA     CPMODE
996             PUTA    FDB,FMODE       ;;SET MODE IN FDB
997             CPI     WRT             ;;WRT?
998             JNZ     SEEK4           ;;NO -- SEEK 1ST BYTE (RD OR RD/WRT)
999             JMP     DONE            ;;EXIT
1000 ;
1001 OPNERR:    CALL    FCBFREE         ;;DEALLOCATE FCB
1002             JMP     ERR
1003 ;
1004 ; TRY TO DEALLOCATE THE CURRENT FCB
1005 ;
1006 FCBFREE:   LHLD    FCBV
1007             PUSH    H
1008             LXI     H,FCBSZ
1009             PUSH    H
1010             CALL    CCFREE          ;;HL=0 IF MEM RELEASED, ELSE ORIG FCBV VALUE
1011             POP     B
1012             POP     B
1013             PUTHL   FDB,FFCB        ;;ZERO FCB PTR OR LEAVE UNCHANGED
1014             RET
1015 ;
1016 ;
1017 ;----- CCCLOSE(fd)
1018 ;
1019 CCCLOSE:   ENTRY
1020             LXI     D,CPFD          ;;FETCH CALL PARM
1021             MVI     C,2
1022             CALL    GETP
1023             CALL    FINDF           ;;SET FDBV & FCBV VECTORS
1024             LXI     D,0
1025             CALL    UCMP            ;;HL=FCBV=0? (I/O PORT)
1026             JZ      ERR             ;;YES
1027             GETA    FDB,FMODE       ;;OPEN?
1028             ANA     A
1029             JZ      ERR             ;;NO
1030 ;
1031             CALL    BFLUSH          ;;FLUSH THE BUFFER IF NECESSARY
1032 ;
1033             GETHL   FCB,FSZ         ;;FILE SIZE ZERO?
1034             LXI     D,0
1035             CALL    UCMP
1036             JNZ     CLSE2           ;;NO
1037 ;
1038             GETHL   FCB,CBF         ;;HIGHEST BUFFER + 1
1039             PUTHL   FCB,FSZ         ;;SET FILE SIZE <---
1040             LHLD    FCBV            ;;COPY FILE NAME FROM FCB TO FILNAME
1041             LXI     D,FILNAME       ;;SINCE FILNAME IS FOLLOWED BY A SPACE
1042             MVI     C,8             ;;WHICH DLOOK NEEDS
1043             CALL    BMOVE
1044 ;
1045             GETA    FCB,DRV         ;;READ FDE FROM DISK
1046             LXI     H,FILNAME
1047             CALL    DLOOK           ;; <---
1048             JC      ERR
1049 ;
1050             INX     H
1051             INX     H
1052             XCHG                    ;;DE -> FSZ IN DOS FDE
1053             ADRHL   FCB,FSZ         ;;HL->FSZ IN FCB
1054             MVI     C,6
1055             CALL    BMOVE           ;;COPY FSZ, FTY, AND TDI INTO DOS FDE
1056             CALL    DWRIT           ;;WRITE UPDATED FDE TO DISK <---
1057 ;
1058 CLSE2:      CALL    FCBFREE         ;;TRY TO DEALLOCATE FCB
1059             XRA     A
1060             PUTA    FDB,FMODE       ;;CLOSE FILE <---
1061             JMP     DONE
1062 ;
1063 ;
1064 ;----- CCREAD(fd, rec, recsz)
1065 ;
1066 ;   fd: file descriptor of an open file
1067 ;  rec: memory address of user data
1068 ;recsz: number of bytes to transfer
1069 ;
1070 CCREAD:    ENTRY
1071             LXI     D,CPRECSZ       ;;FETCH CALL PARMS
1072             MVI     C,6
1073             CALL    GETP
1074             CALL    FINDF           ;;SET FDBV & FCBV VECTORS
1075             GETA    FDB,FMODE       ;;OPEN FOR RD OR RD/WRT?
1076             CPI     RD
1077             JZ      RD1             ;;YES
1078             CPI     RDWRT
1079             JNZ     ERR             ;;NO
1080 RD1:        LXI     H,0             ;;YES
1081             SHLD    TEMP            ;;BYTE INDEX = 0
1082 ;
1083 ; LAST BYTE TRANSFERRED?
1084 ;
1085 RD2:        XCHG                    ;;DE=CUR BYTE INDEX
1086             LHLD    CPRECSZ         ;;HL=BYTES TO TFR
1087             PUSH    D
1088             CALL    UCMP            ;;CUR BYTE<BYTES TO TFR?
1089             POP     H               ;;HL=CUR BYTE INDEX
1090             JC      RD3             ;;YES -- NOT YET FINISHED
1091             RET                     ;;FINISHED -- RETURN COUNT OF BYTES TRANSFERRED
1092 ;
1093 ; GET NEXT BYTE
1094 ;
1095 RD3:        LDA     FCBV
1096             MOV     D,A
1097             LDA     FCBV+1
1098             ORA     D
1099             JNZ     RD5             ;;FILE INPUT
1100 ;
1101 RD3A:       GETA    FDB,FDEV        ;;I/O PORT INPUT -- WHICH PORT?
1102             CALL    CIN             ;; <---
1103             MOV     B,A             ;;SAVE DATA
1104             LDA     UPCSW           ;;HONOR CTL-C INTERRUPT?
1105             ANA     A
1106             MOV     A,B             ;;RESTORE DATA
1107             JZ      RD4             ;;NO
1108             CPI     3               ;;CTL-C?
1109             JNZ     RD4             ;;NO
1110             GETA    FDB,FDEV
1111             ORA     A               ;;FROM CONSOLE?
1112             MOV     A,B             ;;RESTORE DATA
1113             JNZ     RD4             ;;NO
1114             CALL    CCHALT          ;;YES
```

While your manuscript may be too technical for some other micro publications, it may be "just what the doctor ordered" for our audience. Let us take a look at it, won't you? You can reach us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

```
1115            JMP     RD3A
1116    ;
1117    ; MOVE BYTE FROM (A) TO USER AREA
1118    ;
1119    RD4:    LHLD    TEMP
1120            XCHG                    ;;DE=BYTE INDEX
1121            LHLD    CPREC
1122            DAD     D               ;;HL=CUR BYTE IN USER AREA
1123    ;
1124            MOV     M,A             ;; <---
1125    ;
1126            LXI     H,1
1127            DAD     D               ;;BUMP INDEX
1128            SHLD    TEMP
1129            JMP     RD2
1130    ;
1131    ; ANY MORE BYTES IN BUFFER?
1132    ;
1133    RD5:    GETHL   FCB,NBB
1134            XCHG
1135            GETHL   FCB,BSZ
1136            PUSH    D
1137            CALL    UCMP            ;;NBB<BSZ?
1138            POP     D               ;;DE=NBB
1139            JNC     RD6             ;;NO
1140    ;
1141    ; FETCH NEXT BYTE FROM BUFFER
1142    ;
1143    RD5A:   ADRHL   FCB,BUF
1144            DAD     D               ;;HL-> NEXT BYTE IN BUFFER
1145            MOV     A,M             ;; <---
1146            LXI     H,1
1147            DAD     D               ;;BUMP NBB
1148            PUSH    PSW
1149            PUTHL   FCB,NBB
1150            POP     PSW
1151            JMP     RD4             ;;GO PUT BYTE IN USER AREA
1152    ;
1153    RD6:    GETHL   FCB,CBF
1154            INX     H
1155            PUSH    H               ;;SAVE NEW CBF FOR SEEK
1156            XCHG
1157            GETHL   FCB,FSZ
1158            CALL    UCMP            ;;(CBF+1)<FSZ?
1159            POP     H               ;;NEW CBF
1160            JC      RD7             ;;YES
1161            LHLD    TEMP            ;;NO
1162            RET
1163    ;
1164    ; SEEK NEXT SECTOR
1165    ;
1166    RD7:    LXI     D,0             ;;NEW NBB
1167            CALL    RMSEEK          ;; <---
1168            MOV     A,H             ;;SEEK OK?
1169            ORA     L
1170            JNZ     ERR             ;;NO
1171            XCHG                    ;;DE=0=NBB
1172            JMP     RD5A
1173    ;
1174    ;
1175    ;----- CCWRITE(fd, rec, recsz)
1176    ;
1177    ;   fd: file descriptor of an open file
1178    ;  rec: memory address of user data
1179    ;recsz: number of bytes to transfer
1180    ;
1181    CCWRITE: ENTRY
1182            LXI     D,CPRECSZ       ;;FETCH CALL PARMS
1183            MVI     C,6
1184            CALL    GETP
1185            CALL    FINDF           ;;SET FDBV & FCBV VECTORS
1186            GETA    FDB,FMODE       ;;OPEN FOR RD OR RD/WRT?
1187            CPI     WRT
1188            JZ      WRT1            ;;YES
1189            CPI     RDWRT
1190            JNZ     ERR             ;;NO
1191    ;
1192    WRT1:   LXI     H,0             ;;YES
1193            SHLD    TEMP            ;;BYTE INDEX = 0
1194    WRT2:   XCHG                    ;;DE=CUR BYTE INDEX
1195            LHLD    CPRECSZ         ;;HL=BYTES TO TRANSFER
1196            PUSH    D
1197            CALL    UCMP
1198            POP     H
1199            RNC                     ;;FINISHED -- RETURN COUNT OF BYTES TRANSFERED
1200    ;
1201    ; TRANSFER NEXT BYTE
1202    ;
1203    WRT3:   XCHG                    ;;DE=CUR BYTE INDEX
1204            LHLD    CPREC
1205            DAD     D               ;;HL -> CUR BYTE OF USER DATA
1206            MOV     E,M             ;;E=DATA BYTE <---
1207    ;
1208            LDA     FCBV
1209            MOV     D,A
1210            LDA     FCBV+1
1211            ORA     D
1212            JNZ     WRT5            ;;FILE OUTPUT
1213    ;
1214            GETA    FDB,FDEV        ;;I/O PORT OUTPUT -- WHICH PORT?
1215            MOV     B,E             ;;DATA BYTE
1216            CALL    COUT
1217    ;
1218    ; BUMP BYTE INDEX & LOOP BACK
1219    ;
1220    WRT4:   LHLD    TEMP
1221            INX     H
1222            SHLD    TEMP
1223            JMP     WRT2
1224    ;
1225    ; WILL BYTE FIT IN CUR BUFFER?
1226    ;
1227    WRT5:   MOV     A,E             ;;SAVE DATA BYTE (E)
1228            STA     DATA
1229            GETHL   FCB,NBB
1230            XCHG                    ;;DE=NBB
1231            GETHL   FCB,BSZ         ;;HL=BSZ
1232            CALL    UCMP            ;;NBB<BSZ?
1233            JC      WRT10           ;;YES
1234    ;
1235    ; NO -- THEN SEEK OR INIT NEXT BUFFER
1236    ;
1237            GETA    FDB,FMODE
1238            CPI     WRT             ;;WRT MODE?
1239            JZ      WRT9            ;;YES
1240    ;
1241            GETHL   FCB,CBF         ;;NO -- MUST BE RD/WRT MODE THEN
1242            INX     H               ;;NEW CBF
1243            LXI     D,0             ;;NEW NBB
1244            CALL    RMSEEK          ;; <---
1245            MOV     A,H             ;;DID SEEK FAIL?
1246            ORA     L
1247            JNZ     WRT9A           ;;YES -- EOF
1248            JMP     WRT10           ;;NO -- GOOD
1249    ;
1250    ; FLUSH & PAD THE BUFFER, BUMP CBF
1251    ;
1252    WRT9:   CALL    BFLUSH
1253            JC      WRT10           ;;NEXT BUFFER FITS
1254    ;
1255    WRT9A:  LHLD    TEMP            ;;NO ROOM SO RETURN COUNT OF BYTES TRANSFERRED
1256            RET
1257    ;
1258    ; STUFF THIS BYTE IN THE BUFFER
1259    ;
1260    WRT10:  MVI     A,1
1261            PUTA    FCB,CHG         ;;SET CHANGE-FLAG
1262            GETHL   FCB,NBB
1263            XCHG                    ;;DE=NBB OFFSET IN BUFFER
1264            ADRHL   FCB,BUF
1265            DAD     D               ;;HL -> NEXT BYTE IN THE BUFFER
1266            LDA     DATA
1267            MOV     M,A             ;;STUFF IN BUFFER
1268    ;
1269            XCHG
1270            INX     H               ;;BUMP NBB
1271            PUTHL   FCB,NBB
1272            JMP     WRT4
1273    ;
1274    ;
1275    ;----- CCSEEK(fd, disp, base)
1276    ;
1277    ;    fd: file descriptor
1278    ;  disp: displacement from base (32 bit value)
1279    ;  base: 0 = bytes from current position
1280    ;        1,2,3,... = 1,2,3,... byte units from BOF
1281    ;
1282    CCSEEK:  ENTRY
1283            LXI     D,CPBASE        ;;FETCH CALL PARMS
1284            MVI     C,8
1285            CALL    GETP
1286            CALL    FINDF           ;;SET FDBV & FCBV VECTORS
1287    ;
1288            LDA     CPFD
1289            CPI     3+AMAX          ;;I/O PORT?
1290            JM      ERR             ;;YES
1291    ;
1292            GETA    FDB,FMODE       ;;OPEN?
1293            ANA     A
1294            JZ      ERR             ;;NO
1295            CPI     WRT             ;;OPEN FOR WRT?
1296            JZ      ERR             ;;YES
1297    ;
1298    ; CALCULATE NEW CBF & NBB
1299    ;
1300            LHLD    CPBASE
1301            MOV     A,H
1302            ORA     L
1303            JNZ     SEEK1C
1304            GETHL   FCB,BSZ
1305            PUSH    H
```

Our high-quality editorial and your high-quality products are made for each other! Call (415) 323-3111 to reserve your space in the next *Dr. Dobb's Journal.*

```
1306            GETHL   FCB,CBF
1307            LXI     D,0             ;;DE,HL=CBF
1308            POP     B               ;;BC=BSZ
1309            CALL    MULT32          ;;DE,HL=CBF*BSZ
1310            PUSH    D
1311            XCHG
1312            GETHL   FCB,NBB
1313            DAD     D
1314            POP     D
1315            JNC     SEEK1A
1316            INX     D
1317 SEEK1A:   PUSH    D               ;;DE,HL=(CBF*BSZ)+NBB
1318            XCHG
1319            LHLD    CPDISP
1320            DAD     D
1321            POP     D
1322            JNC     SEEK1B
1323            INX     D
1324 SEEK1B:   PUSH    H
1325            LHLD    CPDISP+2
1326            DAD     D
1327            XCHG
1328            POP     H               ;;DE,HL=(((CBF*BSZ)+NBB)+CPDISP
1329            JMP     SEEK1D
1330 SEEK1C:   LHLD    CPBASE
1331            PUSH    H
1332            LHLD    CPDISP+2
1333            XCHG
1334            LHLD    CPDISP          ;;DE,HL=CPDISP

1335            POP     B               ;;BC=CPBASE
1336            CALL    MULT32          ;;DE,HL=CPDISP*CPBASE
1337 SEEK1D:   PUSH    H
1338            GETHL   FCB,BSZ         ;;MUST BE A POWER OF 2 FROM 1 TO 16
1339            MVI     C,-1
1340 SEEK1E:   ANA     A
1341            INR     C
1342            RARD    H,L
1343            JNC     SEEK1E
1344            POP     H               ;;DE,HL=TARGET BYTE POINTER, C=LOG2(BSZ)
1345            CALL    P2DIV           ;;HL=NCBF, DE=NNBB
1346 ;
1347 ; <<< ENTRY POINT FROM READ & WRITE >>>
1348 ;
1349 RWSEEK:   SHLD    NCBF            ;;SAVE NEW CBF
1350            XCHG
1351            SHLD    NNBB            ;;SAVE NEW NBB (REMAINDER)
1352            GETHL   FCB,CBF
1353            CALL    UCMP            ;;NEW CBF = OLD CBF?
1354            JZ      SEEK5           ;;YES, SKIP FLUSH & FILL
1355 ;
1356 ; FLUSH OLD BUFFER
1357 ;
1358            CALL    BFLUSH          ;; <---
1359 ;
1360            LXI     D,0
1361            GETHL   FCB,FSZ         ;;FSZ=0?
1362            PUSH    H

1363            CALL    UCMP
1364            POP     H
1365            JNZ     SEEK2           ;;NO
1366 ;
1367            LHLD    NCBF            ;;NEW CBF < DRIVE SIZE?
1368            CALL    CDSZ
1369            JNC     ERR             ;;NO -- TOO BAD
1370            JMP     SEEK3
1371 ;
1372 SEEK2:    XCHG                    ;;DE=FSZ
1373            LHLD    NCBF
1374            XCHG
1375            CALL    UCMP            ;;NCBF<FSZ?
1376            JNC     ERR             ;;NO -- TOO BAD
1377 ;
1378 ; READ NEW BUFFER
1379 ;
1380 SEEK3:    CALL    NBUF
1381 ;
1382 SEEK4:    MVI     A,DCOMR
1383            STA     CMD
1384            PUSH    B               ;;MAKES JMP TO ERR2 WORK IN DSKIO
1385            CALL    DSKIO           ;; <---
1386            POP     B
1387            JMP     DONE
1388 ;
1389 SEEK5:    CALL    NBUF            ;;SET CBF, NBB, AND CHG
1390            JMP     DONE
1391 ;
1392 ;
1393 ;----- CCMODE(fd)
1394 ;           returns open mode of fd
1395 ;
1396 CCMODE:   ENTRY
1397            LXI     D,CPFD
1398            MVI     C,2
1399            CALL    GETP
1400            CALL    FINDF
1401            GETA    FDB,FMODE
1402            CALL    SXT
1403            RET
1404 ;
1405 ;
1406 ;----- CCFILE(fd)
1407 ;           returns addr of FCB else zero
1408 CCFILE:   ENTRY
1409            LXI     D,CPFD
1410            MVI     C,2
1411            CALL    GETP
1412            CALL    FINDF           ;;HL=FCBV
1413            RET
1414 ;
1415 ;
1416 ;----- CCPURGE(name)
1417 ;           deletes named file
1418 ;
1419 CCPURGE:  ENTRY
1420            LXI     D,CPNAME
1421            MVI     C,2
1422            CALL    GETP
1423            MVI     C,8             ;;BYTES TO MOVE
1424            LXI     D,FILNAME       ;;DEST PTR
1425            LHLD    CPNAME          ;;SOURCE PTR
1426 PURGE1:   MOV     A,M
1427            INX     H
1428            XCHG
1429            ANA     A
1430            JZ      PURGE2
1431 ;
1432            MOV     M,A
1433            INX     H
1434            XCHG
1435            DCR     C
1436            JNZ     PURGE1
1437            JMP     PURGE3
1438 ;
1439 PURGE2:   MVI     A,20H           ;;TERMINAL SPACE FOR DLOOK
1440            MOV     M,A
1441 ;
1442 PURGE3:   MVI     A,1             ;;DEFAULT DRIVE
1443            LXI     H,FILNAME       ;;HL -> FILE NAME, SP
1444            CALL    DLOOK           ;; <---
1445            JC      ERR             ;;NO SUCH FILE
1446            LXI     D,-8
1447            DAD     D               ;;BACKUP 8 BYTES
1448            MVI     A,20H
1449            MOV     M,A             ;;FORCE SP OVER 1ST BYTE OF NAME
1450            CALL    DMRIT           ;; <---

1451            JMP     DONE
1452 ;
1453 ;
1454 ;----- CCFLUSH(fd)
1455 ;           flush fd's buffer if modified
1456 ;
1457 CCFLUSH:  ENTRY
1458            LXI     D,CPFD          ;;FETCH CALL PARM
1459            MVI     C,2
1460            CALL    GETP
1461            CALL    FINDF           ;;SET FDBV & FCBV
1462 ;
1463            MOV     A,H
1464            ORA     L
1465            JZ      DONE            ;;IGNORE IF NO FCB
1466            GETA    FDB,FMODE
1467            ANA     A
1468            JZ      DONE            ;;IGNORE IF CLOSED
1469            GETA    FCB,CHG
1470            ANA     A
1471            JZ      DONE            ;;IGNORE IF NOT MODIFIED
1472 ;
1473            MVI     A,DCOMW
1474            STA     CMD
1475            PUSH    B               ;;MAKES ERR2 WORK
1476            CALL    DSKIO           ;; <---
1477            POP     B
1478            XRA     A
1479            PUTA    FCB,CHG         ;;ZERO CHG FLAG
1480            JMP     DONE
1481 ;
1482 ;
1483 ;----- CCALLOC(size)
1484 ;           allocates size bytes and returns the address
1485 ;
```

Don't let your hot utilities or computing ideas sit on the shelf. *DDJ* is a great place for bright authors to get a start. Contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

```
1486 CCALLOC: ENTRY
1487          LXI     H,2
1488          DAD     SP
1489          MOV     E,M
1490          INX     H
1491          MOV     D,M             ;;DE=MSIZE
1492          LHLD    FREEMEM
1493          PUSH    H               ;;OLD FREE MEM PTR
1494          DAD     D
1495          PUSH    H               ;;NEW FREE MEM PTR
1496          XCHG
1497          LXI     H,0
1498          DAD     SP
1499          CALL    UCMP            ;;NEW PTR < SP?
1500          POP     H               ;;NEW PTR
1501          POP     D               ;;OLD PTR
1502          JC      ALLOC1          ;;YES
1503 ;
1504 ACLUE:   MVI     B,41H           ;;"A" CLUE
1505          JMP     ABORT
1506 ;
1507 ALLOC1:  SHLD    FREEMEM         ;;UPDATE PTR
1508          XCHG                    ;;RET OLD PTR

1509          RET
1510 ;
1511 ;
1512 ;----- CCFREE(addr, size)
1513 ;           releases size bytes at addr
1514 ;
1515 CCFREE:  ENTRY
1516          LXI     D,CPSIZE
1517          MVI     C,4
1518          CALL    GETP
1519          LHLD    CPADDR
1520          PUSH    H               ;;BLOCK TO FREE
1521          XCHG
1522          LHLD    CPSIZE          ;;SIZE TO FREE
1523          DAD     D               ;;FREE ONLY IF LAST BLOCK ALLOCATED
1524          XCHG
1525          LHLD    FREEMEM
1526          CALL    UCMP            ;;MATCH?
1527          JZ      FREE1           ;;YES
1528 ;
1529          POP     H               ;;NO, SO RETURN ORIGINAL BLOCK PTR
1530          RET
1531 ;
1532 FREE1:   POP     D
1533          PUSH    D               ;;BLOCK TO FREE
1534          LHLD    EOP
1535          CALL    UCMP            ;;BLOCK PTR < END OF PROG?
1536          POP     H
1537          JNC     FREE2           ;;NO
1538 ;
1539          MVI     B,46H           ;;"F" CLUE
1540          JMP     ABORT
1541 ;
1542 FREE2:   SHLD    FREEMEM         ;;UPDATE FREE MEM PTR
1543          LXI     H,0
1544          RET
1545 ;
1546 ;
1547 ;----- CCAVAIL()
1548 ;           return number of bytes available for allocation
1549 ;           abort if stack overflow found
1550 ;
1551 CCAVAIL: ENTRY
1552          LHLD    FREEMEM
1553          XCHG                    ;;DE=ADDR OF 1ST FREE BYTE
1554          LXI     H,0
1555          DAD     SP              ;;HL=STACK ADDR
1556          XRA     A               ;;CY=0
1557          SBCM    D,E
1558          JC      ACLUE           ;;STACK OVERFLOW
1559          RET
1560 ;
1561 ;----- LOCAL ARITHMETIC & LOGICAL ROUTINES
1562 ;
1563 ;PUT THE ACCUM INTO HL AND SIGN EXTEND THROUGH H.
1564 SXT:    MOV     L,A
1565          RLC
1566          SBB     A
1567          MOV     H,A
1568          RET
1569 ;FETCH A FULL 16-BIT INTEGER FROM THE ADDRESS IN HL
1570 ;INTO HL
1571 GINT:    MOV     A,M
1572          INX     H
1573          MOV     H,M
1574          MOV     L,A
1575          RET
1576 ;STORE A 16-BIT INTEGER IN HL AT THE ADDRESS IN DE
1577 PINT:    MOV     A,L
1578          STAX    D
1579          INX     D
1580          MOV     A,H
1581          STAX    D
1582          RET
1583 ;
1584 ;COMMON ROUTINE TO PERFORM UNSIGNED COMPARE
1585 ;CARRY SET IF DE < HL
1586 ;ZERO/NONZERO SET ACCORDINGLY
1587 UCMP:    MOV     A,D
1588          CMP     H
1589          JNZ     UCMP1
1590          MOV     A,E
1591          CMP     L
1592 UCMP1:   LXI     H,1
1593          RET
1594 ;
1595 ;MULTIPLY DE,HL BY BC AND RETURN IN DE,HL
1596 ;(UNSIGNED MULTIPLY)
1597 MULT32:
1598          XCHG
1599          SHLD    MULTHIM
1600          LXI     H,0
1601          SHLD    MULTHIQ
1602 MULT1:   MOV     A,C
1603          RRC
1604          JNC     MULT2
1605          DAD     D
1606          PUSH    H               ;;ADD MSB'S
1607          PUSH    D
1608          LHLD    MULTHIM
1609          XCHG
1610          LHLD    MULTHIQ
1611          CADM    D,E
1612          SHLD    MULTHIQ
1613          POP     D
1614          POP     H
1615 MULT2:   XRA     A
1616          MOV     A,B
1617          RAR
1618          MOV     B,A
1619          MOV     A,C
1620          RAR
1621          MOV     C,A
1622          ORA     B
1623          JZ      MULT3
1624          XRA     A
1625          MOV     A,E
1626          RAL
1627          MOV     E,A
1628          MOV     A,D
1629          RAL
1630          MOV     D,A
1631          LDA     MULTHIM
1632          RAL
1633          STA     MULTHIM
1634          LDA     MULTHIM+1
1635          RAL
1636          STA     MULTHIM+1
1637          JMP     MULT1
1638 MULT3:   XCHG
1639          LHLD    MULTHIQ
1640          XCHG
1641          RET
1642 MULTHIQ: DW      0
1643 MULTHIM: DW      0
1644 ;
1645 ; DIVIDE DE,HL BY 2**C GIVING HL (QUOTIENT) AND DE (REMAINDER)
1646 ;(UNSIGNED DIVIDE)
1647
1648 P2DIV:
1649          MOV     B,C
1650 P2DIV1:  XRA     A
1651          ORA     B
1652          JZ      P2DIV2
1653          DCR     B
1654          MOV     A,L
1655          RAR                     ;;CY=LSB
1656          RARD    D,E
1657          RARD    H,L
1658          JMP     P2DIV1
1659 P2DIV2:  MOV     B,C
1660 P2DIV3:  MOV     A,B
1661          CPI     16
1662          RZ
1663          INR     B
1664          ANA     A               ;;CY=0
1665          RARD    D,E
1666          JMP     P2DIV3
1667 ;
1668 ;
1669 CC9997:  JMP     CCBOJ
1670 CC9998:  DS      3               ;;"JMP $+6" LOADS HERE
1671          DS      3               ;;NEXT MODULE, IF ANY, LOADS STARTING HERE
1672          PUSH    D
1673          LXI     D,$+6
1674          JMP     CC9997
1675 ;;FREE MEM STARTS HERE IF LAST MODULE LOADED WITH PROG
1676          ORG     CC9998
1677          JMP     $+6
1678          END
```


Even if you have never been published before,
*Dr. Dobb's* wants
to see your good computing ideas.
Contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# Small-C Grows Up

*"Yon harks the software wizard, who builds mythical castles of the mind; which yet, on second glance, cause wondrous things to occur."*

As a professional software wizard, I spend a fair bit of my effort toward building my tool chest to allow me to be more effective as a systems designer and programmer. Over the last four years I've built quite a set of tools, but I'm always hunting for a better choice than assembler. The 8080/Z80 family tends to be so low-level that even the simplest 16-bit operations require a number of instructions, enough so that the logic of straight-line assembler code tends to get lost in the details.

My interest in languages goes back some ways and my first functional attempt at a "better systems" language is an extended Pascal Compiler, that I call "ASP: Advanced Systems Pascal." ASP went a good way toward proving a reasonable alternative to assembler, yet it was too awkward and ill-defined to be really viable. It continues to be used; and when extended to be full ISO Pascal will again become a really useful tool.

In the meantime, my explorations discovered "C" and its predecessor "B" and "BCPL," languages with the orientation of good programming practice but hewn in the hot fires of real systems programming. C is exceptionally effective in allowing an experienced programmer to express the actions required in a form that lets the compiler generate highly efficient code without complex optimization. My favorite examples are: "++X and X+3", so much easier than trying to optimize "X:=X+3".

Enough prologue – onto the meat of the story. I had been playing with an experimental "C" compiler when Ron Cain's "Small-C" was printed in *Dr. Dobb's* issue #48. *Glory Day* – a much easier path presented itself. My progress was soon dashed as I found the BDS C compiler I'd been using could neither compile the whole source till I'd fragmented it, nor link the total result when done. Back to the drawing board – to form the gem of a new approach. By using the structure of Small-C, I divided the compiler into the basic elements of operation:

Pass 1 – Lexical analysis and tokenizing
Pass 2 – Syntax analysis
Pass 3 – Code generation

## Pass 1 – Preprocessor and Lexical

The preprocessor and a tokenizer become the first pass. First, the preprocessor is invoked and macro substitution takes place. Keywords and operators are recognized and coded as a single byte. Numeric and character constants are coded as 16-bit words. String literals and identifier names are stored in the literal pool and name table, respectively, and replaced with the resulting table offset.

The majority of the functions came directly from Small-C with the appropriate output actions added. Later on, additions were made: conditional compilations using "#ifdef" and, "#ifndef", "#else" and "#endif"; macro deletion using "#undef", B-tree logic for the macro and name tables; binary search on keywords and other optimizations.

The result of pass 1 is three tables in memory: token list, literal pool, name table. The macro table is built as needed and then discarded at the end of pass 1. The in-memory storage may seem limiting, however all modules in the over 256K bytes of final source can be compiled in a 48K machine, in 12 separate pieces; and the speed advantage is tremendous.

## Pass 2 – Syntax

The second step in processing a C program is syntax analysis, or parsing. Again the functions from Small-C were used as the starting point, especially "newfunc", "declvar", "statement" and the expression evaluator sequence of "heir" modules.

All logic relating to language testing was changed to reflect the token codes being processed, rather than raw source; for example "(match("if")==0)" became "(toknis(KIF))" where "KIF" is a "#define" of the value of the token for the keyword "if".

Similarly all keywords and operators were tested appropriately. Symbol processing uses a symbol table structure similar to Small-C, but now the name entry is only a pointer to the name table and all searches need only compare a 2-byte address. The symbol table was also extended to include an element size for structure ("struct") data. The original entries have had new values added to allow for structure definition, and extended the storage modes of static, parameter, initialized and structure member.

As the compiler developed, I extended the features available to include most of the full C statement syntax. Statements "do", "for", "switch", and "case" were fairly simple to add once the generated code was defined and the parse syntax was understood. Similarly many of the missing operators were added, such as "+=", "-=", "++" and "--".

Not quite so simple was the addition of more extensive data structures: "static", "extern" and "struct". The static and external data types fell into place by adding parameters to the "declvar" function. Adding "struct" required extensive changes to "parse", "newfunc" and "declvar" by adding a new "declstruc" function; which in turn calls "defstruc", and "vardetail". Much of the complexity is because structures may be within structures, and so the logic must be recursive. A "struct" declaration can be either a form description, giving the structure shape a name; or a data declaration, actually declaring some variables; or both. A structure consists of a named shape of a specific size, plus members who have no storage but an offset from the start of the structure. The operators "." and "->" had to be added and pointer/array arithmetic had to be expanded to account for structure element size.

Another area which required extensive work was function parameter handling and local stack maintenance. Small-C uses an ingeniously efficient use of the 8080/Z80 hardware stack pointer for local variables; however the design limited the lovely feature that C allows, a variable number of parameters in a function. Small-C did not allow this, due to the order of parameters on the stack and the reference of all local variables via the stack pointer. If a function was called with two parameters then the first might be "SP-6"; while if it was called with three parameters then the first would be

**by Richard Roth**

*Richard Roth, c/o InfoSoft Systems, 25 Sylvan Road South, Westport, CT 06880.*
*Copyright © 1981 by Richard Roth. All Rights Reserved.*

## Added from Original Small-C

Initializers

Structures

++,--,+=,-=

for, do, switch, case, default

Variable number of parameters

Direct relocatable output

#ifdef / #ifndef / #else / #endif/#undef

printf

extern, static, struct

"->" and "." operators

Command line controllable compiler table sizes

Auto chaining between passes

"~", "!" Boolean not & Logical not operators

Missing from small-C plus

goto / label

"||" / "&&" / "?:"

structured Arithemtic (other then int and char)

(PTR+PTR (vs) PTR+INT)

Can't initialize structures except using int pointers

Limitation of order of parameters declarations

Type cast

Bit fields

Register, Explicit automatic

&=,|=,/=,*=,%=,>>=,<<=,^=

"," operator

sizeof

union, short, unsigned

long, float, double

#if, #line

Macro with Parameters

Fortran/ASM Linkage without special ccent/ccret use

#Include "<name>" (Master disk search)

Compiler

| | | |
|---|---|---|
| Pass 1 - | 21K Source | 10K |
| Pass 2 - | 55K Source | 26K |
| Pass 3 - | 27K Source | 19K |

Library

| | |
|---|---|
| Root Code 16K | ~500 Bytes |
| Additional functions 21K | 8K If fully loaded |

*Dr. Dobb's Journal* is interested in a wide range of forward-looking topics in microcomputing. Let us hear about your ideas. Contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

Library functions

String: strlen, strmv, strnew, streq, strcmp

Initializer: __cinit

Standard files: copen, cclose,
cgetc, cputc, cread, cwrite,
cexit, ceof,
getchar, putchar, gets, puts

Formatted I/O: Printf

Memory allocation: memptr, alloc

Chaining: cchain

Misc:

call (ASM module call),
ccdos (I/OS & CPM call),
inp/outp (direct device I/O),
atoi / xtoi / itod / itox (ASCII-DEC/HEX conversions)

ROOT MODULE

ENTRY: ccmark, ccical, ccent, ccret, ccdosw (switch)

SIGNED: cclt, ccgt, ccge, ccge, ccle

UNSIGNED: ccult, ccugt, ccuge, ccuge, ccule

MISC: cceq, ccne, ccmult, ccdiv, ccmod, ccand, ccor, ccxor, ccneg, cccom

OPERATORS: ccsub, ccnot, ccasl, ccasr, ccmktf

"SP-8". There is no way the compiler can adjust for this at compile time. A related problem was some very messy code for allocating local variables, and clearing local variables and parameters.

All these probelms were resolved by introducing a variable storage mode of "parameter" which is a reference off the "BC" register pair, rather than the stack pointer. The "BC" register is maintained by using entry ("ccent") and exit ("ccret") routines, which also handle clearing local variables and parameters with a simple stack load using the "SPHL" instruction.

Various other changes were installed to complete and optimize the language accepted; the syntax is now close to true C. One limitation is that fixed by P. L. Woods in February 1981 *Dr. Dobb's*: "Required Parameter Order." This has not yet been fixed. (A note: the logic used by Mr. Woods looks good except for its failure to support C implied "int" declaration of parameters not otherwise declared.)

The full syntax of extended variable declarations has not been added so the classic, a pointer to a pointer to a char "**CP", must be handled using an assignment to a character pointer such as "*(CP = *IP)". Static or global data initializers have just been added, allowing for the declaration of tables of functions or strings.

## Pass 3 – Code Generation

As the new compiler became functional, code generation was removed from pass 2 and made a separate pass; pass 2 outputs code operations that the code generator expands. My tool kit includes our own relocatable assembler, linker and symbolic debugger, so it was essential to have compatible formats.

Initially the output was assembler code, but as I learned with my ASP compiler, using an assembler to save work for the compiler writer may work for a minicomputer with hard disk; but a micro with floppies is just too limited for this technique to be practical. Then a revised pass 3 (called 4) was made which is itself a 2-pass assembler, but relieved of the tasks of symbol processing and opcode mnemonics. Pass 4 first scans the code-ops file to determine the addresses

# Small C Grows Up (Text begins on page 46)

## Listing I

```
B.pnum t-btree.c
  1 ( 0): /* binary tree search logic - test                */
  2 ( 0):
  3 ( 0): /* based on Wirth's  A+DS = P                     */
  4 ( 0): /*     Search on pg 205                           */
  5 ( 0): /*     Delete on pg 211                           */
  6 ( 0):
  7 ( 0): #define tree_rt_size 6
  8 ( 0):
  9 ( 0): struct
 10 ( 0):        tree_root {
 11 ( 1):                int *tree_ptr;  /* strart of tree       */
 12 ( 1):                int *tree_len;  /* size of tree element */
 13 ( 1):                int *tree_alloc;/* memory alloc func for tree   */
 14 ( 1):                } ;
---------- function 0 -----------
 15 ( 0):
 16 ( 0): struct
 17 ( 0):        tree_element
 18 ( 0):                {
 19 ( 1):                char *tkey;     /* key pointer          */
 20 ( 1):                int  *tleft,    /* left branch          */
 21 ( 1):                     *tright;   /* right branch         */
 22 ( 1):                /*  user area goes after here          */
 23 ( 1):                int  tcount;    /* just count hits      */
 24 ( 1):                } ;
---------- function 1 -----------
 25 ( 0): #define tree_el_size  8
 26 ( 0):
 27 ( 0): getline(line)
 28 ( 0):   char *line;
 29 ( 0):        {
 30 ( 1):        line[0] = 80;
 31 ( 1):        ccdos( 10, line);
 32 ( 1):        ccdos(  2, 10);         /* add lf after input line      */
 33 ( 1):        return line+2;
 34 ( 1):        }
---------- function 2 -----------
 35 ( 0):
 36 ( 0): tallor(size)                   /* do tree space allocate       */
 37 ( 0):    int size;
 38 ( 0):        {  return alloc(size); }
---------- function 3 -----------
 39 ( 0):
 40 ( 0): int   *troot,
 41 ( 0):       *el;
 42 ( 0):
 43 ( 0): char  inline [82],
 44 ( 0):       status;
 45 ( 0):
 46 ( 0):



 47 ( 0): main()
 48 ( 0):       {
 49 ( 1):       troot = tinit(tree_el_size, tallor);
 50 ( 1):       printf("tree root at: %04x\n", troot);
 51 ( 1):
 52 ( 1):       while(1)
 53 ( 1):               {
 54 ( 2):               prt_tree( troot -> tree_ptr);
 55 ( 2):
 56 ( 2):               printf("Next >> ");
 57 ( 2):               getline(inline);
 58 ( 2):               if( inline[1] == 0) break;
 59 ( 2):
 60 ( 2):               el = tsearch( inline+2, troot, &status);
 61 ( 2):               printf("Search rtns: ptr= %04x,  status= %2x\n",
 62 ( 2):                                    el, status);
 63 ( 2):               switch(status)
 64 ( 2):                       {
 65 ( 3):                       case 0: ++ el -> tcount ; break;
 66 ( 3):                       case 1: el -> tcount = 0; break;
 67 ( 3):                       case 2: printf("** Out of room **\n"); break;
 68 ( 3):                       }
 69 ( 2):               }
 70 ( 1):
 71 ( 1):       prt_tree( troot -> tree_ptr);
 72 ( 1):       }
---------- function 4 -----------
 73 ( 0):
 74 ( 0):
 75 ( 0): prt_tree(tree)
 76 ( 0):   int *tree;
 77 ( 0):       {
 78 ( 1):       char *p1, *p2;
 79 ( 1):
 80 ( 1):       printf("Node: %04x    [%04x <<-->>%04x]\n",
 81 ( 1):               tree,
 82 ( 1):               p1 = tree -> tleft,
 83 ( 1):               p2 = tree -> tright ) ;
 84 ( 1):
```

```
 85 ( 1):        if( pl != ø) prt_tree(pl);      /* left         */
 86 ( 1):          else printf("No left fork at %ø4x\n", tree);
 87 ( 1):
 88 ( 1):        if (tree -> tkey != ø)
 89 ( 1):                printf("Value at %ø4x is [%s] for %d times\n",
 90 ( 1):                        tree,
 91 ( 1):                        tree->tkey,
 92 ( 1):                        tree->tcount );
 93 ( 1):
 94 ( 1):        if( p2 != ø) prt_tree(p2);      /* right        */
 95 ( 1):          else printf("No right fork at %ø4x\n", tree);
 96 ( 1):        }
---------- function 5 -----------
 97 ( 0):
 98 ( 0):


 Number of functions= 6
B.
```

**End Listing**

## Listing II

```
 pnum btree.c
 1 ( 0): /* binary tree search logic                    */
 2 ( 0):
 3 ( 0): /* based on Wirth's  A+DS = P                  */
 4 ( 0): /*     Search on pg 205                        */
 5 ( 0): /*     Delete on pg 211                        */
 6 ( 0):
 7 ( 0): #define tree_root_size 6
 8 ( 0):
 9 ( 0): struct
10 ( 0):        tree    {
11 ( 1):                int *tree_ptr;  /* strart of tree       */
12 ( 1):                int *tree_len;  /* size of tree element */
13 ( 1):                int *tree_alloc;/* memory alloc func for tree
14 ( 1):                } ;
--------- function 0 -----------
15 ( 0):
16 ( 0): struct
17 ( 0):        tree_element
18 ( 0):                {
19 ( 1):                char *tkey;     /* key pointer          */
20 ( 1):                int  *tleft,    /* left branch          */
21 ( 1):                     *tright;   /* right branch         */
22 ( 1):                /*  user area goes after here          */
23 ( 1):                } ;
--------- function 1 -----------
24 ( 0):
25 ( 0): /*     EXTERNALLY USED FUNCTIONS                   */
26 ( 0):
27 ( 0): /* tinit( tlen, talloc) - Build tree root initially   */
28 ( 0):
29 ( 0): /*   int tlen;         - length of a tree node
30 ( 0): /*                   tree_el_size + user needs     */
31 ( 0): /*   int *(*talloc()); - ptr to alloc function
32 ( 0): /*                   for tree allocated region     */
33 ( 0):
34 ( 0): /* tsearch(key, tree, insert) - SEARCH function
35 ( 0):
36 ( 0): /*   char *key;        - ptr to search string
37 ( 0): /*   int  *tree;- ptr to tree root, setup by tinit    */
38 ( 0): /*   char *insert;- otr to a variable or ø         */
39 ( 0):
40 ( 0): /* if insert != ø (i.e., a variable address (&var)   */
41 ( 0): /*   then if the search fails                      */
42 ( 0): /*      then a new entry is made and *insert = 1   */
43 ( 0): /*   else no entry is made and *insert = ø         */
44 ( 0):
45 ( 0): /* if tree_alloc function returns ø (no more room)   */
46 ( 0): /*    then insert fails and *insert = 2;           */
47 ( 0):
48 ( 0): /*


          */
49 ( 0): /*    initialize tree root with sentinal entry     */
50 ( 0): tinit ( tlen, talloc)
51 ( 0):   int tlen,  *talloc;
52 ( 0):        {
53 ( 1):        int *tree,  *p;
54 ( 1):        tree            = (*talloc) (tree_root_size);
55 ( 1):
56 ( 1):                /* init sentinal entry  */
57 ( 1):        tree -> tree_ptr = p    = (*talloc) (tlen);
58 ( 1):           p -> tright  =
59 ( 1):           p -> tleft   = ø;
60 ( 1):
61 ( 1):        tree -> tree_len        = tlen ;
62 ( 1):        tree -> tree_alloc      = talloc ;
63 ( 1):        return tree;
64 ( 1):        }
---------- function 2 -----------
65 ( 0):
66 ( 0): /* compare strings for equality or direction of !=    */
67 ( 0):
68 ( 0): tcomp ( s1, s2)
69 ( 0):    char *s1, *s2;
```

of all internal labels and function entry points, and then outputs relocatable object code, with a symbol table. For symbolic debugging, special line number labels are included to give the start of each line of the source.

### The Library

The final and perhaps most complex part of any language processor is the runtime library. For an 8080/Z80, this includes basic 16-bit math as well as more extended I/O functions. In addition, the "CCLIB" includes the low-level routines for module entry/exit and "switch;" the initializing routines for the command line parameters and disk I/O; and such high-level functions as "printf" and "quick sort". I've implemented the "portable C library" from an article by M. E. Lesk of Bell Labs.

### The Result

The result has been a very powerful programming tool. Not only is C a standard language, it is fast becoming the best and most available language for systems programming. Make no mistake about it, C is not for beginners or applications programmers, but for systems work it is fantastic. I've now used the compiler for some commercial work, and many in-house projects and am very pleased with both the productivity of the C language and the speed and efficiency of the InfoSoft C compiler.

The InfoSoft C compiler is available (to *Dr. Dobb's* readers) in two forms: A runable version for $50.00 in conjunction with our "SAL structured assembler" development tool kit (list price $225.00). Or source may be licensed for an additional $250.00 when the runable version is licensed. The C compiler is strictly "as-is" condition, although the "SAL" package is a supported product and includes the relocation, linking macro assembler with structured statements, the relocating overlay linker, and symbolic debugger. The package includes both Z80 and 8080 version. As an example, I've included the binary-tree routine used in pass 1 for the name table and macro table. See Listings I & II . DDJ

# Small C Grows Up

**(Listing continued, text begins on page 46)**

```
 70 ( 0):        {
 71 ( 1):        int c;

 72 ( 1):        while ( *s1)
 73 ( 1):                if ( (c= ( (*s1++) - (*s2++)))  !=ø)
 74 ( 1):                        return c;
 75 ( 1):        return - *s2;  /* equal if *s2==ø, else s1 < s2          */
 76 ( 1):        }
---------- function 3 -----------

 77 ( 0):
 78 ( 0):
 79 ( 0): /* allocate and assign a new sting key for entry of a new node        */
 80 ( 0):
 81 ( 0): tsnew ( key, talloc)
 82 ( 0):   char *key;  /* the key to allocate  */
 83 ( 0):   int  *talloc;        /* the allocate space function  */
 84 ( 0):        {
 85 ( 1):        int cnt;
 86 ( 1):        char *p1, *p2;
 87 ( 1):
 88 ( 1):                /* count new string size, with null      */
 89 ( 1):        p1 = key; cnt =1;
 90 ( 1):        while( *p1++ ) ++cnt;
 91 ( 1):
 92 ( 1):        p2 = p1 = (*talloc) (cnt) ;      /* allocate new key space */
 93 ( 1):        if (p1 == ø)
 94 ( 1):                return ø;        /* fails - no room left */
 95 ( 1):        while ( *p1++ = *key++) ;
 96 ( 1):
 97 ( 1):        return p2;
 98 ( 1):        }

---------- function 4 -----------
 99 ( 0): /*
100 ( 0):


          */
101 ( 0):
102 ( 0): /* tree search function proper                                      */
103 ( 0):
104 ( 0): tsearch (key, tree, insert)
105 ( 0):    char *key;          /* key to find                  */
106 ( 0):    int  *tree;                 /* root of tree                 */
107 ( 0):    char *insert;       /* ø or &var for insert status  */
108 ( 0):        {
109 ( 1):        char *p1, *p2;
110 ( 1):        int  dir, compv, *talloc;
111 ( 1):
112 ( 1):        talloc = tree -> tree_alloc;
113 ( 1):        p2     = tree -> tree_ptr;       /* root of tree */
114 ( 1):        p1     = p2   -> tright;         /* init for search      */
115 ( 1):        dir    = 1;
116 ( 1):
117 ( 1):        /* search the tree      */
118 ( 1):        while ( p1 != ø)
119 ( 1):                {
120 ( 2):                p2 = p1;
121 ( 2):                if ( (compv = tcomp(key, p1 -> tkey)) <ø )
122 ( 2):                        { p1 = p1 -> tleft; dir = -1;   }
123 ( 2):                else
124 ( 2):                if (compv > ø)
125 ( 2):                        { p1 = p1 -> tright; dir = 1;   }
126 ( 2):                else
127 ( 2):                        { dir = ø;  break;              }
128 ( 2):                }
129 ( 1):
130 ( 1):        /* insert if required   */
131 ( 1):        if ( insert != ø )
132 ( 1):           if (dir)
133 ( 1):                {
134 ( 2):                if( (p1 = (*talloc) (tree -> tree_len)) == ø)
135 ( 2):                        *insert = 2;     /* no room for new node */
136 ( 2):                else
137 ( 2):                if( (p1 ->tkey = tsnew(key, talloc)) == ø)
138 ( 2):                        *insert = 2;     /* no room for key      */
139 ( 2):                else
140 ( 2):                        {       /* build new node       */
141 ( 3):                        *insert = 1;
142 ( 3):                        p1 -> tleft  =
143 ( 3):                        p1 -> tright = ø;
144 ( 3):                                /* link to last entry   */
145 ( 3):                        if(dir <ø)  p2 -> tleft  = p1;
146 ( 3):                          else      p2 -> tright = p1;
147 ( 3):                        }
148 ( 2):                }
149 ( 1):           else *insert = ø;    /* insert requested by not needed       */
150 ( 1):
151 ( 1):        return p1;      /* return entry found   */
152 ( 1):        }
---------- function 5 -----------


 Number of functions= 6
B.
```

Why wait to start increasing your sales? Call now to reserve advertising space in the next *Dr. Dobb's Journal* – (415) 323-3111.

## Letters

*(Continued from page 4)*

only inside function definitions. The specific modifications are

a) make litabsz smaller (say 200 instead of 2000)

b) delete the line

```
dumplits();
```

from main()

c) just before the end of newfunc() (after locptr=startloc;), add the following:

```
dumplits(); /*dump the literal
              pool for the function*/
            /*now re-initialize
              the pool*/
litlab=getlabel();
litptr=0
```

I added the hooks for doing dynamic memory allocation and recovery. In trailer() I added one statement before ol('END'); it is

```
if (glbflag) ol("ENDDATA EQU $");
```

This gives me a value for the end of preallocated memory (*note:* the test is to keep from generating duplicate constants when compiling modules separately. However, this works only if globals are defined in one module and it is the last one assembled). Since the Small-c stack grows from high memory down, the area between the stack and preallocated memory is available as a heap. So, I wrote two storage management routines which manipulate the heap like a second stack which grows towards high memory.

```
alloc(b)
int b;
[  /*allocate a block of memory from
     the heap and
   *return the lowest address of the
     block. DO
   *NOT check for collision with the
     stack
   */
   #asm
   POP   H          ;return address
   POP   D          ;block size
   PUSH  D
   PUSH  H
   LHLD  HEAPTOP    ;address of cur-
                     rent top of heap
   XCHG
   DAD   D          ;HL<-new top
                     of heap
   SHLD  HEAPTOP
   XCHG             ;HL<-old top
                     of heap
   RET
HEAPTOP  DW ENDDATA
   #endasm
]
```

To allocate a block of memory simply call alloc with the number of bytes needed and assign the result to a pointer. Individual bytes (words) can be indexed off the pointer. The complementary routine to return the block (and all memory above it) is

```
free(addr)
int addr;
[    /*reset the top of heap pointer to
       addr*/
   #asm
   POP   D          ;return address
   POP   H          ;addr
   PUSH  H
   PUSH  D
   SHLD  HEAPTOP
   #endasm
]
```

I hope these modifications prove useful.

Sincerely yours,
Rodney Black
39 Elbow Hill Road
Brookfield, CT 06804

## Think Smart

Dear Compupholk,

I am currently a subscriber.

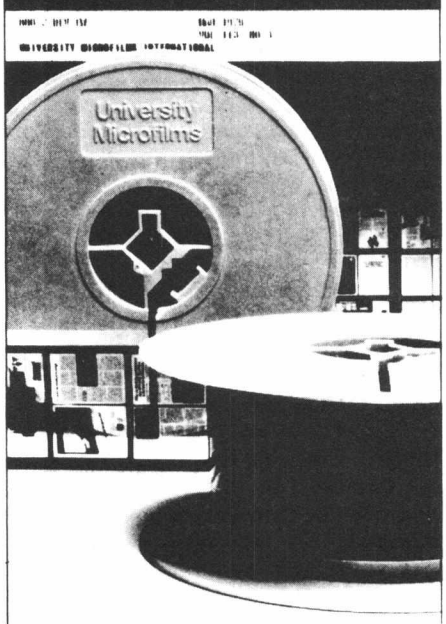
I won a TRS-80 16K Ext. B. Color. I would like software that will make it a smart terminal and not the dumb one that Radio Shack has done with Compuserve Software.

I would like to be able to dump the retrieved information into the cassette and printer.

Hope you can help.
John Reid
40-15 61st Street Apt 3-H
Woodside, NY 11377

*Remember the old saw "I paint what I see"? Well,* DDJ *publishes what it receives. We would be delighted to publish articles on the Color Computer and 16K Extended BASIC, if someone would send us some – including smart terminal programs for all of the Radio Shack (or other) mass-market computers. How 'bout it, folks? A program with simple user instructions and notes on special features? You send the software, and we'll provide the telephone numbers of CBBS's around the country which you can access . . .* DDJ


This publication is available in microform.




University Microfilms International

Please send additional information for _____

Name _____

Institution _____

Street _____

City _____

State _____ Zip _____

300 North Zeeb Road
Dept. P.R.
Ann Arbor, Mi. 48106
U.S.A.

30-32 Mortimer Street
Dept. P.R.
London W1N 7RA
England

# Essay on Hacking

## by Gandalf

*They were among the brightest computer-science students at Stanford. They spent their days and nights in the concrete-and-glass computer center playing harmless games and mock-vicious "wheel wars" on the terminals. Then, one night, a hacker (computer addict) whose code name was Gandalf sat down and tapped out a* cri de coeur*: in their single-mindedness, he and his fellow hackers were losing touch with the human race. Quickly the issue was joined. The complete printout follows. (For a glossary of hacker terminology, turn to page 57.)*

**Stanford University**
**Low Overhead Time-Sharing System**
**(LOTS)**

**From**: G. Gandalf
**To**: Bulletin Board
**Subject**: Essays on Hacking

Dedicated to all my firends at LOTS who will live their lives in an alien culture surrounded by humanity, and to Ernest, who was too human for it.

As much as an essay, this is a story. It is a true story of people paying $9,000 a year to lose elements of their humanity. It is a story of both seen and unseen games. It is a story of the breaking of wills and of people. It is a story of addictions, and of misplaced values. In a large part, it is my story.

There is no one field in particular in academia that has a monopoly on production of single-interest people, and this practice can exist almost anywhere. There is the policital-power seeker, all-consumed by climbing up the bureaucratic rungs. There is the stereotyped premed, ignoring all but his MCAT scores. There is the compulsive artist or writer, forever lost in his work. Narrowness is widespread. But there is one field that seems to be more consistent in this practice. This essay, rooted in personal and painful experience, is about the people in computer science.

In the middle of Stanford University there is a large concrete-and-glass building filled with computer terminals. When one enters this buiding through the glass doors, one steps into a different culture. Fifty people stare at terminal screens. Fifty faces connected to 50 bodies, connected to 50 sets of fingers that pound on 50 keyboards ultimately linked to a computer. If you go farther inside, you can discover the true addicts: the members of the Establishment. These are the people who spend their lives with computers and fellow "hackers." These are the members of a subculture so foreign to most outsiders that it not only walls itself off but is walled off, in turn, by those who cannot understand it. The wall is built from both sides at once.

These people deserve a description. In very few ways do they seem average. First, they are all bright, so bright, in fact, that they experienced social problems even before they became interested in computers. Second, they are self-contained. Their entire social existence usually centers around one another. Very, very few remain close to their families. Very very few associate much with anyone who is not at least partially a member of the hacking group. While they do sometimes enjoy entertainment unrelated to their field, it is almost always with fellow hackers. Third, all aspects of their existence reinforce one another. They go to school in order to learn about computers, they work at jobs in programming and computer maintenance, and they lead their social lives with hackers. Academically, socially, and in the world of cash, computers are the focus of their existence.

The hacker will probably not strongly disagree with what has been said so far. But he will ask the question, "So what?" The answer is: in creating a subculture and isolating it, we are destroying the chance that computers might be used wisely as an integral part of our society. We are precluding the human values so necessary for the wise application of this technological achievement. The most brilliant young minds at our top universities are learning how to play with multimillion-dollar toys first, and how to utilize them constructively second.

Even if we ignore the costs to society as a whole, we have to look at the costs to the people involved. The computer is a modifier of personalities. It is highly addictive. People who gain this addiction for a period of several months tend never to give it up. And the symptoms are very sad.

The first thing to go is other academic interests. Basically what occurs is that the hackers' motivation to challenge themselves in any field not directly linked to computers gradually disintegrates. On the level of grades, straight-A students tacitly accept C's in noncomputer courses. On the level of actual learning, the same students shut off outside subjects even more completely than their grades would indicate. This is common in many areas of specialization, but nothing compares with the incredible consumption of computer-science students for computer-science courses, and their nonchalant attitude toward every other class.

The second thing to go is a normal living pattern. Eating and sleeping are completely rearranged to fit the addiction. The typical hacker thinks nothing of eating one meal a day and subsisting on junk food, or of sleeping from 4 a.m. to noon almost every day of the week. Families are soon disregarded, to an extent uncommon even when one considers the separation that generally occurs in college. It is simply that the parents of hackers are ignorant of the subculture and cannot relate.

The third thing to go is a balanced social life. The hackers' narrowness and strange schedule simply compound the social problems they experienced before hackerdom. Soon, no one except a hacker has the capability to understand other hackers. No one except a hacker will go out with other hackers. No one except a hacker can talk to another hacker.

The fourth and final thing that happens is also the saddest. The personality of the hacker shifts, in order to permanently adjust to the new social conditions. Emotions always have hurt before, so they are effectively isolated. Relations with nonhackers become strained, so why force the effort? It is so much easier just to accept social rejection and isolation, and to do it with a spirit of camaraderie that's shared by the rest of the subculture.

An essay should make an attempt to resolve the problem it points out. In this case, the pointing may be enough, or at least enough to do whatever can be done. I know from personal expereince what a trauma it can be. I was one of the top 10 among several thousand LOTS users last spring for the amount of time I spent here. I have watched people close to me undergo the transformation. I narrowly escaped it.

The tragedy is that I am so involved in piecing my personality and social life back together that I think I have learned very little about how to prevent this from happening in the first place. I am lucky.

*Reprinted from* Psychology Today *magazine. Copyright 1980, Ziff-Davis Publishing Co.*

I will go on to some sort of a balanced life (although my hacker friends will laugh at me, since, to them, my involvement was never serious enough to make me one of them). Weak-willed people, people with unstable social lives, people in formative stages of their lieves, should not become involved in computer science. It should be left until they are truly able to make decisions and be aware of all the consequences. Computers are most often used by people who start when they are immature. This is what causes the single-minded addiction. This is what takes some of the brightest and most capable minds in college today and turns them to narrowness.

**From**: E. Ernest
**To**: G. Gandalf
**Subject**: Reply to "Essay on Hacking"

As the human to whom Gandalf's essay was in part dedicated, I feel I have a responsibility to reply. Obviously, his essay is going to provoke a certain amount of defensiveness from the hackers; on the other hand, he has made a number of points which are indeed very valid.

I became rather disillusioned with LOTS last spring because I found that the attitudes of the people whom I had made my friends were becoming more and more machine-oriented and less and less human-oriented. I began to spend less time here, and see the people less often. It was partly the so-called narrowness to which Gandalf referred, but it was also a simple lack of genuine human caring on the part of these people that I left.

I'm not concerned about their social lives: physics majors, premeds, law students, the regulars at Aunt Minnie's church social – all behave in roughly the same way. Nor am I concerned about their grades, for the same reason that I'm not concerned about anybody's grades, including my own; if they choose to get hassled over grades, they'll do so, and they'll make them if they want them. I AM concerned about the narrow-mindedness that comes from losing interest in such outside things as the draft or the social sciences, or all the various and sundry things that Stanford has to offer. This place should be a broadening experience; to fail to take advantage of that is a waste of $9,000. Furthermore, narrow-mindedness and xenophobia have started more wars than any other cause.

More than anything else, I'm concerned about the hacker's shortage of respect for others and a lack of sympathy for their propblems, because he considers the problem of whether or not he wants a HRROI or a STI in a certain place to be more fun than people are. (If I get a lot of mail telling me that that's a ridiculous example and no one could possibly mistake a HRROI for a STI, it will only prove my point.) By and large, the hackers are consultants, and as consultants they owe it to the users to be patient, pleasant, and, above all, sympathetic. A large number of people in introductory computer-science courses are not cut out for computer work: the thought processes it requires have long ago been squashed by indoctrination and training. They are confused, frustrated, and upset. We do the best we can for them by teaching them PASCAL and giving them an interactive system with a forgiving exec which holds their hands at every turn. But there is nothing like human comfort to make people feel better about it.

Gandalf's problem was in attributing these faults to the computer. The machine itself is not responsible for it: the attitudes of the others toward the machine are to blame. After having spent a summer doing data processing, I know that not all computers have this fascination. Look at J.Q., Vic, Sean, or Karl. They don't demonstrate these attitudes. They know that, fascinating as it may be, the computer is a tool and not a toy. The fault lies in the attitude of the individual.

What is to be done? Nothing is to be done. The hackers know who they are, running around and having "wheel wars" late at night, yakking and yakking while someone stands and waits for them in front of the consultant's terminal. We may only hope that they grow out of it. Gandalf is wrong. It isn't a permanent alteration in their character – it is, hopefully, a short-lived immaturity of people on their way to becoming programmers and computer scientists.

**From**: K. Kanef
**To**: E. Ernest
**Subject**: In Reply to Reply Txt

In your opinion, the two problems that LOTS hackers have developed are a loss of touch with their humanity and an immature attitude about the purpose of the computer. Until recently, I would have disagreed with the first assertion. I'm no longer so sure that it's not true.

I have even stronger feelings about the other claim that you make about

*Dr. Dobb's* readers
have been aiding
the growth and development
of the micro field
for years.
If you would like
to make a contribution,
contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

hackers: that they have an immature view of the purpose of the computer. You assume that childishness is a thing to be avoided. You encourage people to throw away the things that are so hard to retain in growing up: the sense of wonder in new wonderful things, the ability to PLAY. When I compare my attitude to that of Vic and the others you mention as having a more mature view of the computing world, I find that I like mine better. I don't want to be like them! I see the computer as a toy as well as a tool. And as I pointed out to Gandalf last week, this attitude is not limited to hackers. There were many people in my dorm last year whose only knowledge of computers was how to play games and send mail, and they enjoyed using the computer as much as I did. I was one of the few people in that group who was interested enough in the field of computer programming to also take an intro course in PASCAL.

I hope that I will always be as fascinated by my work as I am now. Unfortunately, I know that this is not likely to happen; all too often people become bored with their professions. You're right in saying that the childishness of LOTS hackers is short-lived – and isn't it a shame?

**From**: E. Ernest
**To**: K. Kanef
**Subject**: The Computer As Toy

I am impressed by your skeptical intelligence, and I appreciate your reply. I am sorry that you became aware of the diminution of simple humanitarian caring on the part of hackers the hard way, but pleased that you know of its existence.

As regards play, you present a very convincing argument, but I would like to add a qualifier. To me, as one matures, the highest forms of play cease to be consumptive and begin to be productive. Sure, we all need to play, and there is nothing wrong with consumptive play in an appropriate setting and appropriate circumstances. Middle-aged adults enjoy Great America, and this to me seems to be a perfectly healthy and appropriate form of consumptive play. Nonetheless, I don't believe it is the highest form of play that they should be capable of.

Computer games are tolerable and serve their purpose; even they help to acquaint new users with the computer and to show that it is not a thing to be frightened of or confused by. But programs that type "You lose" when you type "sigh" to the exec don't serve any productive purpose at all; they are vaguely amusing and they take up disk space, and an entire page for six lines of code, at that. Likewise, the wheel war is neither educational nor productive, it is nothing more than consumptive play, and it can be annoying.

**From**: M. MRC
**To**: G. Gandalf, E. Ernest
**Subject**: None Given

Most of the behavior you describe is more or less typical of an undergraduate-aged person and less typical of hackers as they get older. Rather than demeaning it, we should simply recognize it as normal for somebody that age. In a high school kid, it is acutally a sign of maturity to be at that stage so quickly. In a graduate student, it's regressive.

Don't be misled by those who postulate the "compulsive" (and hence parasitic) programmer versus the "professional" programmer – I know of no programmer who does not consider the computer as a toy first. You really do have to enjoy this work to be good at it. I know of no hacker who does not consider the computer as a toy first. What changes are the ways that s/he plays with the toy. There isn't all that much difference, in terms of the pleasure derived, between writing a new display hack or a clever access-control program, except that the former is something derided as being "hacking" and the latter is "responsible programming."

I guess my point is that it's all a part of the irresponsibility that college undergraduates normally have, and it shouldn't be begrudged them; it's the last chance they are ever going to have to be that free.

**From**: A. Anonymous
**To**: E. Ernest, K. Kanef
**CC**: G. Gandalf
**Subject**: The Hacker Chronicles

We are dealing with an infintely malleable tool. People who choose to develop and use that tool, whether for work, play, or both, have that choice and cannot be denied it. A person who chooses to be a musician must devote hours and hours to gain adequate expertise. But would you consider the computer hacker any less creative than such a person? I certainly wouldn't. The computer serves not only as a workhorse, but also as an easel for exercising one's creative abilities. Therefore, in my opinion, the hacker has not limited himself at all. Rather, he has expanded his intellectual horizon because now he has the infinite tool.

As for the charge that it disrupts one's social life, I would tend to agree with this to a point. But it depends on how controlled the individual is. At any time, he can withdraw to a more normal schedule. Why doesn't he? The reason is obvious. The infinite tool that knows few boundaries is accessible to a much higher degree, and thus he can devote more time to it. Why is this wrong? I think it is definitely a bonus, since the usual restraints of 9-to-5 are eliminated and the person is allowed to expand beyond boundaries

Why wait
to start
increasing your sales?
Call now
to reserve
advertising space
in the next
*Dr. Dobb's Journal*
(415) 323-3111

to do what he wants.

Now we come to the human versus the machine factor. Gandalf stresses the necessity of human interaction and the inherent evil of the machine. Would you stress the evil of the instruments in an orchestra, or the instruments in a laboratory, or the typewriter of an author? All of these occupations demand extraordinary amounts of time for excellence. But I see no greater human interaction in these fields than in computers. I feel that people who disparage computers for a seemingly decreased human interaction are not at all familiar with the true import of the computer. Not only is it the infinite tool, it is also an extremely fluid medium of communication.

**From**: K. Kuddles
**To**: G. Gandalf
**Subject**: Hacking and the Real World

I don't want to turn this into a personal defense or anything like that, but as I've said before, people have always found me strange. I have always worn my hair long because I felt it was comfortable for me. I prefer clothes that are comfortable and perhaps a bit scroungy. For this, and being human, and getting excited, I have gotten pretty bashed. As a matter of personal history: it was nice for me to get away from a smog-hole of a city, where the mayor said aloud on TV once that the "black problem" existed because the "blacks weren't disciplined properly." I spent time in an ad agency where people were not only truncated and anesthetized, but positively deformed in personality. They viewed people as real, honest-to-goodness objects of $$ and/or pieces of meat. They'd promise you anything, but split with your portfolio.

Working with computers has been the only thing tolerable I have done in quite a while. It does have its drawbacks, and you can get into just doing that and not trying anything else. Part of it derives, I think, from a strange version of the American work ethic. The thing I find sad is when hacking comes before people; and it does sometimes for me, usually where $$$ and all this financial-survival crap is concerned. But it is saddest when someone will not take time out from hacking to be with people at all. I mean when people get angry because someone sends them a message asking them what they are up to, and the other person gets all freaked-out and sends nasty notes back.

The problem is that the social structure of LOTS rewards people who hack well but don't interface well with people. If you look at who gets most rewarded at LOTS and for what, you find it is the people who do programs and little maintenance tasks. Very seldom do people who get along with others and relate well to them get highly rewarded.

**From**: G. Gandalf
**To**: A. Anonymous
**Subject**: A Reply

You base much of your argument on comparisons (e.g., with musicians). I agree with you in that I specifically pointed out that many people besides computer-science majors are very narrow. This is no justification, however. As the son of an artist, I think I can say that such creative outlets are essential, but they do not and should not make up the totality of one's existence.

Secondly, you accuse me of stressing the evils of the machine world. This is an interpretation of my essay that is purely yours. I attempted to make clear that I thought computers could be highly useful tools, but that much of their valuable potential was being lost due to the narrow approach to them. It is a HUMAN failure and not a mechanical one.

**From**: E. Ernest
**To**: G. Grog, W. Woody, H. Hima, B. Berlin, 0.07, K. KDA, Y. Yduj, S. Squash, J. Q. Johnson
**Subject**: The Hacker Papers

Dear Sir or Madam,

You have contributed NOTHING, repeat NOTHING, to the Hacker Papers. As a hacker, it is your responsibility to defend yourself against this vicious attack on your way of life. Do you intend to stand idly by while mealy-mouthed, bleeding-heart, fuzzy-studies majors shred you? Surely not! Voice your opinion! Stand up for your right to enjoy not enough sleep, not enough terminals, not enough core! What people do with consenting computers in private is their business!

**From**: J. Q. Johnson
**To**: E. Ernest
**Subject**: The Hacker Papers

What, pray tell, are the "Hacker Papers"? A work by Margaret Hacker? A lurid tale of ax murder? A discussion of the psychology of computer nonprogramming? If the last, I have nothing to add, and am not sure I appreciate being addressed as a hacker; I'll have you know that I AM a "mealy-mouthed, bleeding-heart, fuzzy-studies major," since my degree is in political science, which (next to AI) is the fuzziest of studies.

Our high-quality editorial and your high-quality products are made for each other! Call (415) 323-3111 to reserve your space in the next *Dr. Dobb's Journal*.

My feelings are aptly summarized in Joseph Weizenbaum's *Computer Power and Human Reason*, where he talks about hackers who become so obsessed in tinkering with their grandiose programs that they lose sight of the problem to be solved. Though I disagree with his ultimate conclusion that the computer cannot be taught to reason like human beings, I tend to agree with his diagnosis of the effect they have on people's ability to reason. I most heartily recommend the book for your consideration. ▶▶J

Most of the contributors to "The Hacker Papers" are either still Stanford undergraduates or are working as computer specialists. In order of their messages, they include Kenneth Peter (G. GANDALF), Ernest Adams (E. ERNEST), Robert Kanefsky (K. KANEF), Mark Crispin (M. MRC), Steve Kudlac (K. KUDDLES), and John Q. Johnson (J. Q. JOHNSON). A. ANONYMOUS stands for a hacker who did not wish to be identified. Credit for discovering the papers goes to psychologist Philip Zimbardo.

## HACKER GLOSSARY

**Hacker:** A person who makes constant small and unimportant modifications to a computer program for enjoyment rather than to improve the program.

**Hacker display/Display hack:** An alteration in a computer program, done sloppily, and, often, for fun. The kind of program involved is for display of information on a terminal screen.

**PASCAL:** A relatively new computer language (like BASIC or FORTRAN).

**Wheel Wars:** Wheels are computer users who obtain special privileges. Their wars are games played on computers in which they do vicious things to one another, like blocking other wheels' access to the terminals or trying to confuse and frustrate them. (The wars occur only during late night hours or in summer, when they won't interfere with the work of other programmers.)

**AI:** Artificial Intelligence, the study of how computers can be programmed to simulate various intellectual processes of human beings, such as language use.

**Joseph Weizenbaum:** A well-known AI researcher at MIT who condemned hackers in his book, *Computer Power and Human Reason*. Compulsive programming, Weizenbaum wrote, "represents a psychopathology that is far less ambiguous than, say, the milder forms of schizophrenia or paranoia."

**HHROI, STI:** Esoteric machine instructions used mainly by wheels and computer repair people.

**Exec:** The executive command processor, the program that carries out the user's orders.

**Core:** The work space available in the computer, its internal memory capability.

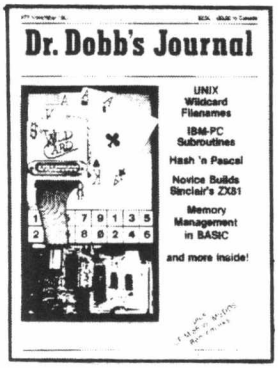

# Dr. Dobb's Journal

For Users of Small Computer Systems

Each issue includes:
- valuable software tools
- algorithms & problem solving
- industry news
- important product reviews

With in-depth coverage of:
- telecommunications
- systems programming
- language development
- machine independent programs

and much, much more!




Yes! Please enter my subscription for
☐ 1 yr. (12 issues) $25 ☐ 2 yrs. $47 (save $13 off newsstand)
☐ Please bill me ☐ I enclose check/money order

Name_____

Address _____

City_____ State_____ Zip_____

**Dr. Dobb's Journal, A Publication of People's Computer Company**
**1263 El Camino Real, P.O. Box E, Menlo Park, CA 94025**

## Are you ready?

*DDJ*, the world's foremost microcomputer publication, has been working for years to prepare its readers to be innovators, to lead the wave of breakthroughs in our changing technology.

Every issue of *Dr. Dobb's Journal* helps one to understand the nuts and bolts of small computer systems. We offer entire listings of valuable software: our pages have included compilers, cross-assemblers, editors, new languages, hardware interfaces and more – usually before anyone else thinks of them!

## Even more important!

Our subscribers share insights, correspond, and contribute to one another's work, more than any other group we know. They treat *Dr. Dobb's Journal* as a "hands-on" publication.

This warm cooperation has done more to refine software products, and generally to advance the state of microcomputer technology, than perhaps any other resource. And it is available to you through our pages!

## For the straight facts . . . . . .

If you are a serious computing professional or enthusiast, then you should take a very close look at what *DDJ* offers you. We've been on the cutting edge since 1976.

# DR. DOBB'S CLINIC

by D. E. Cortesi, Resident Intern

## Ep, Son of Ep

In the August Clinic we published a plea for help in dealing with the special features of the Epson MX-80 printer. Taylor Caldwell of Burgess, VA couldn't figure out how to embed wide or emphasized characters within an ordinary line. His request produced several good responses. One, an article-length effort by Gary P. Novosielski will appear next month.

Billy Smith (address unknown; we lost his envelope in our highly organized filing system) sent a letter filled with examples of special-font characters in midline. He says that double-width characters are toggled on and off by ASCII SO and SI (control-n and control-t) bytes surrounding the would-be wide words. Wide characters can apparently be switched on and off at any time.

The other special fonts – compressed, emphasized, and double-strike – take effect over an entire line, regardless of where in the line the special characters that evoke them appear. A line that contains ESCape, E, for example, will be printed in emphasized mode whether the ESC, E pair comes at the beginning, in the middle, or at the end. This effect can be tamed, Smith says, by printing a line in two passes. Print the normal part of the line and end it with a CR but not a LF, thus:

```
LPRINT "NORMAL";CHR$(13);
```

then overprint the special part:

```
LPRINT CHR$(27);"E";
     "          EMPHATIC"
```

It seems to us that the spacing could get tricky when compressed or double-strike characters are used.

Finally, we received a letter from Steve Semos, an employee of Epson America in Torrance, CA. He wrote first regarding Caldwell's misconception that "a control code embedded in the text may be recognized or not, depending on the direction the print head is to go." Not so, Semos says, "When a byte is received at the printer it is placed in a one-line buffer. The data will be printed when one of the following occurs: the buffer becomes full, a carriage return is received from the computer, or when a linefeed is received. The data is printed by a microprocessor in the printer . . . It will format the data in the same manner regardless of the print direction."

Semos notes that some BASICs swallow ASCII control codes. "The ASCII control codes have values from 0 to 31. Some computers will intercept these codes instead of sending them to the printer. This can be avoided by adding 128 to the desired control character number. The Escape code is decimal 27; plus 128 makes it 155. When sent through BASIC the computer will pass 155 to the printer, and the printer will understand this as the ASCII code ESC."

He also noted that the expanded character mode is switched off by the end of a line (a carriage return or linefeed) as well as by the SO control code, so that, for example

```
10 LPRINT CHR$(14)
20 LPRINT "THIS WOULD BE PRINT-
   ED NORMAL"
30 LPRINT CHR$(14);
40 LPRINT "THIS PRINTS EXPAND-
   ED"
50 LPRINT CHR$(14); "EXPANDED";
60 LPRINT CHR$(20);" NORMAL."
```

## Kruskal's Disassembler

Reader Mark Northrup of Milwaukee, WI presents a programming problem that probably has no good solution, but which raises some interesting questions of probability. Mark writes:

> When you're disassembling object code and encounter a symbol table, the disassembled code is garbage. When the table ends, my problem begins. When does the code line up and become meaningful again, and how can you tell? Martin Gardner provides evidence that the code will line up with an 83% probability within 52 bytes or less (see "Mathematical Games," *Scientific American*, February 1978).

Northrup enclosed a copy of the referenced column. The item, too long to quote here, concerns a mathematical curiosity dubbed "Kruskal's Principle." Gardner presented it in terms of counting chains of cards as they are dealt from a deck, but the principle clearly would apply to the disassembly of a machine code containing instructions of varying length. It ought to be possible to come up with a statement of the probability that a disassembler would be back in synch within some number of bytes. Might it never regain synch? As to "how can you tell," that seems to be a question of another order of difficulty entirely.

## Stop, Heath!

Terence Hills of Don Mills, Ontario wrote to tell of a dangerous quirk in the Heath H8. The H8 has a software-controlled front panel. A timer interrupt gives control to the panel monitor code every 2ms. If the monitor detects that the CPU was in the halt state, it awaits user input; otherwise it resumes execution of the program. Hills says that the way the monitor detects a prior Halt can cause problems:

> The monitor pulls the interrupted program counter from the stack, decrements it, and checks the addressed location for 166 octal [an 8080 HLT instruction, 76 hex]. The problem is, this byte might be immediate data, an address, or a variable. Even if it is an opcode, it may not have been the last one executed. Nevertheless, any of these things will send the machine into Monitor mode.

The more we thought about this, the worse it looked. The monitor must decrement the stacked PC to find a HLT, but it could equally well find the third byte of "CALL 7600H" or the second byte of "MVI A,118", or a "DB 'v'" that preceded a branch target. An unplanned return to monitor mode could occur with complete unpredictability – disaster for a program meant to run unattended. Maybe a utility could be written to search an object program for such pseudo-halts and flag them.

## The Root of the Matter

Once in a while, perhaps in a graphics routine, one might need to take a square root. Even if the problem is confined to integers, the thought of doing so in assembly language is a daunting one. One of the Clinic's more frequent respondents, L. Barker of Chicago, IL, is not daunted. He writes:

> A number of ways have been presented to approximate the hypotenuse of a right triangle. However, the more accurate solutions involved multiplication and division. If the square root was approximated because of difficulty in programming, then see program SQRT1 (Listing 1, p. 60). Limiting input to unsigned numbers of 16 bits gives SQRT2. This takes a maximum time of

9K T-states and is less than 15 bytes long. For people in a real hurry there is SQRT3 (both in Listing 2, p. 60). This runs a worst case time of about 1200 T-states which is a little faster than a 16- by 16-bit multiply. Adding a divide to the end of any of these programs doubles the precision of the computed answer. When the Z800 comes out next year with the Z80 instruction set and multiply and divide, this will all look like a waste of time, but until then, enjoy.

Well, the Z800 won't be replacing all Z80's overnight, you know. Still, we'd all like to know more about that chip; has a reader any hard information?

## Thirty Days Hath Septober . . .

We published a pair of calendar routines in the July Clinic (#57, p. 41). Edward S. Dayhoff of Silver Springs, MD has written to correct some misconceptions and our sloppy terminology. "These programs," he says, "have only limited validity ranges because of the chaotic nature of the calendars used throughout history. In addition, the term 'Julian Calendar Date' is completely erroneous." He continues,

> The Julian *Calendar* is completely unrelated to the Julian *date* as a short perusal of an encyclopedia will show. The program by C. T. McDaniel uses the Julian *date*. The Julian *calendar* is the calendar without century-adjusting leap years which came down to us from Augustus Caesar and which was used throughout the "civilized" world until reformed by Pope Gregory XIII in 1582. The reformed calendar was adopted individually by each country at different times and was known as the Gregorian calendar. The programs both produce correct *Gregorian* calendar dates, but be cautioned that they will produce them for years before the Gregorian calendar was in use! The "universal" calendars found in some reference books may suffer from the same problem.
>
> The Gregorian calendar was actually decreed by the Council of Nicea in 325 but was not used. The errors of the Julian accumulated and when Gregory published the reformed calendar he insisted that the errors be corrected retroactive to 325. That amounted to 10 days. Thus the day after October 4, 1582, was October 15, 1582. The calendar change was enacted immediately in France, but in England and in the British colonies in America the change took place in 1752, by which time the error was 11 days. In that year September 2 was followed by September 14! Another change was made at the same time. Previously the first day of each year was taken as March 25, Annunciation day. Thus March 24, 175*1* was followed by March 25, 175*2*, while December 31, 1751 was followed by January 1, 1751. You fellows who like to program calendars, have a go at that one!

Dayhoff supplied changes to McDaniel's code, resulting in the routines shown in Listing 3, p. 60. The second calendar program in issue #57 (another L. Barker effort), suffered badly at the hands of two typesetters. Your Intern's word processing software swallowed all the ">" and "<" symbols, and the regular typesetter lost parts of other lines. It's such a nice little program that we're printing it again (Listing 4, p. 61).

## Basically Difficult

The USR function of Microsoft BASIC is a nasty, but often necessary, way to escape into machine code to accomplish something that can't be done well enough (or at all) through the interpreter. One thing that can't be done through the interpreter is to gain access to CP/M's RDR and PUN logical devices. Scott Nowell, of Nashua, NH needed to drive a programmable industrial controller that was interfaced through the RDR and PUN devices. "The problem was to write a program that would send a string to the controller and get another string returned by the controller." Nowell found the syntax of the USR function deceptive:

> It says in the manual that when a string is passed to the user subroutine the DE register pair points to three bytes called a string descriptor. Byte 0 of the descriptor contains the length of the string (0 to 255). Bytes 1 and 2 are a pointer value to the starting address of the string. Although it does not say that a string can return a value to the program, the whole intent of the USR command is to do just that. An example is
>
> I$ = USR(A$)
>
> It seems rather obvious that this should assign the value returned by USR to I$. Unfortunately it is really equivalent to I$=A$; whatever the USR function returns in the registers is discarded.
>
> We first determined that we could modify the actual string that MBASIC passed to us, if we didn't change its location or length. So our routine put the returned string right back where the original was. This causes the statement to work as we wanted. In order for this to work with any length of input string, we decided to always output strings with a fixed length. A second requirement was not to send junk on the end of the string. That is done with an end of string mark. The following statements accomplish it:

```
 5 DEF USR=&HE612
10 I$=SPACE$(80)
20 INPUT "ENTER DATA:";A$
30 LSET I$ = A$ + CHR$(255)
40 I$ = USR(I$)
```

> This does limit the incoming string to be no longer than the outgoing one, but it accomplishes the job.

## Share the Wealth

This Clinic contains several exemplary "sponses" – readers' revelations that inform, warn, or stimulate. The Clinic thrives on such input. The Micro Logic Corp. (POB 174, Hackensack, NJ 07602) sent us a very nice plastic 6502 reference card. We'll send it out to the next author of a "sponse" who requests it.

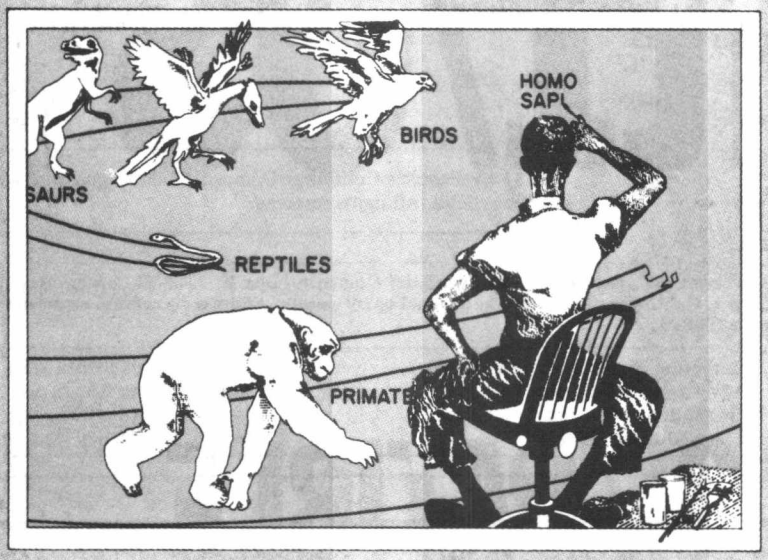
Next month we will present Octametry, a new branch, or twig, of mathematics, and such other gleanings from your fields of endeavor as may turn up by then. DDJ

$2.50
($3.00 in Canada)

# DR. DOBB'S JOURNAL

For Users of Small Computer Systems

Number 62 December 1981 Volume 6, Issue 12




**Actors, Property Lists, and LISP**
*Ronald L. Nicol*

**Automatic Loading of Assembly Language Subroutines**
*John B. Palmer*

**Small-C Expression Analyzer**
*J. E. Hendrix*

**Improved MX-80 Printer Driver For CP/M BIOS**
*Gary P. Novosielski*

**and much, much more!**

**Artificial Intelligence for Micros?**

**Powerful Memory Test for 8080's . . .**

**. . . more on Small-C!**

DR. DOBB'S JOURNAL
For Users of Small Computer Systems

December, 1981 Volume 6, Issue 12, Number 62

# CONTENTS

**Actors, Property Lists, and LISP** . . . . . 12
*by Ronald L. Nicol*
**Marching the 8080** . . . . . 20
*by James Laurino*
**Improved MX-80 Driver for CP/M BIOS** . . . . . 24
*by Gary P. Novosielski*
**Connect a Parallel Printer to Your NorthStar** . . 35
*by Alan Bomberger*
**Small-C Expression Analyzer** . . . . . 40
*by J. E. Hendrix*
**Automatic Loading of Assembly Language Subroutines** . . . . . 44
*by John B. Palmer*

**Departments**

Letters . . . . . 4
Dr. Dobb's Clinic . . . . . 6
Of Interest . . . . . 8

© 1981 by People's Computer Company unless otherwise noted on specific articles. All rights reserved.

*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025.

Editor, Marlin Ouverson
Assistant Editor, Julie Anton
Contributing Editors, Dave Caulkins and Dave Cortesi
Marketing Director, Craig S. Harper
Advertising Sales, Laird Foshay
Circulation Assistants, Susan Hinton and Alice Hinton
Art Director, Clifford West
Artist, Barbara Ruzgerian
Typesetter, Renny Wiggins
Proofreader, Nancy Heubach

# EDITORIAL

No, this month's feature article is not about speech problems on Broadway. When the stage was first set for LISP to run on micros, it was inevitable that artificial intelligence would soon make an entrance. In his article, author Nicol explains how an actor, or object-oriented, language works, and how a program written in it can "learn."

As memory capacity grows, so will AI concerns in microcomputing. Some applaud this, while others sit in the back row and hiss. We hope both factions will offer their critical input and software for publication in this journal. We most humbly believe that AI would benefit from a little exposure to the good Doctor's community of friends.

Some readers have accused our overseas distributors of being artificially intelligent – and minimally so, at that. We have had problems in the past, which have mostly been resolved. Many times it is the vagaries of international postal systems that are to blame, not the distributors. And if they sell their entire stock – well, readers everywhere must deal with that, unless they choose to subscribe.

So, if you live outside the United States and decide it would be nice to receive your very own copy of *DDJ* each month, here are some sure-fire tips to help you place a trouble-free subscription:

1) Your **payment** to "Dr. Dobb's Journal" must accompany your order. An international postal money order will suffice; if you pay by check, it must be drawn on a U.S. bank and payable in U.S. funds. Both must be for the exact amount of your subscription (see page 46 for rates).
2) **Print** your name and full address (including country) carefully.
3) We highly recommend **airmail**. Boats are slow and sometimes sink – or at least the mail bags seem to do so.
4) If you wish to order any **back issues**, do so separately and pay for them separately.

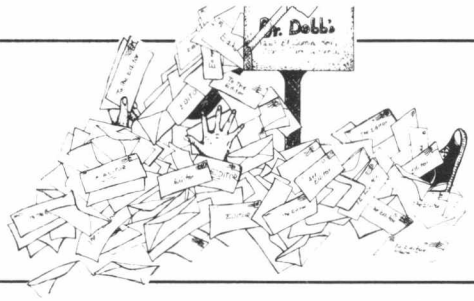
Following the above advice will prevent most common errors from happening. If you do experience any trouble with your subscription, allow extra time for international delivery. If that doesn't work and you need to write us a letter (please, let it be in the King's good English!), describe your experience step-by-step and include a photocopy of your original check (both sides of it) or money order receipt.

We hope you will subscribe and become a long-time friend of *Dr. Dobb's Journal*. Best wishes to you all. May the coming year find your bootleg copies of useful *DDJ* articles supplanted by shiny new issues of your own!

Marlin Ouverson

Marlin Ouverson
Editor

# LETTERS




## Forthright . . .

Dear *DDJ*,

Although I appreciate the beauty of B. Ragsdale's assembler for the 6502 (*DDJ* #59, pp. 12-24) in terms of the mechanism of code generation for the different addressing modes – there is a major flaw in it.

There is no justification whatsoever to make the words BEGIN, UNTIL, IF, THEN, ELSE, CODE END-CODE immediate words except to prevent sophisticated macro definitions. If those words weren't immediate, one could define, e.g.,

```
: 16INC,   ( ADR – )
  DUP INC,0= IF, ROT 1+ INC,THEN, ;
```

which, at execution time, will generate a macro expansion for incrementing a 16-bit value. It's usage would be somewhat like

```
CODE XXXX
    . . . COUNTER 16INC, . . .
  END-CODE
```

supposing COUNTER is a two-byte field.

Another – debatable – point is compiler security with the conditional constructs. I think that compiler security doesn't make very much sense in assembler coding – there are far too many pitfalls besides conditionals in any case. On the other hand, the security makes writing of unstructured code unnecessarily complicated.

I want to thank the editors for dedicating an entire issue to FORTH and I hope that it will result in discussions and suggestions which will enhance the quality of our systems.

Klaus Schleisiek
c/o J. Buettner
Eppendorfer Landstr. 16
D-2000 Hamburg 20
W-Germany

## ZSID Bug and (Risky?) Patch

Dear Mr. Marlin Ouverson!

Instead of complaining about the quality (in all, good) and the taste (it depends) I send you my renewal for two years (I hope that I will get an extra issue for prompt renewal, too, even if it is to overseas). And I send you a small article (perhaps better a bug-note) as I know that the journal depends on input! (This is my fifth contribution to *Dr. Dobb's.*)

It is about a CP/M program made by Digital Research™. I think it will be useful to some of the readers. By the way, my article of September, 1979 brought four intense contacts since then. Perhaps you may again publish my full address with the statement that I still seek contacts to European CP/M users.

Best wishes for you and the whole crew.

There is a small hidden bug in ZSID version 1.4. It is easily demonstrated when trying the following: in the direct assembly mode, try to assemble three 16-bit loads with the values of (in hex):

FF7F, FF80, FFFF.

Afterwards, list the product!

The false values are not only used in the assembler, but in the F (ill), M (ove) and D (ump) commands as well. The cure is very simple: change the byte at 02AE (hex) in ZSID from "C2" to "C3".

SID version 1.4 has not this "feature," so I suspect that the routine starting at 02A1H is, at some place, necessary in the Z80-specific parts, e.g., for error-checking in relative jumps??

I did not encounter any problems with the changed version of ZSID (see Table 1), but I am sure that this change is not without danger. . . . Best wishes to all CP/M users!

Holger Petersen
Samwerstr.21
2300 KIEL 1
West Germany

**Table 1. ZSID Changes**

```
A>DIR
A: ZSID      COM
A>ZSID
ZSID VERS 1.4
#A1ØØ
Ø1ØØ  LD HL,ØFF7F
Ø1Ø3  LD HL,ØFF8Ø
Ø1Ø6  LD HL,ØFFFF
Ø1Ø9
#L1ØØ,1Ø8
  Ø1ØØ  LD   HL,FF7F
  Ø1Ø3  LD   HL,ØØ8Ø
  Ø1Ø6  LD   HL,ØØFF
  Ø1Ø9
#IZSID.COM
#R
NEXT  PC  END
29ØØ Ø1ØØ B9FF
#L2A1,2BC
  Ø2A1  PUSH DE
  Ø2A2  CP   41
  Ø2A4  JP   NC,ØBCF
  Ø2A7  CALL ØEAA
  Ø2AA  LD   B,D
  Ø2AB  LD   A,B
  Ø2AC  CP   FF
  Ø2AE  JP   NZ,ØØB8
  Ø2B1  LD   A,E
  Ø2B2  OR   A
  Ø2B3  JP   P,ØØB8
  Ø2B6  LD   B,ØØ
  Ø2B8  LD   A,E
  Ø2B9  POP  DE
  Ø2BA  DEC  B
  Ø2BB  INC  B
  Ø2BC  RET
  Ø2BD
#S2AE
Ø2AE C2 C3
Ø2AF B8 .
#L2ADDE,2BØ
  Ø2AE  JP   ØØB8
  Ø2B1
#GØ

A>SAVE 4Ø ZSIDP.COM
A>
A>DIR
A: ZSID   COM : ZSIDP    COM
A>ZSIDP
ZSID VERS 1.4
#A1ØØ
Ø1ØØ  LD HL,ØFF7F
Ø1Ø3  LD HL,ØFF8Ø
Ø1Ø6  LD HL,ØFFFF
Ø1Ø9  .
#L1ØØ,1Ø8
  Ø1ØØ  LD   HL,FF7F
  Ø1Ø3  LD   HL,FF8Ø
  Ø1Ø6  LD   HL,FFFF
  Ø1Ø9
#GØ

A>
```

## Pet-urbed

Dear Mr. Ouverson:

I thought your readers might be interested in this *before they contemplate buying a 16K Commodore PET computer.* Commodore has, for about a year, [been]

*(Continued on page 19)*

by D. E. Cortesi, Resident Intern

## Octametry

In this column's first appearance we included an approximation to the hypotenuse that was found purely by chance, from no particular theoretical basis. That little item generated an amazing amount of interest. Now we've received the following dissertation from C. Roger Glassey of the University of California.

"There is more to trigonometry than the hypotenuse of the right triangle. One quick approximation to the hypotenuse, discussed in *DDJ* #56, takes the form:

(1) h(x,y) = A*x + B*y

"The geometry behind this approximation can be used to derive consistent approximations to the sine, cosine, and tangent functions. These may be useful in, for example, computing the screen positions of points of a rotating object, or in other applications that require fast approximate conversion between rectangular and polar coordinates.

"The formula used to compute the hypotenuse determines how distances will be measured. When Euclid's hypotenuse, SQRT(SQR(x)+SQR(y)), is used, the locus of points at a distance of 1 unit from the origin form the unit circle. But when equation (1) is used, points on the octagon shown on the cover of the July *DDJ* are 1 unit from the origin. In order to simplify the discussion, I assume that x and y are the coordinates of a point in the first octant of the plane, so that $0 \leqslant y \leqslant x$. Furthermore, when converting from rectangular to polar, it seems desirable that the point 1,0 (that is, the unit vector along the x-axis) have unit length. In other words, h(1,0)=1. Therefore, I will take A in equation (1) to be 1.

"In Euclid's plane, the length of the arc of the unit circle subtended by an angle is the measure of that angle. In the approximate plane, we use the unit octagon, shown in Figure 1 by the dashed lines. The point (x,y) is on this unit octagon and therefore, by definition, h(x,y)= 1. The length of the line from the point (1,0) to the point (x,y) is z, the measure of the angle at (0,0). If we let z increase, then we see (from the law of similar triangles) that y=bz and 1-x=az where a and b are constants of proportionality.

"Therefore the octagonal sine, cosine and tangent functions are:

(2) s(z) = y/h(x,y) = b*z

(3) c(z) = x/h(x,y) = 1-a*z

(4) t(z) = s(z)/c(z) = b*z/(1-a*z)

Letting y/x = r, the octagonal arctangent function is

(5) at(r) = r/(b+ar).

"We have only one degree of freedom in choosing the constants a and b because, since they are the proportionality constants for the sides of a right triangle in Euclid's plane,

(6) a*a + b*b = 1.

The use of this single degree of freedom will determine the error of these approximations.

"The octagonal trig functions can be used to calculate the hypotenuse as in ordinary trigonometry. Given x and y, use equation (5) to obtain z. Then use h(x,y) = y/s(z) or h(x,y) = x/c(z). Either way, the result is h(x,y) = x+y*a/b. Therefore, the constant B in equation (1) is a/b, and we have a set of approximations to the trigonometric functions that are consistent with the hypotenuse approximation.

"Rather than confronting the issue of a suitable value for B, consider what happens when z increases until it subtends an angle of 45 degrees, or 1/8 of the octagon. Let k be this value of z. A half circumference of the unit octagon is then 4*k, which corresponds to $\pi$ in Euclid's world, so 4*k is the octagonal approximation to $\pi$. Thus if you use equation (5) to compute the arctangent in degrees, remember that 90 degrees is 2*k, not $\pi/2$. I regard this constant k, the length of one side of the unit octagon, as the fundamental constant in the approximation plane. From it all others can be derived.

"Imagine Figure 1 in which z=k, so that x=y. Since the tangent of k is 1, we have the condition

(7) a*k + b*k = 1.

A little algebra applied to equations (5) and (6) produces:

(8) b = (1 + SQRT(2*k*k-1))/2*k

(9) a = (1 - SQRT(2*k*k-1))/2*k

Since a and b must be positive and real, k must be chosen between SQRT(2)/2 and 1. If k is set to 0.7895, so a=0.3188 and b=0.9478, then the maximum relative absolute error in h(x,y) is about 5.5%. It occurs at z=0.336, which is about 20 degrees, and also at z=k or 45 degrees. The corresponding value of B in (1) is 0.3364.

"The definition of s(z) given in (2) is valid for $-k \leqslant z \leqslant k$. For $z<0$, use the fact that c(z)=c(-z). Outside this range, imagine you are doing common trigonometry with tables that only cover the range 0 to 45 degrees. For example, if $k<z<2k$, use s(z)=c(2k-z). For angles representing more than one complete rotation, remember that 8k may be subtracted from z without changing the value of any of the octagonal trig functions.

"Happy approximations,
– C. Roger Glassey."

## And Furthermore . . .

While browsing through back issues of that excellent journal, *Software, Practice and Experience*, we found a related item. In a short article in volume 7, number 2 (March-April, 1977), p. 277, D. G. Colquhoun describes "A Fast Approximation to the SIN Function."

Colquhoun's method is well-suited to machines that store floating point numbers with an exponent that is a power of two stored as a binary integer. Two functions were offered, both requiring nothing but floating-point addition and integer operations on the exponent field of the number.

The less accurate approximation is

For $0 \leqslant x \leqslant 0.829347$,

$\sin(x) = \frac{15}{16} * x$

For $0.829347 < x \leqslant \pi/2$,

$\sin(x) = \frac{5}{16} * x + 0.518342$

A circle plotted using this approximation comes out as an octagon! Colquhoun notes that fifteen-sixteenths of x is easily gotten as x-x/16, while five-sixteenths x is just (x/4)+(x/16).

A more accurate approximation comes from a four-stage approximation:

For $x \leqslant 0.426350$,

$\sin(x) = x$

For $0.426350 < x \leqslant 0.968634$,

$\sin(x) = \frac{3}{4} * x + 0.106587$

For $0.968634 < x \leqslant 1.413800$,

$\sin(x) = \frac{3}{8} * x + 0.469825$

For $1.413800 < x \leq \pi/2$,

$\sin(x) = 1$.

A circle plotted with this function produces a 16-agon. Three-fourths and three-eighths are again easily gotten by modifying the binary exponent of a float number.

## Touching on Tones

In issue #58 (August 1981) we published a request from J. R. Snow of San Francisco, CA. He was looking for a cheap way to decode standard touch-tone signals. Snow's ultimate objective is the ability to control a computer remotely, by calling it up and commanding it through the touch-tone pad.

Two readers responded. Gene Head of Corvallis, OR, suggests that Snow should check the *Radio Amateur's Handbook* published by the Amateur Radio Relay League (ARRL), and also that the ads in *QST*, the magazine of the ARRL, should yield some suppliers.

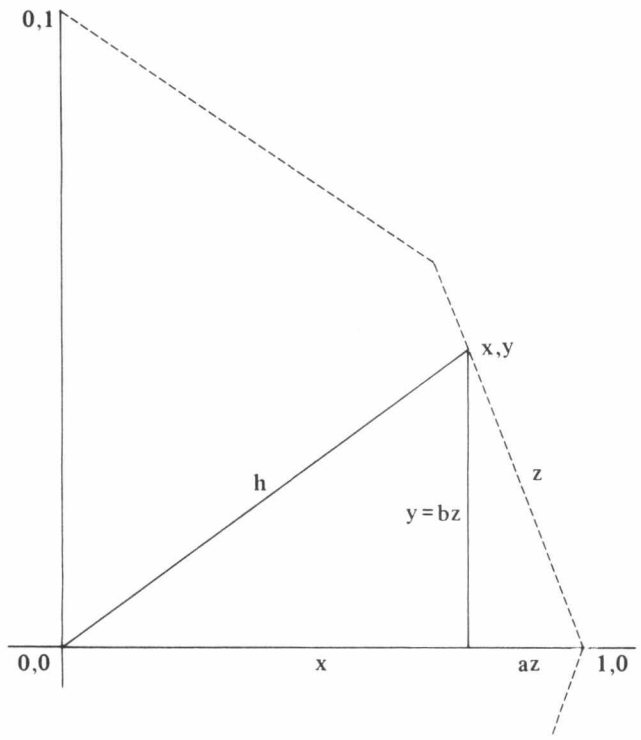
David L. Hoats of Bordentown, NJ, referred us to an article, "Build this Unicorn-1 Robot, Part 9" by J. A. Gupton, published in *Radio-Electronics* magazine in April, 1981. That article cites suppliers of a decoder kit costing $70. Hoats writes, "Having built the decoder, I can assure you that it is of highest quality. Construction is fairly simple, but calibration can be tedious without the aid of a frequency counter." He adds, "I would like to see an article on interfacing this sort of decoder to, say, a KIM-1, since (being only a feeble-minded hardware hacker) I'm not clever enough to work out the programming involved."

We can think of a lot of applications of Snow's idea. A very small machine (a KIM or AIM would be perfect) could be equipped to detect when the phone was ringing, to answer it, and to perform some action under touch-tone command. A slightly larger machine could, with the addition of a Votrax Type 'n' Talk, talk back to the caller. In fact, the world's smartest phone-answering machine could be built in this way, for under $1000.

A good, cheap, telephone interface design could almost be machine independent. The interface points would be the standard Bell wall-jack on one end, and a set of parallel latches on the other. An input bit for ring-detect, an output bit controlling an off-hook relay, four bits for the decoded touch-tone digit, and a digit-valid bit would be enough. Some means of switching the audio signal to a recording device would be nice. If anyone wants to design that circuit, *DDJ* would be delighted to publish it.

Suppose one had such an interface? Then what should the command language look like? The alphabet of the command is composed of the touch-tone digits. We are told that there are really 16 of them defined, but the standard phone has only 12 buttons (a portable, 16-key touch-tone pad would be another nice project). The specific commands would depend on the application, but we can see one problem that would be a constant. Namely, how can an ASCII literal string be represented, using a 12-digit alphabet? Bear in mind that a human interface is involved here; whatever is used should be simple and natural enough that a person in a phone booth on a wet and windy night could recall and use it without notes. Stare at your touch-tone phone for a while and send suggestions to the Clinic.

Figure 1.




**Engineers**
**Scientists**
**Systems Programmers**
**Business Owners**
All these read *Dr. Dobb's Journal*.
Reach them with cost-effective advertising.
(415) 323-3111.

# OF INTEREST

by Dave Cortesi

## An Extraordinary Offer

We are in receipt of a letter from **Bill Johnston** of Applied Computing, who writes as follows:

> A comple of years ago our company was involved in a commercial project to install automatic ribbon re-inkers on microcomputer printers. These modification kits are mounted inside the printers where they automatically re-ink the ribbon whenever the machine is in operation. The result is consistent crisp, dark print, along with extension of ribbon life by a factor of ten times or more . . . . We have perhaps a hundred of these kits left on hand. The original selling price was $150 per kit, which included installation. The remaining kits are being made available to interested computer users at $18.25 each, postpaid ($1 extra for air mail). The purchaser must do the installation, of course. The re-inkers will work on any printer with a one-half inch or narrower ribbon.

No technical information was included, so we can't comment on the skill needed for the installation. Readers who feel competent to modify their own printers, and who aren't worried about the possible effect on their printer's warranty, may want to follow up on this.

## An Extraordinary Board

In the September issue we dropped a couple of items to make room for an IBM announcement. One of the items had an accompanying picture, which didn't get dropped. Here's the item to go with the photo: "**Eventide Clockworks** have made the first announcement we've seen of a RAM board that uses the new 64K-bit chips. They have packed 512K bytes of RAM on a single board to fit the HP 9845 desktop computer. They note that one board has more capacity than both the machine's tape cartridges. A moment with our pocket calculator reveals that the board costs just over $12 per kilobyte, suggesting that a 64K board for an S-100 system could be brought in at well under $1000."

Publication of the photo without the text caused some confusion for one optimistic soul. Richard Factor, Eventide VP, writes that a reader called to ask if the board could possibly be used to expand a KIM! Well, why not, Richard? There are *lots* more KIMs than there are HP-9845s, after all. If the KIM market is too small, give some thought to how much you could put on a single S-100 board. Say, 256KB with error-correcting logic, for CP/M 86? Or the bank-switched storage for an 8-user MP/M system on one card? Either one ought to come in around $2500, we figure.

## A Significant Step

Tandy Corp. has been making a number of highly significant moves this fall. The first was to provide bisync communications support, and emulation of IBM and DEC bisync workstation and terminal devices ("Of Interest," September). The second move was the introduction of a communications multiplexer that allowed a TRS-80 Model II to supply local network services to 16 satellite VideoTex terminals ("Of Interest," October). A third was the recent announcement of a hard disk for the Model II.

Now Tandy have announced that they will support the protocols and hardware of ARCNET, a local network architecture devised by the Datapoint corporation. ARCNET seems to resemble Xerox's Ethernet in that it can connect large numbers of machines cheaply through coaxial cables. Unlike Ethernet, ARCNET has been in commercial use since 1977. A TRS-80 Model II is linked into an ARCNET by the addition of a single interface card and a length of RG-62 cable. The card will cost "about $400," which Tandy claim is "significantly" less than an Ethernet interface. Four Model II's can be interconnected through a $200 junction box; larger clusters will require an active junction costing "about $2000."

Tandy suggest that one machine could provide hard-disk file services to three others, the others being used simultaneously for word processing, data entry, or what-have-you. But of course, one node of the ARCNET could be a VideoTex server providing information retrieval for 16 users. Yet another could be a batch workstation connected via a bisync line to an IBM or DEC mainframe; it could keep its input and output job queues on the hard disk run by the file server. Or you could build a continent-spanning bisync network, each of whose nodes was a local ARCNET of up to 255 Model II's. In fact, we can't think of any university or corporate computing need that couldn't be met by some combination of central mainframe and such a two-level network.

Taken as a group, these moves give Tandy a technological base from which they could assault markets that have until now been the province of IBM, DEC, and Data General. If they can convert that technical lead into marketing results, they may simply run away from the rest of the micro makers.

## Z89 Times 6 = Z90

**Zenith** notified us that they are shipping double-density double-sided drives for the Z89. The present Z89 (and H89) drives hold 100KB. With the new Z-89-37 double-density disk controller card, a standard drive becomes soft-sectored and so can hold 160KB. The controller is priced at $395. Its real use is in driving the new Z-37, a box containing a pair of double-sided, quad-density, five-inch drives. Each drive handles 80 tracks per surface, or 160 tracks per drive, or 640KB of data per drive. The dual drive unit with its 1.3MB total capacity is priced at $2000.

An upgraded Z89, containing the new disk controller and 64KB of RAM standard, is now being shipped as the Z90 model. The release notes that the disk controller will fit an H89, but it doesn't give any indication of what these desirable enhancements might cost under the Heathkit name. Updates for both HDOS and the CP/M CBIOS are available.

## Bus to Bus

**D & W Digital** tell us of an S-100 board that sounds interesting. It's a bus driver for the IEEE-488 instrumentation bus. The centerpiece of the S-100 card is a TMS 9914, a chip that handles the 488 protocol in any of the Talker, Listener, or Bus Controller roles. Other board circuitry maps the chip to 8-bit I/O ports, and provides three parallel ports besides. The board is priced at $375. Software drivers written in MBASIC are provided.

## Lots of Dots

**Okidata**, hard on the heels of Epson, have come out with a dot-graphics option for their Microline 82, 83, and 84 printers. The feature consists of a pair of ROM chips which are "easily installed by the user," and are supposed to be available now at a price "under $100." With the feature installed, the Microline 82 and 83 printers can print any dot on the page with a resolution of 60 dots horizontal and 66 vertical. The Microline 84 can do 72 by 72 dots per inch. Software for

making a dot-for-dot copy of the screen of an Apple is included.

## Apple Upgrades

Here's another way to print the screen of an Apple. **Orange Micro** make a smart printer interface card for the Apple (apple? orange?). At heart it seems to be a Centronics-type parallel interface, but the card also contains on-card firmware for copying the high resolution graphics screen to any of several dot-graphics printers. The Grappler, as they call it, can print the screen as black dots for white spots, or vice versa, or double-sized, or rotated 90 degrees. It is also said to support margins, automatic formfeeds, and word-wrap. The only thing Orange Micro omitted to tell us about it was the price.

And a distributor called **Computer Data Services** tells us of some Apple improvements. One is a ROM card that supports two of either 2K ROMs, or 4K ROMs, or 2K RAM chips. That $125 item comes with a utility ROM that supports line renumbering and several other functions. Another is a 32KB RAM card for $239; it comes with software that "will automatically relocate DOS into one of the two 16K banks on the board." That, they say, allows one to have "Applesoft and Integer BASIC and DOS, and 48K to program with." We don't speak Apple, so this is not awfully meaningful to us – but it sounds good.

## Yet Another XDIR

There's a diskette of CP/M 2 utilities to be had for $35 from **Elliam Associates**. The featured program is – an extended DIR. This one displays filenames in alphabetical sequence, with their sizes. It takes advantage of CP/M 2 to be device independent, working with any disk type or density (about time we saw that done). Other programs in the package will convert a PRN file back to ASM source, make a symbol table for SID from an ASM listing, and so on. All reminiscent of programs from CPMUG, but if they have been properly upgraded to take advantage of CP/M 2 as claimed, they are worthwhile.

Speaking of CP/M software, our overseas readers might like to know about **TCL Software**, a British distributor that seems to aim at being the kind of software supermarket in the UK that Lifeboat is in the US. Their catalog lists a good selection of CP/M applications and utilities at prices comparable to those in the US, and they carry all the CP/M Users Group volumes.

## BASIC, BASIC, and C

Three notices of interest to language buffs came our way this month. An outfit called **BPS** (bits per second? Bob, Peter, and Sam?) has built a listing formatter for Microsoft BASIC. The $124 program has two functions: it converts a saved program into a titled, paginated listing, and it creates an alphabetized cross reference of program names to the lines on which they are used.

**Allen Ashley** has extended his COMSTAR compiler for North Star BASIC to provide program overlays. "An overlay differs from program chaining," he says, "in that a root program segment and selected program variables can survive intact as a new program segment is introduced." The compiler enhancement is $75 to present users.

And **JMI** has recreated a memory-resident executive for programs in C. That's run-time support for everything except file I/O. The executive supports multiple tasks with intertask communication and formatted I/O. It can drive multiple terminals. The executive is written "mostly" in C; it is available now for the DEC LIS-11 and the M68000, wih 8080 and Z80 versions in preparation. The price is $300.

People within reach of San Francisco who want to know more about C and Unix should check out a tutorial and con-





**Back Issues**

**Available for $3.50**

*IBM PC, it's still hot news*

**CP/M-86 vs. MSDOS: A Technical Comparison**
Our author sets the two operating systems for the IBM Personal Computer head-to-head, to see which comes out the winner!

**Word Processors for the IBM Personal Computer**
Reviewers sample some of the latest releases in this ever-popular application of computing power.

**Video Graphics for the IBM PC**
One of our popular writers delves into the details (some not clearly documented elsewhere) of high-resolution graphics. This advanced material is carefully introduced so less experienced readers won't be lost.

This and other back issues are available for $3.50 from:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**

ference to be held by **Walter Zintz' Uni-Ops**. There will be a three-day tutorial, January 11th to 13th 1982, followed by the group's "first international conference." No fees are mentioned in the release, but we wouldn't assume that it was free until we'd checked.

## Info for Entrepreneurs

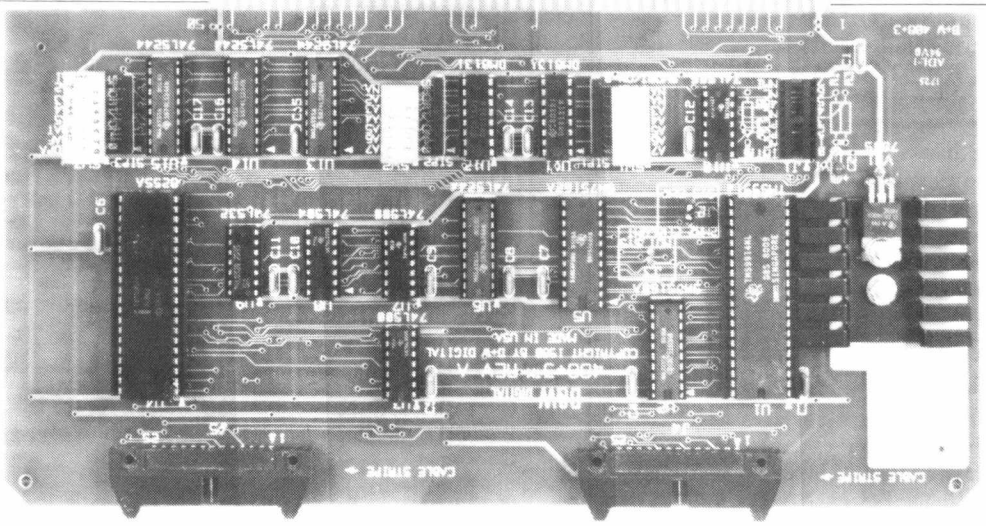
Those trying to make it as independent business people might like to know about some information sources we heard of. *Software Wanted: How and Where to Sell Your Programs* is the title of a report from **Battery Lane Publications**. They claim to have gathered information from software publishers and distributors on the kinds of programs they want, the royalties they pay, and what assistance they offer the author. The $25 report covers "over 60" companies and is sold with a money-back guarantee.

Hewlett-Packard, like TI and Atari, has realized the value of third-party software. Now HP has launched a program to sell third-party software for its line of programmable calculators. Programs will be listed in a single catalog with HP-written programs; the catalog will go to all HP dealers. Authors who qualify for the program will be able to buy HP equipment at a "significant" discount. The release didn't give a mailing address; interested parties should call **Jack Peters** direct.

If you want to sell to computer dealers, and if you have an Apple, we are about to make your day. Learn about **Computer Information Systems**, that have prepared two mailing lists on Apple II diskettes. The first list contains "over 800" authorized Apple dealers and costs $150. The second list is a superset; it has the Apple dealers and some 800 more microcomputer retail outlets "but no Radio Shack stores." That costs $250. For a single mailing you may order the lists printed on gummed labels for half the above prices, but that would clearly be false economy.

If you are a consultant or a buyer who finds it hard to keep up with the software market, you may get some relief from the services of **Transdata Network Telecommunications**. They claim to have, on-line, "The equivalent of 50,000 pages of literature on hardware and software." Their release confused us a bit, but it seems you can access this data in one of two ways. You can establish a credit line and an equipment profile with them and then ask them questions by phone. Or you can rent microfiche copies of literature – the manuals for various software products, for instance – to examine for yourself. Each phone query is $3; 'fiche rental is around $10.

**DDJ**



D & W Digital bring the IEEE-488 bus to S-100 machines.

## Contact Points

**Allen Ashley**, 395 Sierra Madre Villa, Pasadena, CA 91107; (213) 793-5748.

**Battery Lane Publications**, POB 30214, Bethesda, MD 20014; (301) 770-2726.

**BPS**, 82 Woods End Road, Fairfield, CT 06430; (203) 254-1659.

**Computer Data Services**, POB 696, Amherst, NH 03031; (603) 673-7375.

**Computer Information Systems**, 218 Huntington Road, Bridgeport, CT 06608; (203) 579-0472.

**D & W Digital**, 1524 Redwood Drive, Los Altos, CA 94022; (415) 966-1460.

**Elliam Associates**, 24000 Bessemer Street, Woodland Hills, CA 91367; (213) 348-4278.

**Eventide Clockworks**, 265 West 54th Street, New York, NY 10019; (212) 581-9290.

**JMI** Software Consultants, 3 Neshaminy Interplex, Trevose, PA 19047; (215) 638-1112.

**Bill Johnston**, c/o Applied Computing, 1808 Pomona Drive, Las Cruces, NM 88001.

**Okidata**, 111 Gaither Drive, Mt. Laurel, NJ 08054; (609) 235-2600.

**Orange Micro**, 3150 E. La Palma, Suite I, Anaheim, CA 92806; (714) 630-3622.

**Jack Peters**, Hewlett-Packard Corvallis (OR) Division; (503) 757-2000, extension 2207.

**TCL** (Transam Components Limited) Software, 59/61 Theobald's Road, London WC1; 01-405-5240/2113 (England).

**Transdata Network Telecommunications**, 1530 East Shaw, Suite 117, Fresno, CA 93714; (209) 227-0477.

**Walter Zintz' Uni-Ops**, POB 5182, Walnut Creek, CA 94596; (415) 933-8564 mornings.

**Zenith** Data Systems, 100 Milwaukee Avenue, Glenview, IL 60025; (312) 391-8181.

Since 1976, the most innovative magazine for the technically-sophisticated microcomputer user.

# Dr. Dobb's Journal

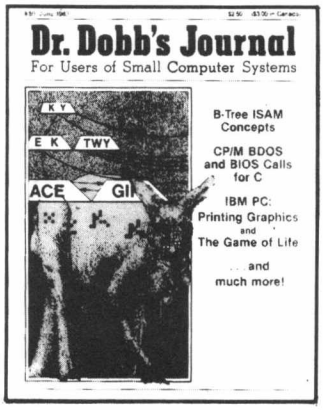
## For Users of Small Computer Systems




1976



1983




Have you been looking for a magazine that will provide you with serious articles on programming? *Dr. Dobb's Journal* can give you the technical depth you need to understand the latest software advances. We cover every microprocessor with articles on C, Forth, Unix, CP/M and much more. Our reputation for editorial quality is well known. Subscribe now, and discover what industry insiders have enjoyed since 1976.

Coming articles will cover:

- new language development
- 8- and 16-bit technology
- operating systems
- compilers, editors
- assemblers/cross-assemblers
- clever algorithms

**$25**

one year subscription
12 issues
saves $5 off the newsstand price

**YES!** Enter my subscription to *Dr. Dobb's Journal* and bill me later.

name

address

city state zip

**Mail to: DDJ, P.O. Box E., Menlo Park, CA 94025**

# Actors, Property Lists, and

## AI Language Finds Micros

by Ronald L. Nicol

Complexity is the superposition of simplicity; it is the reduction of this complexity to its basis in simplicity that is the goal of Artificial Intelligence (AI). Even though the field of AI is a tree of many branches, virtually all in the field speak LISP, a list processing language. LISP allows fluent use of property lists, a data structure for storing information. Actor languages, also known as object-oriented languages, are eloquently written in LISP.

Actor languages are languages in which objects communicate by the sending and receiving of messages. Each actor is a member of a class, and that class may be a member of a superclass, and so on. If an object receives a message to which it does not know how to respond, this message is passed on to that actor's class, and so on, until an actor is called which can respond.

LISP was developed in the early 1960's by John McCarthy as a recursive, list-processing language. The language has two types of objects, atoms and lists. An atom is a sequence of alphanumeric characters, each of which may be considered a type of actor in the language. Lists are designated by left and right parentheses. A left parenthesis indicates the beginning of a list and a right parenthesis signifies the termination of the list. Lists may be nested to any level of complexity, i.e., there may be lists of lists. The following is a list of three atoms:

```
(IN THE BEGINNING)
```

while the following is a list of three lists:

```
((IN)(THE)(BEGINNING))
```

Property lists are associated with atoms, and each atom has a property lists which allows that particular atom to remember attributes which that atom may have. Properties are accessible through the use of an indicator which itself is an atom. The properties themselves may be either an atom or a list structure. There are two functions in LISP which allow the manipulation of property lists, PUTPROP and GET.

PUTPROP provides the means of placing properties on atoms under a specified indicator. PUTPROP takes three arguments: the atom receiving the property, the property itself, and the indicator under which the property will be stored. The syntax of the PUTPROP function is:

```
(PUTPROP atom property indicator)
```

An example of the use of this function follows:

```
(PUTPROP(QUOTE APPLE)
   (QUOTE(FORBIDDEN FRUIT))
   (QUOTE DESCRIPTION))
```

The example also makes use of another LISP function, QUOTE. QUOTE simply means that what follows the word QUOTE is to be taken literally; that is, APPLE is considered as the word "apple" not as some function named APPLE. The result of executing the above example is to put, on the property list of the atom APPLE under the indicator DESCRIPTION, the list (FORBIDDEN FRUIT).

GET provides a means of retrieving the property once it is placed on the atom. GET is a function of two arguments, the first of which is the atom to be investigated. The second is the indicator under which the property is stored. The general form of the function GET is:

```
(GET atom indicator)
```

An example of the use of this function follows:

```
(GET(QUOTE APPLE)(QUOTE DESCRIPTION))
```

Note here that QUOTE is used to take the atoms APPLE and DESCRIPTION literally. The result of the example will be the list (FORBIDDEN FRUIT). If no property is saved under the requested indicator, GET returns the atom NIL.

At this point it is appropriate to discuss the form a program or function takes in LISP. By necessity this discussion will contain only a subset of available LISP functions. The user builds his own LISP functions by using the functions available in the basic LISP system and previously defined user functions. At the heart of the user-defined function is the atom LAMBDA. The atom LAMBDA was chosen as the functional specifier in LISP in tribute to the lambda calculus of Alonzo Church. The general form of a user-defined function in LISP is:

```
(LAMBDA(variables)(text of function))
```

where the variables specified in the list following the atom LAMBDA are variables which are local to this function. The text of the function contains previously defined user LAMBDA functions or system functions. A simple function follows:

```
(LAMBDA(X)(CAR X))
```

Again a new LISP function, CAR, is introduced. CAR returns the first element of the list which is its argument. The above LAMBDA function will return the first element of the list it is given as an argument.

Now that the structure of a LISP function has been exposed, it is necessary to have a way in which to assign a name to a particular user-defined function. This is accomplished through the user of property lists. The chosen atom name for the function has the LAMBDA expression of the function placed on its property list under the indicator EXPR. The LISP system knows that when it finds a list under the indicator EXPR on an atom that this is a user-defined function. For example, let us define the function FIRST which returns the first element of a list.

```
(PUTPROP(QUOTE FIRST)
   (QUOTE(LAMBDA(X)(CAR X)))
   (QUOTE EXPR))
```

Now the function FIRST has meaning. Tracing the following entry reveals the inner workings of LISP:

```
(FIRST(QUOTE(CHICKEN OR THE EGG)))
```

The LISP system first encounters the atom FIRST and since it is not quoted it is assumed to be a function. The system scans the property list of FIRST and finds the function, (LAMBDA (X)(CAR X)). It then assigns the argument to FIRST, which in this case is the list (CHICKEN OR THE EGG), the local variable X. The function then takes the CAR of the argument and returns that value which, in this case, is the atom CHICKEN.

Another important system function in LISP is CDR. CDR takes one argument which is a list and returns a list which is the original argument with the first element removed. At this point you should be able to define a function called REST which returns a list with its first element missing. Other functions are defined using combinations of CAR and CDR. For example, CADR is the CAR of the CDR of a list, and CDDR is the CDR of the CDR of a list.

*Ronald L. Nicol, LT., USN, 3102 Joyce Drive, Orlando, FL 32812.*

An important construct in LISP is the function COND. This is the conditional statement in LISP and is similar to BASIC's IF . . . THEN . . . ELSE. The general form of COND is:

```
(COND(p1 e1)(p2 e2) . . . (pn en))
```

where the p's are predicate functions, i.e., functions which return a true or false value and the e's are expressions. When the COND function is executed, the predicates are tested in ascending order and when a particular predicate results in other than a false value, the value of the COND is the corresponding expression. For instance, if p1 is false but p2 is not false, the value of the COND will be e2. In LISP the values true and false are represented by the atoms T and NIL respectively. By way of example:

```
(COND((QUOTE NIL)(QUOTE(FIRST CLAUSE)))
   ((QUOTE T)(QUOTE(SECOND CLAUSE))))
```

is equal to the list (SECOND CLAUSE) because NIL is the false value of LISP and T, of course, is the true value. In LISP, certain atoms need not be quoted, among these are: T, NIL, EXPR, LAMBDA, and a few others. In this light, the above COND may be written as:

```
(COND(NIL(QUOTE(FIRST CALUSE)))
   (T(QUOTE(SECOND CLAUSE))))
```

Some of the other LISP functions which will be used throughout this article are presented in Table 1.

## And All the World is a Stage

The actor language which will be developed is loosely based on a subset of the language discussed by Kenneth M. Kahn[1]. It is limited but is easily extensible, and for the sake of clarity, the graphics sections of the language will not be discussed.

As in most actor languages, the concept of class will become very important. Only one actor will not have a class and that is the actor SOMETHING which is at the top of the hierarchy. In this particular language, each actor may be a member of one class and of course that class may be a member of another class, etc. . . . .

As a precursor to defining the first actor, it is necessary to extend the LISP GET function. GET obtains properties only from the atom which it is given. In the actor environment, if a particular actor does not have a given property, the class of which it is a member must be interrogated. If that class has no property, the property list of the class of which *it* is a member must be checked, and so on. A function which will meet these specifications is listed below:

```
(PUTPROP(QUOTE GETANS)
   (QUOTE(LAMBDA(X Y)
      (COND((GET X Y)(GET X Y))
         ((GET X(QUOTE CLASS))(GETANS
(GET X(QUOTE CLASS))Y))
         (T(LIST X
            (QUOTE HAS)
            (QUOTE NO)
            Y)))))
   EXPR)
```

This function, GETANS, is a recursive function, i.e., it is a function which is contained in its own definition. This is one of the most powerful features of LISP, and a most interesting *forte*.

The above PUTPROP defines the function GETANS. It requires two arguments, an atom and an indicator. It is a COND which has three clauses, each of which responds to certain conditions. The first clause tests to see if there is a property on the atom given as the first argument under the indicator given as the second argument, and if so this property is returned. The second clause calls GETANS with the same indicator but with the class of which the first argument is a member. If this recursion does not result in the function GETANS finding the indicator on any of the ancestors of the first argument, the third clause will cause the printing of a message which informs the user that the property does not exist in the bloodline of the actor.

## Something Exists

It is now time to define the first actor which will subsequently be used to define all other actors. The first actor is SOMETHING, which as previoulsy mentioned is at the highest level. It must have the capabilities to create and destroy actors, to force actors to remember and to forget, and it must be able to interrogate other actors.

So that SOMETHING can learn other traits, it must first be taught to remember. SOMETHING remembers by:

```
(PUTPROP(QUOTE SOMETHING)
   (QUOTE(LAMBDA(MESSAGE)
      (PUTPROP(CAR MESSAGE)
         (CADR(CDDR MESSAGE))
         (CAR(CDDR MESSAGE)))))
   (QUOTE REMEMBER))
```

After the above is entered, SOMETHING has the ability to remember whatever the user desires. To make SOMETHING remember, the input is:

```
(SOMETHING REMEMBER DAY MONDAY)
```

Before SOMETHING will REMEMBER, however, it is necessary to write a small LISP function which will allow SOMETHING to receive the proper orders. The function TRANSLATE will be defined as given in Listing 1. Its sole purpose is to translate input supplied by the user to information which the collection of actors can use. Note that once TRANSLATE is called, it continues to loop and await input for the actors; therefore it is only necessary to call TRANSLATE once at the beginning of the conversation. TRANSLATE uses the PROG feature of LISP which is a mechanism for iterative programming so familiar in the world of BASIC. The general format of a PROG is:

```
(PROG(prog variables)(s1)(s2)(s3) . . . (sn))
```

where the prog variables are variables which are local to this particular PROG, and s1 through sn are LISP statements which will be executed sequentially unless a GO statement is encountered. The GO statement is similar to the GOTO statement in BASIC and is directed to an atomic label which is placed before a statement.

Thus far a conversation with SOMETHING would consist of entering:

```
(TRANSLATE)
(SOMETHING REMEMBER DAY MONDAY)
```

This allows SOMETHING to remember that the property stored under the indicator DAY is MONDAY.

Since SOMETHING can now remember, it is useful to tell SOMETHING how to make new actors. This is accomplished

by having SOMETHING remember the function MAKE:

```
(SOMETHING REMEMBER MAKE (LAMBDA
(MESSAGE)
                (PUTPROP(CAR(CDDR MESSAGE))
                  (CAR MESSAGE)
                  (QUOTE CLASS))))
```

The function MAKE operates by placing on the new actor the class of which it is a member, under the indicator CLASS. Therefore, to create an actor named OBJECT the following would be required:

```
(SOMETHING MAKE OBJECT)
```

This creates the actor OBJECT which is a member of the class SOMETHING.

Now that actors can remember and create other actors, it is desirable to ask actors what they know. This is done by teaching SOMETHING the function WHAT.

```
(SOMETHING REMEMBER WHAT (LAMBDA
(MESSAGE)
                (PRINT(GETANS(CAR MESSAGE)
                  (CAR(CDDR MESSAGE))))))
```

Actors are now able to answer the question "what" as illustrated by:

```
(SOMETHING WHAT DAY)
```

which solicits the response MONDAY. Notice also that

```
(OBJECT WHAT DAY)
```

also equals MONDAY because the function GETANS is used in WHAT, and OBJECT is of the class SOMETHING.

Occasionally it is necessary to have an actor forget some information. This is accomplished through the use of FORGET which is created as follows:

```
(SOMETHING REMEMBER FORGET (LAMBDA
(MESSAGE)
                (REMPROP(CAR MESSAGE)
                  (CAR(CDDR MESSAGE)))))
```

Now an actor can be made to forget anything it knows. An exemplary message might be:

```
(SOMETHING FORGET DAY)
```

Now SOMETHING will not know that the day is Monday.

Just as an actor might need to forget something, so also is it sometimes necessary to destroy an actor. Eradication of an actor is possible with the use of UNMAKE which is brought into being as follows:

```
(SOMETHING REMEMBER UNMAKE (LAMBDA
(MESSAGE)
                (RPLACD(CAR(CDDR MESSAGE))
                  NIL)))
```

This has the effect of completely removing all properties from the actor. This is true because the property list of an atom exists as its CDR and replacing the CDR of the atom with NIL removes all properties. Sample input from the UNMAKE function might be:

```
(SOMETHING UNMAKE OBJECT)
```

which results in the actor OBJECT being forgotten by the system.

## The Amoeba and Man

One method of simplifying a complex system is to sever it into smaller components. An illustration of this method and of the newly defined actor language will be presented in the field of Biology. Since the middle of the eighteenth century when Carolus Linnaeus, the Swedish botanist, wrote *Systema Naturae*, man has been attempting to place all living things into a classification scheme. The hierarchy of living things can be considered as a living tree with each of the branches sharing certain properties of the limbs from which it sprouts.

Of course, it would be absurd to attempt to represent all phyla and species of organisms in a simple program, so for that reason only a small subset will be considered. It is hoped that enough information will be available to give a taste of the method and a flavor for why this particular type of computer language may open previously closed doors in information theory.

The technique of generating the actors which compose this tree of life is represented in Listing 2. A survey of this listing will indicate the conversational flow of information and is not unlike the teaching of a child. Once all descendents of ORGANISM have been told what they need to know, we can interrogate verious organisms and the uninformed can actually learn significantly from the actors present.

One might ask:

```
(AMOEBA WHAT LOCOMOTION)
```

which will promptly be answered by:

```
(MOVE BY MEANS OF PSEUDOPODS)
```

The question then arises, do just amoeba move in this manner or are they a member of a larger class. Therefore one might ask:

```
(AMOEBA WHAT CLASS)
```

which solicits the response

```
(SARCODINA)
```

Now, the inquisitive user asks, do all sarcodina also move in the way that the amoeba moves:

```
(SARCODINA WHAT LOCOMOTION)
```

Which produces:

```
(MOVE BY MEANS OF PSEUDOPODS AND A FEW
BY FLAGELLA)
```

So it is now clear that not all sarcodina move by pseudopods. To be even more general, the class of the sarcodina can be determined

```
(SARCODINA WHAT CLASS)
```

and the answer PLASMODROMA will be given. The following dialogue with the actors might ensue:

```
User: (PLASMADROMA WHAT LOCOMOTION)
Computer: (POSSESS FLAGELLA AT SOME STAGE
  OF LIFE CYCLE)
User: (PLASMODROMA WHAT CLASS)
Computer: PROTOZOA
User: (PROTOZOA WHAT CLASS)
Computer: ANIMAL
User: (ANIMAL WHAT CLASS)
Computer: ORGANISM
User: (ORGANISM WHAT CLASS)
Computer: SOMETHING
```

Starting with man, *homo-sapien*, the succeeding conversation might be exemplary:

```
User: (HOMO-SAPIEN WHAT DIGESTIVE-SYSTEM)
Computer: (TUBULAR: TWO OPENINGS)
User: (HOMO-SAPIEN WHAT CELL-STRUCTURE)
Computer: (CELL MEMBRANE)
User: (HOMO-SAPIEN WHAT EMBRYONIC-CELL-
  LAYERS)
Computer: THREE
```

```
ALIST:   (1). Number of arguments:  none.
         (2). Type of arguments:  not applicable.
         (3). Value:  List of current system variables.

APPLY:   (1). Number of arguments:  3, arguments are evaluated.
         (2). Type of arguments:
              (a). Argument 1 must evaluate to a function.
              (b). Argument 2 is a list of arguments for
                   the function in argument 1.
              (c). Argument 3 is the ALIST which will specify
                   the environment in which the application of
                   the function will occur.
         (3). Value:  Result of applying the specified function
                      to the specified arguments in the
                      specified environment.

LIST:   (1). Number of arguments:  indefinite, arguments are
                                   evaluated.
        (2). Type of arguments:  lists or atoms.
        (3). Value:  list of evaluated arguments.

REMPROP:   (1). Number of arguments:  2, arguments are
                                      evaluated.
           (2). Type of arguments: atoms.
           (3). Value:  first argument, effect is to remove the
                        property which is the second argument
                        from the first argument.

RPLACD:   (1). Number of arguments:  2, arguments are evaluated.
          (2). Type of arguments:  first and second argument is
                                   an atom or a list.
          (3).  Value:  first argument with its CDR replaced by
                        the second argument.

SETQ:   (1). Number of arguments: 2, the first argument is not
                                  evaluated but the second
                                  argument is evaluated.
      (2). Type of arguments:  first argument must be an atom
                               but the second argument can be an
                               atom or list.
         (3). Value:  the second argument, the effect is to set
                      the atom which is the first argument to
                      the second argument.
```

Figure I. Additional LISP functions used by the actors which are not described elsewhere in the text.

User: (HOMO-SAPIEN WHAT CLASS)
Computer: PRIMATE
User: (PRIMATES WHAT CLASS)
Computer: MAMMALIA
User: (MAMMALIA WHAT SYMMETRY)
Computer: BILATERAL
User: (MAMMALIA WHAT LOCOMOTION)
Computer: (AT LEAST ONE MOBILE PHASE IN DEVELOPMENT)
User: (MAMMALIA WHAT CLASS)
Computer: VERTEBRATA
User: (VERTEBRATA WHAT CLASS)
Computer: CHORDATA
User: (CHORDATA WHAT METABOLISM)
Computer: (ALL DERIVE FOOD FROM OTHER ORGANISMS)
User: (CHORDATA WHAT CLASS)
Computer: METAZOA

Of course, the above cast of actors can be expanded and such a system would lead to a much more complete and meaningful collection of facts. Even with the abbreviated version given, it is apparent that interaction with the various actors can lead to a rewarding learning experience. Through the queries directed to the actors, an astute student would be able to concentrate on the similarities and differences of the various organisms and it would certainly be possible to develop other actors which could ask questions which might generate parallels not previously drawn between varied organisms.

## Curtain Call

The actor language discusssed above is not very powerful by virtue of its simplicity. It can be expanded upon greatly by the use of a pattern-matching function which allows the addition of the IF . . . RECEIVE message and a syntax which is of a more general form. Additionally, graphic functions can be added so that actors can SHOW, APPEAR, ROTATE and perform other motions on the display device. An excellent LISP pattern matcher is found in *Artificial Intelligence* by Patrick Winston[2].

Actor languages provide a new perspective to Turing's idea of the incremental computer and it is obvious that LISP is a superb vehicle for this generic type of language. The general utility of LISP is not limited to this application alone, but extends into all corners of the Artificial Intelligence community. By chipping away at the collective bricks of intelligence with tools like LISP, we will soon have dissected the process of thinking. It is ironic that Artificial Intelligence itself is a recursive process, the mind studying the mind. ▶▶J

(Figure II and III on following page)

## References

[1] Kahn, Kenneth M. *An Actor-Based Animation Language*. AI Working Paper #120. Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

[2] Winston, Patrick Henry. *Artificial Intelligence*. Addison Wesley Publishing Co., 1977.

**Other recommended reading:**

Nicol, Ronald L. *Supersoft LISP Users Manual*. Supersoft Associates, 1981.

Siklossy, Laurent. *Let's Talk Lisp*. Prentice-Hall Inc., 1976.

**You Read**

**Dr. Dobb's Journal**

**And You Don't Subscribe?!**

**Save $13**
**off newsstand prices for 2 yrs.**
**Save $5 for 1 yr.**

Can you afford to miss an issue with information vital to your interests? As a subscriber you can look forward to articles on Small-C, FORTH, CP/M, S-100, Compiler optimization, Concurrent Programming and more, delivered right to your door. And you'll never miss the issue that covers your project.

**DR. DOBB'S JOURNAL**
**P.O. Box E**
**Menlo Park, CA 94025**

**Yes!** Sign me up for
____ 2 yrs. $47 ____ 1 yr. $25

____ I enclose a check/money order
____ Charge my Visa, MasterCard, American Express
____ Please bill me later

Name _____

Address _____

_____

_____ Zip _____

Credit Card _____ Exp. date _____

Account No. _____

Signature _____

```
(PUTPROP (QUOTE TRANSLATE)
         (QUOTE (LAMBDA()(PROG(MESSAGE)
                              LOOP (SETQ MESSAGE (READ))
                                   (APPLY(GETANS(CAR MESSAGE)
                                                (CADR MESSAGE))
                                          (LIST MESSAGE)
                                          (ALIST))
                                   (GO LOOP))))
         EXPR)
```

Figure II. Defining TRANSLATE which sends the proper messages to the actors.

```
(TRANSLATE)
(SOMETHING MAKE ORGANISM)
(ORGANISM MAKE PLANT)
(ORGANISM MAKE ANIMAL)
(PLANT REMEMBER METABOLISM (PHOTOSYNTHETIC OR DERIVED FROM
PHOTOSYNTHETIC ANCESTORS))
(ANIMAL REMEMBER METABOLISM (ALL DERIVE FOOD FROM OTHER
ORGANISMS))
(PLANT REMEMBER LOCOMOTION (NONMOTILE AND ARE DISPERSED IN
SPECIALIZED REPRODUCTIVE PHASES SUCH AS SPORES OR SEEDS))
(ANIMAL REMEMBER LOCOMOTION (AT LEAST ONE MOBILE PHASE IN
DEVELOPMENT))
(PLANT REMEMBER CELL-STRUCTURE (CELL WALL))
(ANIMAL REMEMBER CELL-STRUCTURE (CELL MEMBRANE))
(ANIMAL MAKE PROTOZOA)
(ANIMAL MAKE METAZOA)
(PROTOZOA REMEMBER HABITAT (MOIST OR AQUATIC FRESH WATER AND
MARINE))
(PROTOZOA REMEMBER REPRODUCTION (SEXUAL AND ASEXUAL))
(PROTOZOA MAKE PLASMODROMA)
(PLASMODROMA REMEMBER LOCOMOTION (POSSESS FLAGELLA AT SOME
STAGE OF LIFE CYCLE))
(PLASMODROMA MAKE SARCODINA)
(SARCODINA REMEMBER LOCOMOTION (MOVE BY MEANS OF PSEUDOPODS AND
A FEW BY FLAGELLA))
(SARCODINA MAKE AMOEBA)
(AMOEBA REMEMBER LOCOMOTION (MOVE BY MEANS OF PSEUDOPODS))
(METAZOA MAKE CHORDATA)
(CHORDATA REMEMBER EMBRYONIC-CELL-LAYERS THREE)
(CHORDATA REMEMBER DIGESTIVE-SYSTEM (TUBULAR:  TWO OPENINGS))
(CHORDATA REMEMBER CIRCULATORY-SYSTEM PRESENT)
(CHORDATA REMEMBER SYMMETRY BILATERAL)
(CHORDATA MAKE VERTEBRATA)
(VERTEBRATA MAKE MAMMALIA)
(MAMMALIA MAKE PRIMATES)
(PRIMATES MAKE HOMO-SAPIEN)
```

FigureIII. Teaching the actors some facts about organisms.

drilling out the memory IC expansion locations on the logic boards to prevent the purchaser from expanding their computer to 32K. The hardest hit by this is schools, who cannot expand them themselves to 32K once this is done. I know of several schools who purchased these units and now are sorry they did. This I feel is unfair to purchasers and to the dealers who sell these machines. The stores have to exchange the board for a fully expanded one to 32K at a cost of about $500, passed on to the purchaser.

These type of schemes by microcomputer manufacturers can only hurt the industry and result in lost sales, not only for Commodore but for other manufacturers as well.

I am collecting user evaluations of microcomputer products and peripherals (printers, terminals, modems, etc.). Of special interest is schemes like the above for prospective purchasers to be aware of. I expect to publish my results if I get enough responses.

Sincerely,
Harold E. Cohen
c/o HSC Computer Services, Ltd.
P. O. Box 43
Brooklyn, NY 11236

### Pascal Peeve . . .

Dear Marlin,

I have just finished debugging the Pascal Prettyprinting Program that is in the appendix of *Pascal With Style* by Henry Ledgard, Paul Naugn, and John Hueras. After fixing all my typos (no easy task) I still had a problem. The procedure SKIPSPACES on page 176 returned random numbers for the number of spaces skipped. Setting the variable SPACESBEFORE equal to zero on entry fixed the problem. Anyone else typing in the program might like to know.

By the way, the program is an excellent tool that illustrates a complex program, and performs a useful function. I like your new format, and think the Clinic is a great idea.

Sincerely,
Jim Shaw
7021 N. Park Ave.
Indianapolis, IN 46220

### . . . and Praise

Dear Sir:

I'd like to let my fellow *DDJ* readers know about a fine software product: Dynasoft Pascal. Dynasoft's version of Pascal is a tiny Pascal for 6800, 6809 or 6502 computers and requires only 12K of memory and a single cassette recorder. The program is distributed on KC or Apple standard cassettes and comes with a 22-page manual and a 4-page supplement for the particular microprocessor used. The cost is only $45 postpaid from Dynasoft Systems Ltd., P. O. Box 51, Windsor Junction, Nova Scotia, Canada B0N 2V0. Dynasoft can accept checks

*(Continued on page 34)*

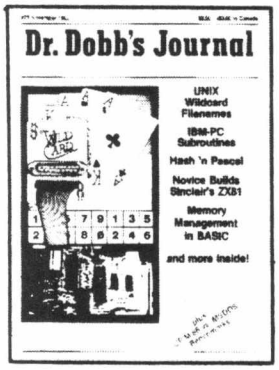

# Dr. Dobb's Journal

For Users of Small Computer Systems

**Each issue includes:**
- valuable software tools
- algorithms & problem solving
- industry news
- important product reviews

**With in-depth coverage of:**
- telecommunications
- systems programming
- language development
- machine independent programs

**and much, much more!**




**Yes! Please enter my subscription for**
☐ 1 yr. (12 issues) $25 ☐ 2 yrs. $47 (save $13 off newsstand)
☐ Please bill me ☐ I enclose check/money order

Name_____

Address _____

City_____ State_____ Zip_____

**Dr. Dobb's Journal, A Publication of People's Computer Company**
**1263 El Camino Real, P.O. Box E, Menlo Park, CA 94025**

## Are you ready?

*DDJ*, the world's foremost microcomputer publication, has been working for years to prepare its readers to be innovators, to lead the wave of breakthroughs in our changing technology.

Every issue of *Dr. Dobb's Journal* helps one to understand the nuts and bolts of small computer systems. We offer entire listings of valuable software: our pages have included compilers, cross-assemblers, editors, new languages, hardware interfaces and more – usually before anyone else thinks of them!

## Even more important!

Our subscribers share insights, correspond, and contribute to one another's work, more than any other group we know. They treat *Dr. Dobb's Journal* as a "hands-on" publication.

This warm cooperation has done more to refine software products, and generally to advance the state of microcomputer technology, than perhaps any other resource. And it is available to you through our pages!

## For the straight facts . . . . . .

If you are a serious computing professional or enthusiast, then you should take a very close look at what *DDJ* offers you. We've been on the cutting edge since 1976.

# Marching the 8080

## A Simple Memory Chip Tester

**by James Laurino**

*James Laurino, 414 S. 47th St., Philadelphia, PA 19143.*

This paper addresses the problem of showing that a microcomputer memory is functioning properly. Virtually every microcomputer user is faced with this problem sooner or later, and my experience indicates that few users have a good solution available. I will describe a simple and powerful basic test program which I adapted from a memory chip tester. I have used various forms of this program successfully for several years on several systems.

This is not a general purpose worst case test, such tests are developed with a particular system in mind. Nevertheless, the basic test of correct function is adequate for many purposes, and an understanding of the technique will provide a starting point for developing more stringent tests.

This test is fast: its running time is only proportional to the size of the memory, not the size squared. It is easy to code, requiring about 20 to 30 bytes of program. Finally, it requires that the memory have the specified number of separately addressable, correctly operating bits in order to pass.

Let me emphasize the last claim for this test: it is necessary to show that each bit in the memory operates independently of all the others. Many tests ignored this issue, or only half resolve it by allowing wrap arounds in certain cases.

How can one show that all memory locations are separate? In larger systems the conventional quick test is a store-address-as-data pattern, that is, at each location in memory write the address of that location, then go back again to each location and verify that it contains the correct address. This test is effective for machines where each memory word can store the full address bit pattern, but it doesn't work for 4- or 8-bit machines.

The simplest approach to think of in this latter situation is probably something like the following:

- Store all zeros in every location in memory.
- Go to the first location and change it to a one.
- Now check all the other locations to see if they still have zeros.
- Finally, restore the zero to the first location and go to the next location.
- Apply this process to all succeeding locations until the whole memory has been checked.

This shows that all locations can store ones and zeros, and that each location functions independently from all the others.

The problem in using this test comes from its long running time. The test of each location requires looking at all the other sites once, giving a total test time proportional to the square of the memory size. Thus, for a 32K memory we get a test time proportional to $2^{15}$ squared, or $2^{30}$, approximately a billion memory tests. At 20 microseconds a test (say a five instruction inner loop) this would take about 20,000 seconds, or more than five hours.

There are some symmetries we can exploit to shorten the test. For instance, if one location, A, is tested and shown to be functioning independently of another, B, then when the time comes to test B we can omit A from the list of location to check. If we apply this simplification systematically, we eliminate an average of half of the checks.

A further consequence of not checking A after it has passed its own test is that we no longer need to restore the zero, since we won't do any checks here again. This saves some time and it allows an even greater simplification. Many checks can be performed at once.

Consider this: since location A is going to stay one, we can defer checking B for any change until the most convenient time. This would be when the normal course of our test procedure brought us to the point of showing that B was independent of all the remaining untested locations. If we read B before changing it to a one, and if we find it still a zero, then the one left in location A must not have affected it – it must be a separate memory cell from A. In fact, since all the locations tested before B have been left ones, we have shown in a single read that B is different from them all.

The next step in the test is to make B a one. If we now proceed to the next location after B and find it still a zero we have shown that it in turn is independent of all its predecessors, including B.

We can proceed thus through the memory checking each location for zero and leaving it a one until we get to the end. If all the checks are passed we have shown that each location is separate from its predecessors and from its successors, and thus that they all function separately.

This is obviously a much simpler test. We need only to make two passes through the memory, one to set it to all zeros, and another to check for the zeros and write the locations to ones as they are checked. Thus the test length is only proportional to the memory size and will take a tolerable time.

There are many variations and refinements of this scheme possible, and I will discuss some of the ones I have used after I demonstrate the coding of the basic algorithm for the 8080.

The core of the program has to be two successive loops, the first to fill the memory with the starting pattern and the second to check the pattern and complement it. A high level description would be:

- For every value in the address range:
  store the starting pattern;
- For every value in the address range:
  check that the starting pattern is still there,
  complement the pattern.

At this point, two other kinds of tests need to be considered. The first is the proof that each bit in the memory can be set to both a one and a zero. One way to do this would be to read each location after it had been complemented to see that the complement pattern stored correctly. Another way would be to run the test a second time with the complement pattern to start. The other issue is to show that all the bits in all memory words are independent of each other. We can resolve both issues at once. By making different bits in the word assume opposite levels we show they are independent, and if we do it both ways we show that they can be both one and zero. For an 8-bit machine, a simple and adequate way to achieve this is to use a walking pattern, that is, moving a single one bit through successive bit positions and repeating the test for each of the 8 patterns. This results in an outer loop to generate the patterns.

- For each of the eight patterns: perform the march test (as above).

I have coded a realization of this procedure as an 8080 assembly language

subroutine below. It uses all registers except E. For speed and simplicity, the program requires that addresses start and end on 256 byte boundaries, or in other words, that the low address byte be zero at both ends of the range. When the program is called, B must contain the high byte of the highest address in the range plus one, and C must contain the high byte of the lowest address in the range. For example, to test an 8K memory starting at zero, C would contain zero and B would contain 20 hex. D is used to hold the current test pattern, one of the set, 1, 2, 4, etc. up to 80 hex.

If the test fails, then there will be a non-zero condition upon return, H and L will point to the failing location, and A will have the bad data pattern.

Many failures are simple to diagnose. For example, if one address line is stuck or ignored, then memory will repeat at an interval corresponding to the binary weight of the bad address line. If address line A3 ($2^3$) were stuck, then the test would fail at address 8. Obviously, you must test all the memory at once if you want to find address interactions.

The program as it stands is designed to be small, both for easy hand loading, and to make the main points obvious. Here are some suggestions about the ways that the program can be extended.

I personally usually include a check of each location once it has been complemented (that is between lines 14 and 15), because this is needed for a single pattern test and because I once found a problem this way.

When a longer test is desired, the program can easily be changed to use all 256 possible test patterns instead of only eight.

When the check scan of memory is complete, the program can be extended to do a new check using the complement pattern as the starting data.

It should be clear that the test doesn't depend on the actual sequence in which the memory locations are tested, and another order may make a more stringent, but slower, test. Similarly, the data pattern needn't be constant but can be any function of the address.

For establishing independence between bits in a very wide word, it would be faster to divide the word in half – half the bits set as one, and half as zero – and then divide each group in half again, etc.

In conclusion, let me just say that this technique has served me well and I hope you will find it useful in solving your own memory test problems. **DDJ**

MARCH Subroutine

upon call: B= upper limit, C= lower limit.

changes: D,H,L,A, Flags.

upon return: Z flag set if test passed.

| line | label | op | arguments | comments |
|---|---|---|---|---|
| 1 | START | MVI | D,ONE | set initial pattern |
| 2 | | MVI | L,ZERO | start on 256 byte boundary |
| 3 | REPEAT | MOV | H,C | set lowest address |
| 4 | | MOV | A,B | setup end test value in A |
| 5 | FILL | MOV | M,D | store pattern in current location |
| 6 | | INX | H | go to next location |
| 7 | | CMP | H | check for end of range |
| 8 | | JNZ | FILL | continue till end reached |
| 9 | | MOV | H,C | set lowest address again |
| 10 | CHECK | MOV | A,M | read current location |
| 11 | | CMP | D | compare it to initial pattern |
| 13 | | CMA | | get pattern complement |
| 14 | | MOV | M,A | store it at current location |
| 15 | | INX | H | go to next location |
| 16 | | MOV | A,B | setup end test |
| 17 | | CMP | H | check for end of range |
| 18 | | JNZ | CHECK | continue till end reached |
| 19 | | MOV | A,D | get the current pattern |
| 20 | | RLC | | generate the next pattern |
| 21 | | MOV | D,A | save next pattern in D again |
| 22 | | JNC | REPEAT | continue till bit shifts off left end |
| 23 | | RET | | then done, return with Z set by last CMP H |

31 bytes

Why wait
to start
increasing your sales?
Call now
to reserve
advertising space
in the next
*Dr. Dobb's Journal*
(415) 323-3111

# For CP/M BIOS

## An Improved MX-80 Printer Driver

**by Gary P. Novosielski**

*Gary P. Novosielski, 21 W. Pierrepont Ave., Rutherford, NJ 07070.*

The Epson MX-80 printer is certainly one of the better printer values on the market for the small system user. I feel safe in saying so because it's also one of the hottest selling. But no hardware is perfect, and the MX-80 has a few peculiarities worth talking about.

No, not just talking about, doing something about, because by using a simple software driver, and by taking advantage of the built-in logic of the printer, it can be made substantially more versatile.

First, a little background about the printer itself. In a call for help in Doctor Dobb's Clinic (*DDJ* #58), Turner Caldwell is quoted as saying "there is a catch" in the printer logic. Mr. Caldwell states that control codes embedded in the text are interpreted differently depending on the direction the bidirectional print head is travelling. Let's get that one out of the way first. 'Tain't so.

Oh, they may be interpreted in unexpected ways, but it has absolutely nothing to do with the direction of travel of the print head. The printer and its support logic will accept data bytes from the cpu output port, buffering all characters until a carriage return, line feed, form feed, or vertical tab is received, or until the line is full, which forces a new line. Only then is the physical printing performed. The direction of travel of the print head is chosen in such a way as to minimize the travel distance.

How does it work? Very well. That's all we need to care about. We can think of each line as being printed left-to-right, and let the printer logic worry about the details. One point is important to the following discussion, however: a carriage return is merely a data byte which instructs the printer to print the buffer contents, *if any*. It does *not* return the carriage. For example, suppose we send 30 spaces to the printer, a carriage return character, another 29 spaces, another carriage return, 50 spaces, and so forth, the print head won't move at all. We haven't sent any printing characters or line feeds. Until there's something to print, all the leading spaces, tabs, and control characters just tell the printer where we want the characters to go, and what we want them to look like. How it gets them there is its own business.

Contrast this with the actions of an early generation teletypewriter. If we fed it the same input, the print head would clatter and bump across the line printing nothing, but following each instruction to the "letter." The result is the same, with one important difference. All that mechanical motion took just as much time as if we were printing something. The Epson sat there silently accepting characters at the full speed of the parallel output port (which is pretty darn fast) and just smiled. That's what "smart printer" means. The print head moves only when, and only as far as necessary.

Enough background, what's the story on the control characters? Well, to be fair to Mr. Caldwell, the MX-80 manual, written by David A. Lein, is not very specific about the workings of these codes. This manual, which has received rave notices in all the product reviews I've seen, is described as a "learner's manual," not a "user's manual," and unfortunately that's exactly what it is. It is written in an excessively chatty style, aimed squarely at a novice computer user who barely knows his way through a simple program in Radio Shack Level I BASIC. That's fine, but when you want a straight answer to a straight question, you have to wade through a mound of sample programs and "gee-whiz" commentary.

The plain facts on control characters, as far as I have been able to determine them empirically are these:

1. Double Width Characters
   (ON=CTRL-N; OFF=CTRL-T)

This mode causes each dot of each character to be printed twice, causing the character to occupy two print columns. The on and off codes can be embedded in the print line to turn the mode on or off for specific characters, and are handled properly as read from left to right, even if the printer decides to print the line backwards. This mode is reset to off automatically when a physical print operation occurs; i.e., when the carriage return to end the line is received. The programmer must specifically set it on again if multiple lines are to be printed in this mode. The implication of use with a word processing program must be stressed here. Since, with many programs the user does not know in advance where individual print lines will end, the only safe method of printing several double-width words in a row is to precede each of the words with a CTRL-N character in case a line is broken between the words, which would set the mode off prematurely.

2. Tab characters (CTRL-I)

Tabs are handled as expected, causing the next character to the right to print at the next tab stop. When setting tabs, the bytes following the ESC "D" are treated as 7-bit binary numbers, high bit ignored. The sequence is terminated with a NUL (00H or 80H). All previous tab stops are cleared. The first column of the line is column zero. Obviously you can't set a tab stop there. If no tabset sequence has been sent, tab characters are ignored. The maximum number of tabs is said to be 112. Try though I might, I have been unable to get tabs to operate while Double Width mode is turned on. They seem to be ignored totally in that mode.

The above codes are the only ones which can be embedded in the buffer to affect individual portions within a single line. The following three modes all work in a different manner from the above, but similarly to each other. They all set or reset internal flags governing the printing of the buffer as a whole, and are not actually embedded in the buffer at all. They can occur at any time prior to physical printing with the same effect regardless of their position in the line. The most recently transmitted code of any on/off pair determines the condition of the respective flag. The flags are not automatically reset at the end of a line, and remain set until specifically reset by the off code.

3. Compressed Printing
   (ON=CTRL-O; OFF=CTRL-R)

Causes the speed of print head travel across the page to be reduced by a factor of 1.65, while still printing characters at the normal rate. This allows 132 characters to fit on a single line, which normally holds only 80 characters. Although I stated above that these commands could occur anywhere in the line, you must obviously tell the printer to allow for more than 80 characters before sending the 81st character. Also, if Compressed Printing is on, and is then turned off after more than 80 characters have already been received, an immediate print will occur and the characters past column 80 will be lost.

4. Emphasized Printing
   (ON=ESC "E"; OFF=ESC "F")

When set, causes all characters to have twice as many dots in the horizontal

direction (four times as many if Double Width is also on for that character), but the print head speed is cut by a factor of two, so the characters remain their normal (or double) width, but look darker and more fully formed. The manual implies that the print wires actually hit the paper harder in this mode, which is probably rubbish.

NOTE: Compressed and Emphasized printing cannot both be in effect for the same line. Apparently the print head has only three speeds: Normal, Normal/1.65, and Normal/2.

5. Double Strike Mode
(ON=ESC "G"; OFF=ESC "H")

Each line is first printed normally, as determined by the settings of the above flags. The printer then does a micro linefeed (equal to one dot height) and the entire line is then overprinted, doubling the vertical dot count without appreciably affecting the height of the characters. One would expect this to take twice as long as a single-strike line, but this action is poorly optimized in the printer's logic. The print head always backs up to the opposite end of the line so as to print the second pass in the same direction as the first, thus taking up to three times as long to print such a line.

The combined effect of modes 4 and 5 is to virtually eliminate any space between adjacent dots in either the horizontal or vertical direction. This produces practically letter quality (if somewhat boldface) printeing, with an associated speed penalty – about one fifth normal.

Enough control codes for a while. It's obvious by now that we have a printer of great potential here, but its versatility is limited by the all-or-nothing approach of modes 4 and 5. Mode 3 (compressed) would normally be desirable only for entire lines of pages at a time, and frankly does not look attractive when mixed with non-compressed characters on the same line.

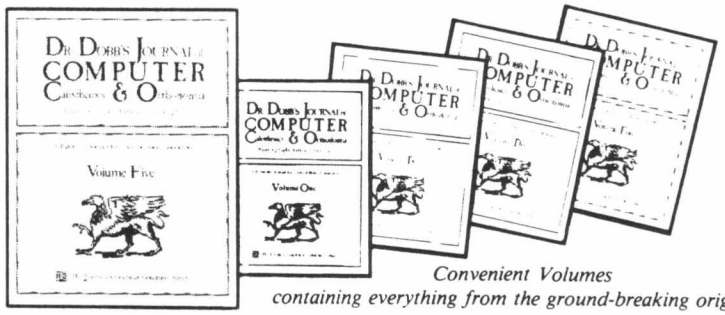
But it would be nice to produce an occasional boldface word in the middle of a line for emphasis. Double Width mode, which can easily accomplish this, has a touch of overkill about it.

Since it would be nice, let's do it. Isn't this the computer revolution where all is possible? One way out is the following: print the line, but leave a space where the boldface word belongs. Send a carriage return, but leave off the line feed so the paper doesn't advance. Space over to where the empty spot is (Hint: remem-


For Your Personal

# REFERENCE COLLECTION

## Thousands of dollars' worth of microcomputer information and listings!




*Convenient Volumes*
*containing everything from the ground-breaking originals*

**Dr. Dobb's Journal Vol. V** – All the ground-breaking issues from 1980 in one volume! Systems software reached a new level with the advent of CP/M, chronicled herein by Gary Kildall and others (*DDJ*'s all-CP/M issue sold out within weeks of publication). Software portability became a topic of greater import, and *DDJ* published Ron Cain's immediately famous Small-C compiler – reprinted here in full!

Contents include: The Evolution of CP/M, a CP/M-Flavored C Interpreter, Ron Cain's C Compiler for the 8080, Further with Tiny BASIC, a Syntax-Oriented Compiler Writing Language, CP/M to UCSD Pascal File Conversion, Runtime Library for the Small-C Compiler, and, as always, even more!

**Dr. Dobb's Journal Vol. I** – The material brought together in this volume chronicles the development in 1976 of Tiny BASIC as an alternative to the "finger blistering" front-panel machine-language programming which was then the only way to do things. This is always pertinent for bit crunching and byte saving, language design theory, homebrew computer construction and the technical history of personal computing.

Topics include: Tiny BASIC, the (very) first word on CP/M, Speech Synthesis, Floating Point Routines, Timer Routines, Building an IMSAI, and more.

**Dr. Dobb's Journal Vol. II** – 1977 found *DDJ* still on the forefront. These issues offer refinements of Tiny BASIC, plus then state-of-the-art utilities, the advent of PILOT for microcomputers and a great deal of material centering around the Intel 8080, including a complete operating system. Products just becoming available for reviews were the H-8, KIM-1, MITS BASIC, Poly Basic and NIBL.

Articles about Lawrence Livermore Lab's BASIC, Alpha-Micro, String Handling, Cyphers, High Speed Interaction, I/O, Tiny Pilot & Turtle Graphics, many utilities, and even more.

**Dr. Dobb's Journal Vol. III** – The microcomputer industry entered its adolescence in 1978. This volume brings together the issues which began dealing with the 6502, with mass-market machines and languages to match. The authors began speaking more in terms of technique, rather than of specific implementations; because of this they were able to continue laying the groundwork industry would follow. These articles relate very closely to what is generally available today.

Languages covered in depth were SAM76, Pilot, Pascal and Lisp, in addition to RAM Testers, S-100 Bus Standard Proposal, Disassemblers, Editors and much, much more.

**Dr. Dobb's Journal Vol. IV** – This volume heralds a wider interest in telecommunications, in algorithms, and in faster, more powerful utilities and languages. Innovation is still present in every page, and more attention is paid to the best ways to use processors which have proven longevity – primarily the 8080/Z80, 6502, and 6800. The subject matter is invaluable both as a learning tool and as a frequent source of reference.

Main subjects include: Programming Problems/Solutions, Pascal, Information Network Proposal, Floating Point Arithmetic, 8-bit to 16-bit conversion, Pseudorandom Sequences, and Interfacing a Micro to a Mainframe – more than ever!

**YES!** ☐ Please send me the following Volumes of *Dr. Dobb's Journal* for $20.75 each.
☐ ALL 5 for ONLY $88, a savings of over 15%!

Please charge my: ☐ Visa ☐ MasterCard ☐ American Express
I enclose ☐ Check/money order

Card # _____ Expiration Date _____

Signature _____

Name _____ Address _____

City _____ State _____ Zip _____

*Prices subject to change.*

Mail to: Dr. Dobb's Journal, P.O. Box E, Menlo Park, CA 94025
Allow 6-9 weeks for delivery.

Vol. I ____ x $20.75 = ____
Vol II ____ x $20.75 = ____
Vol. III ____ x $20.75 = ____
Vol IV ____ x $20.75 = ____
Vol. V ____ x $20.75 = ____
ALL 5 ____ x $88.00 = ____
Sub-total $ ____

**Postage & Handling** ____
**Must be included with order.**
*Please add $1.25 per book in U.S.*
*($1.75 each outside U.S.)*

**TOTAL $** ____

ber what was said earlier about leading spaces). Change the print mode to emphasized. Send the boldface word. Send a CR-LF to advance to the next line.

That seems like a complex way out. The printer would have no trouble, just zip-zip and that's it. The problem is the programmer's. Quite a few minutes with a printer layout sheet would be required to do much of that, and if we're using a word processing program, the whole thing goes out the window when the lines get re-arranged, justified, and what have you. What we need is a way to send control codes in a conventional left-to-right fashion and have them interpret for us automatically. The algorithm above is okay, but the computer should do the work.

To see how this can be done, let's consider first a related fact about the MX-80:

6. Backspace characters (CTRL-H)
Backspaces are ignored.

Well, that's unfriendly to say the least. It tabs, but it won't backspace. Actually that's not at all unusual for a printer with buffered output and bidirectional printing logic, but as long as there's an underline in the character set, a backspace is no luxury item.

Let's do something about that first. A backspace should just move the print head one space backwards. But as we've seen, we can't directly tell the printer how to move the print head. We can only say when to move it, and that's only a suggestion. Therefore, we need a driver subroutine (Listing 1) which will intercept the backspace character, and put the appropriate data bytes out to the printer to make it do what we want it to. Say we are at column N and want to back up one column. We tell the printer to print the line (the first part) which is currently in the buffer. We do this by sending a carriage return but no line feed. This will only work, by the way, if you have set the internal switches to suppress automatic line feed insertion after carriage returns. The next thing is to get back to column N-1, where we want to be. We send N-1 spaces, which the printer takes as advice on where the next character belongs.

To be able to do this, the subroutine needs to know at all times the current position in the output line, just in case the next character turns out to be a backspace. So we have to count printing characters and spaces as we send them out. We don't count control codes or escape sequences. We have to be careful though. For value tests, we have to use only the seven lower bits of each byte for comparison, since the printer recognizes control codes with or without the high bit set; but the character we send out must have all 8 bits intact, or we'll never be able to use the graphics or Katakana chaaracters. Also, within escape sequences, e.g., setting tab stops, some of the data bytes may happen to resemble printing characters, but the counter should not be incremented.

To complicate matters, some escape sequences are of fixed length, while others are of variable length. The Emphasized Printing mode for example, has 2 characters. Setting form length takes three – ESC, "C", n. Where n is the number of lines per page in 7-bit binary. The same for ESC, "A", n; which sets vertical line spacing (n = 72/x; where x is in lines per inch). But vertical tabstop setting may have up to 64 n's in the sequence ESC, "B", n1, n2, . . ., nx, NUL. In this case, our only clue is the occurrence of the NUL at the end.

It turns out that ESC "B" and ESC "D" (vertical and horizontal tabset) are the only variable ones. In these we'll just look for the value 00 or 80H to signal the end. The ESC "A" and ESC "C" sequences are three bytes, and all the rest (even invalid ones ignored by the printer) are two bytes long, counting the ESC itself.

Tab characters must also be properly counted. It would be feasible to intercept the tabset sequence, store the values in a table, and on receipt of a tab character, move the next highest table value into the column counter. I have taken a simpler approach, since I always have tabstops set every 8 columns for compatibility with normal CP/M conventions. The subroutine has a one-shot initialization sequence which sends this tabset sequence to the printer the first time it is called, and thereafter it assumes that a tab always positions to the next column divisible by 8.

The characters 0AH through 0DH, i.e., line feed, vertical tab, form feed, and carriage return all cause printing to occur prior to any paper motion, and all reset the column counter to zero (in the printer as well as in the subroutine). Vertical tabs are treated as line feeds if no vertical tabstops have been set. There is no need to count these for our present purposes. All other non-printing characters are merely sent to the printer and not counted, with one exception.

The final character to be considered is one chosen from those normally ignored by the printer. It will be used for the purpose of printing emphasized or double strike words within a line. Since these modes normally affect an entire line, this new character will have the following conceptual definition:

"Causes the line as transmitted so far to be printed according to the current setting of the mode flags, but unlike a carriage return, does not reset the column counter to zero."

For the purposes of transportability of software I hereby propose a STANDARD character for this purpose for use in this and similar MX-80 driver software – the ASCII "ETB" (End of Text Block) character seems the natural choice. This is the CTRL-W character, 17H or 97H, which presents no conflicts with CP/M line editing conventions.

The subroutine does not pass this character to the printer, but simply calls the backspace routine, and then transmits one space, meeting the requirements of the definition.

Usage is simple. The ETB character is embedded in the text just prior to a mode changing command which is intented to apply to only a subsection of one line. It is followed by the mode set command itself, and the characters to be printed in the new mode. (It is not required for Double Width characters, since they are already handled properly.) Another ETB is embedded just prior to the mode reset command. Use of this embedded character method should make use with word processing programs relatively simple. It also has the advantage that it does not require any modification to existing MX-80 compatible software. If you don't use the ETB character, the printer performs normally.

Since the subroutine is intended for use in the BIOS section of a CP/M system, it expects to find the character in the C register. The subroutine call to PROUT is a call to an EPROM resident routine which sends the character in the B register to the parallel port when the ready flag is true.

If is is not practical or desirable to use it in this way, simply write a replacement for PROUT which saves the registers, sends the byte in B to the printer via BDOS call or whatever method is used in your DOS, restores the registers and returns. Then include the driver in the ap-

plication software. I chose to make the installation once and be done with it rather than have to include special drivers in every piece of software, some of which I have no source code to; PIP for instance.

One point of caution is required if Double Strike mode is used with backspacing or the ETB function. Double Strike, as mentioned earlier does a 1/72 inch line feed as part of the printing function. This causes a minor misalignment of the remainder of the line. The effect is barely noticeable if it occurs only once or twice in a line, but can add up if incautiously used. In backspacing, the following method is preferred. Backslashes represent the backspace character.

```
THE WORDS OP CIT\\\\\\__ ___
    SHOULD BE UNDERLINED.
```

This causes the line to be printed in only two passes:

```
THE WORDS OP CIT
              __ ___ SHOULD BE
    UNDERLINED.
```

Consider the alternative method in which the backspace character immediately follows each character to be underlined.

```
THE WORDS O\_P\_ C\_I\_T\_
    SHOULD BE UNDERLINED.
```

This causes six passes:

```
THE WORDS O
          _P
            _ C
              _I
                _T
                  _ SHOULD BE
    UNDERLINED.
```

If the Double Strike mode is on and this second method is used, the words will seem to march down the page as in the illustration, though not as exaggerated, of course. The first method will minimize this effect. If a text processing program is used which automatically breaks up lines, we might find that a word is printed at the end of one line and its underlined characters don't make it onto the same line. To avoid this, compromise by underlining each word individually:

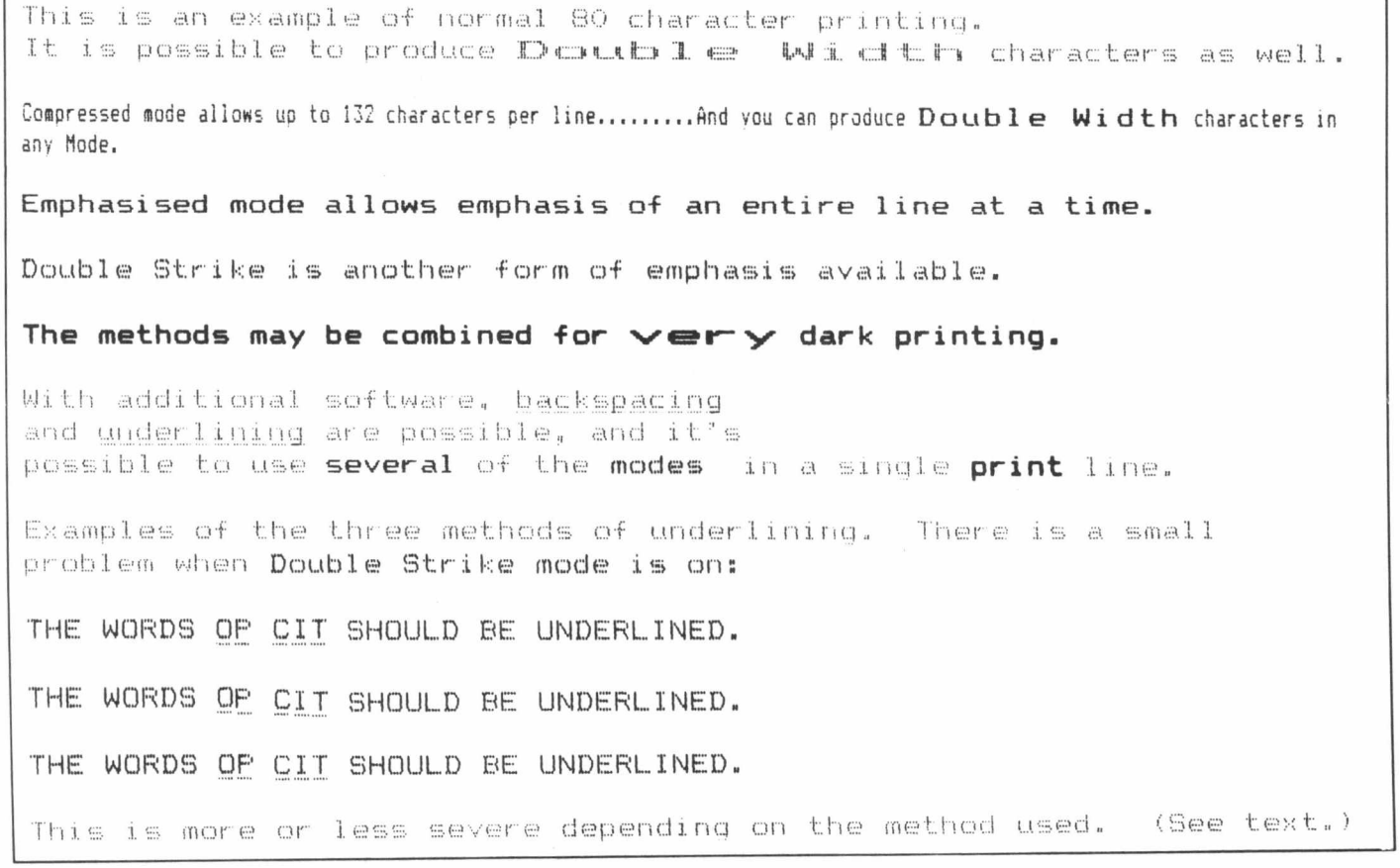
```
THE WORDS OP\\__ CIT\\\___
    SHOULD BE UNDERLINED.
```

This ensures that a word and its underline form a contiguous string with no intervening blanks. In this example, three passes are performed. Actual examples of the printing are shown in Figure 1.

As an additional aid to implementation of applications using the features of the MX-80, here are a few points which are not clearly stated, not stated, or incorrectly stated in the manual.

1. The form length is remembered internally in terms of print lines, not

**Figure I.**



inches. If the number of lines per inch is changed in mid-page, a form feed will not properly advance to the top of page, since the same number of lines now take up a different number of inches. It really gets loused up.

2. You can't remedy the situation above by changing the form length. Changing the form length in mid-page also resets the counter to zero, making wherever we are the new top of form. A form feed will then advance to the same mid-page spot on the next page. The best bet is to only change form length and line spacing when at the physical top of a page, or else count all your lines in software, and use line feeds to position the paper.

3. The control codes in Appendix B of the manual for line spacings, ESC "0", ESC "1", and ESC "2" are the ASCII characters "0", "1", and "2", i.e., 30H, 31H, and 32H, *not* 00H, 01H, and 02H as implied in the appendix. Also, ESC "0" and ESC "1" have nothing to do with the ESC "A" command as is stated.

4. The CANCEL code does *not* restore compressed characters to normal width. DEL probably doesn't either, but in any case should be left disabled via the configuration switches. The DEL character has a long story behind it dating back to punched paper tape. Printers are supposed to ignore it. Make sure yours does, if you don't want a headache some day when a piece of software assumes it does.

5. Moving on to Appendix I, the form length is not limited to 66 lines. I commonly use 88 lines per page with 8 line-per-inch spacing and it works fine.

6. The description of the escape codes states properly that the series of bytes in tabset sequences are treated as seven-bit binary numbers. It then states that vertical tabs can be set from line 1 to 255. I can't count higher than 127 in seven bits.

7. Mr. Lien states several times in the text that double width characters are turned off after every program line. He doesn't mean print line, but program source line. He makes a point of that. I say printers can't tell what program line you're in. They can't tell if you're in a program. They can't even tell if its a computer they're hooked up to. Double Width is turned off at the end of a print line. Maybe there's a bug in Radio Shack BASIC that's confusing him.

I'd appreciate hearing of any problems with or enhancements to the printer driver. It doesn't solve every problem, but it should do quite a bit to improve the versatility of an already excellent printer.

▶▶J

**(LISTING ON PAGE 30)**

*Dr. Dobb's* readers
have been aiding the growth and development
of the micro field for years.
If you would like to make a contribution,
contact us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

```
                ;       +----------------------------------------+
                ;       |          MX-80 PRINTER DRIVER          |
                ;       |  GARY P. NOVOSIELSKI                   |
                ;       +----------------------------------------+
                ;
                ;       SOME USEFUL EQUATES:
                ;
001F =          CTRL    EQU     '_'-1           ;CONTROL KEY MASK
0000 =          NULL    EQU     CTRL AND '@'    ;SOME ASCII DEFINITIONS
0008 =          BS      EQU     CTRL AND 'H'    ;BACKSPACE
0009 =          HT      EQU     CTRL AND 'I'    ;HORIZONTAL TAB
000A =          LF      EQU     CTRL AND 'J'    ;LINE FEED
000B =          VT      EQU     CTRL AND 'K'    ;VERTICAL TAB
000C =          FF      EQU     CTRL AND 'L'    ;FORM FEED
000D =          CR      EQU     CTRL AND 'M'    ;CARRIAGE RETURN
000E =          SO      EQU     CTRL AND 'N'    ;SHIFT OUT
0014 =          DC4     EQU     CTRL AND 'T'    ;DEVICE CONTROL 4
0017 =          ETB     EQU     CTRL AND 'W'    ;END OF TEXT BLOCK
0018 =          CAN     EQU     CTRL AND 'X'    ;CANCEL
001B =          ESC     EQU     CTRL AND '['    ;ESCAPE
007F =          DEL     EQU     7FH             ;DELETE
                ;
                ;INITIALIZATION STRING FOR PRINTER:
0000 80         INITS   DB      NULL OR 80H     ;MAKE SURE NO ESC SEQUENCE
                                                ;IS IN PROGRESS.
0001 18                 DB      CAN             ;CANCEL ANYTHING IN BUFFER.
0002 1B44               DB      ESC,'D'         ;START THE TABSET SEQUENCE.
0004 0810182028         DB      8,16,24,32,40,48,56,64,72,80
000E 5860687078         DB      88,96,104,112,120
0013 00                 DB      NULL            ;END THE SEQUENCE.
0014 00         PTRINIT DB      0               ;INITIALIZATION FLAG
0015 00         PESCF   DB      0               ;ESCAPE IN PROGRESS FLAG
0016 00         PTRCHAR DB      0               ;COLUMN COUNTER
0017 00         WIDEFLAG DB     0               ;DOUBLE WIDTH FLAG
                ;
0000 =          FALSE   EQU     0
FFFF =          TRUE    EQU     NOT FALSE
                ;
                ;ENTRY TO THE SUBROUTINE IS MADE HERE:
                LPTOUT:         ;ON ENTRY, CHARACTER TO BE PRINTED IS IN
                                ;REGISTER C.
                ;
0018 3A1400             LDA     PTRINIT         ;SEE IF INITIALIZATION
001B B7                 ORA     A               ;IS REQURED.
001C C23000             JNZ     INITD           ;JUMP IF NOT.
001F 210000             LXI     H,INITS         ;POINT TO THE INIT STRING.
0022 46         INITP   MOV     B,M             ;PUT THE CHAR IN B REGISTER
0023 CDEE00             CALL    PROUT           ;FOR THE OUTPUT ROUTINE.
0026 78                 MOV     A,B             ;GET BACK THE CHARACTER.
0027 23                 INX     H               ;POINT TO NEXT CHARACTER.
0028 B7                 ORA     A               ;IF WE JUST SENT A ZERO (NUL),
                                                ;WE'RE DONE,
0029 C22200             JNZ     INITP           ;ELSE DO SOME MORE.
002C 3D                 DCR     A               ;PUTS  OFFH IN THE A REG.
002D 321400             STA     PTRINIT         ;STORE TRUE IN INIT FLAG
                                                ;SO WE DON'T INITIALIZE EVERY
                                                ;TIME.
                ;THE PRINTER HAS BEEN INITIALIZED.
0030 41         INITD   MOV     B,C             ;GET CHARACTER TO BE PRINTED
                                                ;TO THE B REGISTER.
0031 79                 MOV     A,C             ;COPY TO A.
0032 E67F               ANI     7FH             ;BUT ONLY LOW SEVEN BITS.
0034 4F                 MOV     C,A             ;SAVE THE STRIPPED COPY IN C
                                                ;FOR COMPARISONS LATER.
0035 3A1500             LDA     PESCF           ;SEE IF THE ESCAPE FLAG
```

```
0038 B7                 ORA     A               ;IS TRUE, INDICATING
0039 C29D00             JNZ     PESCSP          ;WE ARE IN THE MIDDLE OF AN
                                                ;ESCAPE SEQUENCE.
                ;ELSE THIS IS NOT AN ESCAPE SEQUENCE.  JUST A CHARACTER.
003C 79                 MOV     A,C             ;GET STRIPPED COPY TO A
003D FE7F               CPI     DEL             ;IF ITS A DEL,
003F C8                 RZ                      ;IGNORE IT AND RETURN.
                ;
                ;
0040 211600             LXI     H,PTRCHAR       ;POINT TO THE CHARACTER COUNTER.
                                                ;FOR LATER.
0043 FE20               CPI     ' '             ;CHECK IF THIS IS A PRINTING
0045 D27900             JNC     PRTCHAR         ;CHAR.  IF SO PRINT IT.
                ;THIS IS A CONTROL CHARACTER.  CHECK FOR SPECIAL CASES
0048 FE08               CPI     BS              ;BACKSPACE CHAR?
004A DAEE00             JC      PROUT           ;ANYTHING LESS GOES OUT AS IS.
004D CAD200             JZ      PBS             ;IF EQUAL, PROCESS BACKSPACE.
                ;
0050 FE09               CPI     HT              ;TAB CHARACTER?
0052 CA8500             JZ      PPTAB           ;  IF SO, PROCESS IT.
                ;
0055 FE0E               CPI     SO              ;DOUBLE WIDTH SET COMMAND?
0057 DA7100             JC      NEWLINE         ;ANYTHING LESS MUST BE EITHER
                                                ;LF, VT, FF, OR CR.  HANDLE IT.
005A CAC500             JZ      SETWIDE         ;  IF EQUAL, SET THE FLAG.
                ;
005D FE14               CPI     DC4             ;DOUBLE WIDTH CLEAR COMMAND?
005F CACB00             JZ      CLRWIDE         ;  IF SO, CLEAR THE FLAG.
                ;
0062 FE1B               CPI     ESC             ;ESCAPE CHARACTER?
0064 CA9500             JZ      PESCP           ;INITIATE ESCAPE SEQUENCE
                                                ;PROCESSING.
                ;
0067 FE17               CPI     ETB             ;SPECIAL CHARACTER ETB MEANS
                                                ;DUMP THE BUFFER BUT DON'T
                                                ;AFFECT PRINT POSITION.
0069 CAEA00             JZ      PETBP
                ;
006C FE18               CPI     CAN             ;CANCEL MEANS ERASE THE BUFFER.
                                                ;AND RESET COUNTER.
006E C2EE00             JNZ     PROUT           ;NONE OF THE ABOVE.  JUST SEND.
                NEWLINE:        ;A CHARACTER HAS BEEN RECEIVED WHICH
                                ;RESETS THE COLUMN COUNT.
0071 AF                 XRA     A               ;A CHEAP ZERO
0072 77                 MOV     M,A             ;TO THE COUNTER
0073 321700             STA     WIDEFLAG        ;AND THE DOUBLE WIDTH FLAG
0076 C3EE00             JMP     PROUT           ;SEND WHATEVER IT WAS FROM
                                                ;REGISTER B.
                ;
                PRTCHAR:        ;A NORMAL PRINTING CHARACTER OR SPACE
                                ;IS GOING OUT.
0079 34                 INR     M               ;INCREMENT COLUMN COUNT
                ;
007A 3A1700             LDA     WIDEFLAG        ;IF DOUBLE WIDTH
007D B7                 ORA     A               ;  IS OFF
007E CAEE00             JZ      PROUT           ;  PRINT IT.
                ;
0081 34                 INR     M               ;ELSE INCREMENT A SECOND TIME
                                                ;  FOR DOUBLE WIDTH.
0082 C3EE00             JMP     PROUT           ;  AND THEN PRINT IT.
                ;
                PPTAB:  ;PROCESS A TAB CHARACTER
0085 3A1700             LDA     WIDEFLAG        ;IF DOUBLE WIDTH
0088 B7                 ORA     A               ;  IS ON
0089 C2EE00             JNZ     PROUT           ;  TAB CHARACTERS ARE IGNORED
                                                ;  FOR SOME UNKNOWN REASON.
                ;
```

*(Listing continued on next page)*

```
008C 7E                 MOV     A,M             ;ELSE GET THE OLD COLUMN COUNT,
008D C608               ADI     8               ;  ADJUST TO NEXT

008F E6F8               ANI     NOT 7           ;  TAB STOP
0091 77                 MOV     M,A             ;  AND UPDATE THE COUNT.
0092 C3EE00             JMP     PROUT           ;SEND HT CHAR TO PRINTER.
                ;
                PESCP:          ;AN ESCAPE CHARACTER HAS BEEN RECEIVED
                                ;MAKE A NOTE THAT A SEQUENCE IS IN PROGRESS.
0095 3EFF               MVI     A,TRUE          ;PUT ALL ONES
0097 321500             STA     PESCF           ;  IN THE ESCAPE FLAG
                ;
009A C3EE00             JMP     PROUT           ;SEND ESC TO PRINTER.
                PESCSP:
                ;       PROCESS THE NEXT CHARACTER OF AN ESCAPE SEQUENCE.
009D FEFF               CPI     TRUE            ;IF THE ESCAPE FLAG IS ALL
                                                ;  ONES, WE JUST BEGAN,
                                                ;  SO THIS MUST BE THE
009F CAB800             JZ      SECOND          ;  SECOND BYTE.
                        ;ELSE THIS IS THE THIRD BYTE OR MORE
00A2 FE41       THIRD   CPI     'A'             ;IF THIS IS AN ESC A SEQUENCE
00A4 CAB100             JZ      LAST            ; OR
00A7 FE43               CPI     'C'             ; AN ESC C SEQUENCE,
00A9 CAB100             JZ      LAST            ; THE THIRD BYTE IS THE LAST.
                        ;ELSE THIS SEQUENCE TERMINATES WITH A NUL.
00AC 79                 MOV     A,C             ;SEE IF THIS BYTE
00AD B7                 ORA     A               ;  IS A NUL.
00AE C2EE00             JNZ     PROUT           ;IF NOT WE'RE STILL WITHIN THE
                                                ;SEQUENCE.
00B1 AF         LAST    XRA     A               ;LAST ONE. END THE SEQUENCE
                                                ;  BY...
00B2 321500     NEXT    STA     PESCF           ;  ...UPDATING THE FLAG.
00B5 C3EE00             JMP     PROUT           ;SEND THE BYTE TO THE PRINTER
                SECOND:
                ;SECOND CHAR OF ESC SEQUENCE.  SEE WHAT THE ID CHARACTER IS:
00B8 FE41               CPI     'A'             ;A THRU D ARE 3 OR MORE BYTES
00BA DAB100             JC      LAST            ;  OTHERS ARE JUST TWO
00BD FE45               CPI     'E'             ;  SO WE DON'T SAVE ANYTHING
00BF D2B100             JNC     LAST            ;  JUST PUT THEM OUT.
                                                ;FOR THE LONG ONES, WE SAVE THE
00C2 C3B200             JMP     NEXT            ;ID CHARACTER BY...
                ;
                SETWIDE:        ;A CTRL-N HAS BEEN RECEIVED.
00C5 321700             STA     WIDEFLAG                ;SET THE SOFTWARE FLAG
00C8 C3EE00             JMP     PROUT                   ;  AND THE HARDWARE ONE.
                ;
                CLRWIDE:        ;A CTRL-T HAS BEEN RECEIVED.
00CB AF                 XRA     A                       ;CLEAR
00CC 321700             STA     WIDEFLAG                ;  THE SOFTWARE
00CF C3EE00             JMP     PROUT                   ;  AND HARDWARE FLAGS.
                PBS:    ;PROCESS A BACKSPACE CHARACTER
00D2 AF                 XRA     A               ;CLEAR THE

00D3 321700             STA     WIDEFLAG        ;DOUBLE WIDTH FLAG.
00D6 7E                 MOV     A,M             ;GET THE COLUMN COUNTER.
00D7 B7                 ORA     A               ;IF AT LEFT MARGIN, NO EFFECT.
00D8 C8                 RZ                      ;  SO FORGET IT.
                                                ;ELSE
00D9 56                 MOV     D,M             ;SAVE COUNT IN D
00DA 35                 DCR     M               ;BACKSPACE THE COUNT.
00DB 060D               MVI     B,CR            ;PRINT THE LINE SO FAR BY SEND-
                                                ;  ING A CARRIAGE RETURN
00DD CDEE00             CALL    PROUT           ;  TO THE PRINTER.
                                                ;NOW GET READY TO SEND SOME
00E0 0620               MVI     B,' '           ;  SPACES.
```

```
                PBS1:                   ;SPACE OVER TO WHERE WE BELONG.
 00E2 15                DCR     D                       ;DONE YET?
 00E3 C8                RZ                              ;  LEAVE IF SO
 00E4 CDEE00            CALL    PROUT                   ;ELSE PUT OUT A SPACE.
 00E7 C3E200            JMP     PBS1                    ;  UNTIL WE GET BACK TO WHERE
                                                        ;  WE ONCE BELONGED.
                ;
                PETBP:  ;HANDLE THE ETB CHARACTER. DONE SAME AS BS
                        ;EXCEPT ONE MORE SPACE IS SENT.
 00EA 34                INR     M                       ;ONE MORE SPACE
 00EB C3D200            JMP     PBS                     ;GO TO IT
                ;
 00EE =         PROUT   EQU     $
                        ;THE PROUT ROUTINE MUST SEND THE CHARACTER
                        ;IN THE B REGISTER TO THE PRINTER, AND
                        ;RETURN ALL REGISTERS IN THEIR ORIGINAL
                        ;CONDITION, EXCEPT FOR A AND FLAGS.
                        ;
                        ;THE FOLLOWING IS AN EXAMPLE.  THE LINES
                        ;BEGINNING WITH A "U" CONTAIN UNDEFINED PORT
                        ;VALUES WHICH DEPEND ON THE USER'S INDIVIDUAL
                        ;HARDWARE.
                        ;
U00EE DB00              IN      STATUSPORT
U00F0 E600              ANI     BUSYMASK
 00F2 C2EE00            JNZ     PROUT                   ;MAKE IT "JZ" IF THE FLAG
                                                        ;BIT IS A "READY"
                                                        ;RATHER THAN A "BUSY".
 00F5 78                MOV     A,B
U00F6 D300              OUT     DATAPORT
 00F8 C9                RET
                        ;
                ;
 00F9                   END


0008 BS         0018 CAN        00CB CLRWIDE    000D CR         001F CTRL
0014 DC4        007F DEL        001B ESC        0017 ETB        0000 FALSE
000C FF         0009 HT         0030 INITD      0022 INITP      0000 INITS
00B1 LAST       000A LF         0018 LPTOUT     0071 NEWLINE    00B2 NEXT
0000 NULL       00D2 PBS        00E2 PBS1       0015 PESCF      0095 PESCP
009D PESCSP     00EA PETBP      0085 PPTAB      00EE PROUT      0079 PRTCHAR
0016 PTRCHAR    0014 PTRINIT    00B8 SECOND     00C5 SETWIDE    000E SO
00A2 THIRD      FFFF TRUE       000B VT         0017 WIDEFLAG

:1000000080181B4408101820283038404850586089
:10001000687078000000000003A1400B7C230002178
:10002000000046CDEE007823B7C222003D32140016
:100030004179E67F4F3A1500B7C29D0079FE7FC82F
:10004000211600FE20D27900FE08DAEE00CAD200A6
:10005000FE09CA8500FE0EDA7100CAC500FE14CA88
:10006000CB00FE1BCA9500FE17CAEA00FE18C2EEBE
:1000700000AF77321700C3EE00343A1700B7CAEE6C
:100080000034C3EE003A1700B7C2EE007EC608E6A1
:10009000F877C3EE003EFF321500C3EE00FEFFCA44
:1000A000B800FE41CAB100FE43CAB10079B7C2EE42
:1000B00000AF321500C3EE00FE41DAB100FE45D2BA
:1000C000B100C3B200321700C3EE00AF321700C355
:1000D000EE00AF3217007EB7C85635060DCDEE00E4
:1000E000062015C8CDEE00C3E20034C3D200DB0009
:0900F000E600C2EE0078D300C95D
:0000000000
```

End Listing

drawn on U.S. banks. Also available is an OEM version for $85. This version includes a listing of the p-code interpreter and a license to distribute the interpreter and software in p-code. A ROM version is available, but I don't know the price.

I bought the SWTPC 6800 hobbyist's version of Pascal. The package actually is a complete software development system for microcomputers, consisting of a supervisor, editor, Pascal compiler and p-code interpreter. The system loads up to address 2160 hexadecimal, leaving about 3.5K free in a 12K system. This is comfortable for medium-size programs. The supplied tape is set up for SWTPC's SWTBUG monitor ROM, which I have on my machine, and I had no problem in getting the tape to load and run. Enough information is given for someone with a different monitor ROM to make all the patches needed to get Pascal on the air; as an example of custom I/O routines, patches to use the JPC Products TC-3 high-speed tape interface and CFM/3 tape file management system are included in the supplement for 6800 computers.

The syntax of Dynasoft Pascal is, with few exceptions, consistent with standard Pascal within the subset implemented by Dynasoft. Variable types include INTEGER, CHAR, BOOLEAN, SCALAR, SUBRANGE, POINTER and ARRAY. This Pascal permits long variable names, but looks at only the first 4 letters; therefore, BUFF and BUFFER refer to the same variable. The only type checking is done in the I/O formatting routines, which means that INTEGER values may be assigned to POINTERS and vice versa. This permits building linked lists, otherwise impossible because this version of Pascal lacks the RECORD data type. Being able to assign values to pointers makes it possible to directly manipulate memory locations and I/O ports.

Control structures included in Dynasoft Pascal include PROCEDURES and FUNCTIONS, which may be declared as FORWARD or EXTERNAL. Facilities are provided for passing parameters to machine-language subroutines via the Pascal parameter stack. Other control structures included are IF . . . THEN . . . ELSE, CASE . . . OF . . . OTHERWISE, REPEAT . . . UNTIL, WHILE . . . DO and FOR . . . DO. *Not* included are labels and the goto.

To me, Dynasoft Pascal is best as an inexpensive way to learn and experiment with Pascal. The supplied manual, while good as a reference to Dynasoft Pascal, doesn't qualify as a teaching tool for novices. A look in the local B. Dalton's book store revealed several "how-to Pascal" books, any of which would be helpful to the beginner. An experienced Pascal user would find Dynasoft Pascal easy to use – just the thing to use instead of an assembler for that next project. A lot can be done in Dynasoft Pascal and the low price makes up for Dynasoft's limitations. After all, I'd have to pay ten times as much for a copy of the UCSD system, not to mention the far more extensive hardware requirement. Dynasoft Pascal certainly rates a "best buy."

Very truly yours,
William R. Hamblen
946 Evans Road
Nashville, TN 37204

DDJ

*DDJ's* responsive readership can give you invaluable feedback on your ideas and innovations. Find out how you can reach them by contacting us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# Connect a Parallel Printer to Northstar

## More on the MX-80


**by Alan Bomberger**

*Alan Bomberger, 3721 Starr King Circle, Palo Alto, CA 94306.*


I would like to share my recent experience connecting an MX80 printer to the parallel port of my North Star computer. A conversation with the North Star "hotline" reaffirmed my belief that the "load" pulse available on the spare pin of the parallel interface was too short to serve as an output strobe for the MX80 printer. The PO (parallel out) flag is the wrong polarity to serve as an output strobe so I was left with either a software solution or a hardware modification.

The software solution suggested by North Star involved using the high order bit of the output byte as the output strobe. To output to the printer, three OUT 0 instructions are issued. The first is with the high order bit on, the second with the high order bit off, and the third with it on again. If the high order bit from the parallel interface is wired to the MX80 data strobe and the high order bit of the MX80 data lines is tied to ground, the result is correct operation of the printer. This is, in fact, the way North Star has written the parallel output routine in its DOS system. The disadvantage is that no graphics are possible.

Since I wanted to use the graphics capabilities of the MX80, I sought a hardware solution. My CP/M system assumes that the existing output strobe is sufficient to operate the external device and it uses the sequence OUT 0, "reset parallel output flag", expecting the acknowledge signal to set the parallel output flag. The North Star "hotline" suggested that I extend the pulse on the spare line and use it for the printer strobe. There is, however, a serious problem with this solution. If interrupts are enabled and an interrupt is received between the OUT 0 instruction and the "reset parallel output flag", the system may hang up. The hangup happens when the interrupt routine takes enough time so that the acknowledge signal returns from the printer before the "reset parallel output flag" instruction executes.

There are two solutions. First, the order of the instructions can be reversed so that the Output Flag is cleared before the OUT 0 instruction. In this way, the interrupt will only delay the issue of the OUT 0. Second, the PO flag can be used to generate the output strobe. I chose the latter as it made no difference in the hardware needed and did not require any changes to the software.

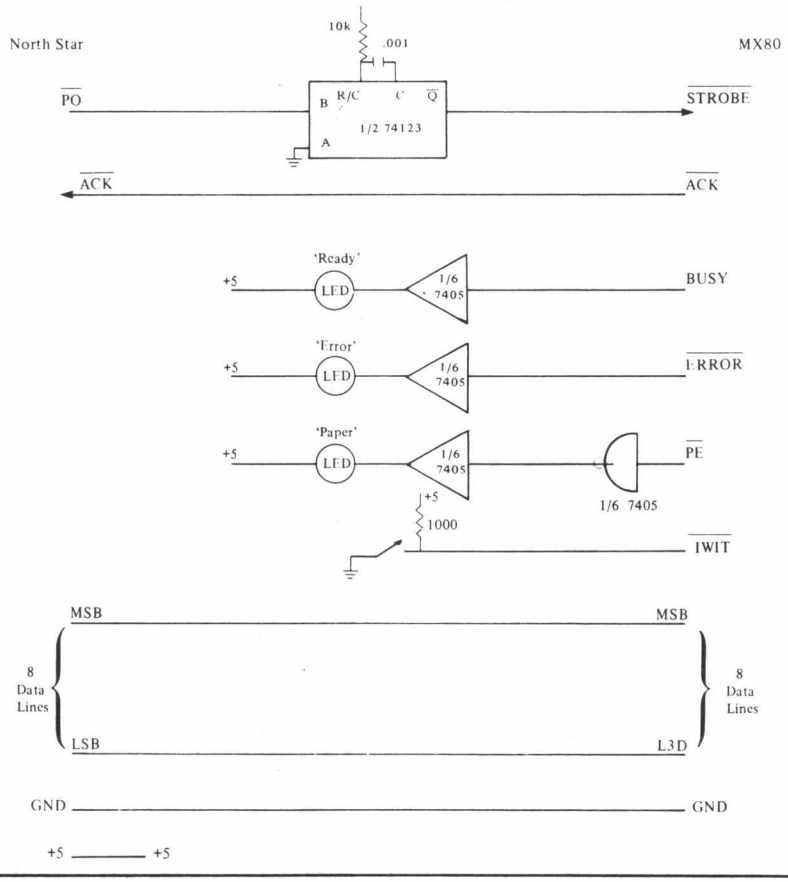
I put the circuitry necessary to generate the output strobe in a small utility box with some status indicators and a "RESET" button. The reset button is a handy way of resetting the "TOP OF FORM" status of the printer without powering it off and on. Since power is needed for the circuitry, I cut one of the three ground connections to the parallel interface connector and supplied 5 volts from the mother board instead.

The complete circuit diagram is attached. It is very simple and probably needs no explanation. The program included is one that I use to set the printer mode of operation as well as exchange the "PRINT" and "PUNCH" functions of my CP/M system. This allows me to use either the serial or parallel port for PRINT functions without changing any applications. The IOBYTE would serve the same purpose if it was implemented. The program prompts the console for various combinations of print modes and sets the printer TAB stops to the CP/M standard.

I hope this note will make it easier for others to connect parallel printers to their North Star computers. DDJ


(LISTING ON PAGE 38)


OUTBOARD MX80 INTERFACE for NORTH STAR



# NorthStar Parallel Printer

(Text begins on page 35)

```
                        ;
                        ;       print setup program
                        ;       swaps the list and punch jump addresses
                        ;       and sets up the mx80 printer modes
                        ;
                        .8080
0005                    bdos    equ     5
0000                    boot    equu    0
0005                    list    equ     5
0009                    pstring equ     9
000A                    rstring equ     10
001B                    esc     equ     27
000D                    cr      equ     0dh
000A                    lf      equ     0ah
0000'                   start:
0000'  11 00FE'                 lxi     d,aswap         ; ask about swap
0003'  0E 09                    mvi     c,pstring
0005'  CD 0005                  call    bdos
0008'  CD 01AC'                 call    answer
000B'  C2 0029'                 jnz     askenh          ; no next question
                        ;
000E'  2A 0001                  lhld    boot+1          ; get jump table address
0011'  16 00                    mvi     d,0
0013'  1E 0D                    mvi     e,13            ; this is where list address is
0015'  19                       dad     d
0016'  4E                       mov     c,m             ; get address
0017'  23                       inx     h
0018'  46                       mov     b,m             ; into bc
0019'  23                       inx     h               ; skip c3
001A'  23                       inx     h               ; now at punch address
001B'  7E                       mov     a,m             ; pick up punch address
001C'  71                       mov     m,c             ; put in list address
001D'  4F                       mov     c,a             ; and save
001E'  23                       inx     h
001F'  7E                       mov     a,m             ; rest of punch address
0020'  70                       mov     m,b
0021'  47                       mov     b,a
0022'  2A 0001                  lhld    boot+1
0025'  19                       dad     d               ; point to list address again
0026'  71                       mov     m,c             ; put in punch address
0027'  23                       inx     h
0028'  70                       mov     m,b             ; finish swap
                        ;
                        ;       now see what setup
                        ;
0029'                   askenh:
0029'  21 0136'                 lxi     h,nenhan        ; turn off enhanced mode
002C'  CD 019C'                 call    prout
002F'  11 009A'                 lxi     d,menhan        ; enhanced mode?
0032'  0E 09                    mvi     c,pstring
0034'  CD 0005                  call    bdos            ; prompt
0037'  CD 01AC'                 call    answer          ; get answer
003A'  C2 0043'                 jnz     askemp
003D'  21 0146'                 lxi     h,enhan
0040'  CD 019C'                 call    prout           ; set up printer
0043'                   askemp:
0043'  21 0139'                 lxi     h,nempha        ; turn off emphasized mode
0046'  CD 019C'                 call    prout
0049'  11 00B2'                 lxi     d,mempha
004C'  0E 09                    mvi     c,pstring
004E'  CD 0005                  call    bdos            ; prompt
0051'  CD 01AC'                 call    answer
0054'  C2 005D'                 jnz     askcond
0057'  21 0149'                 lxi     h,empha
005A'  CD 019C'                 call    prout
005D'                   askcond:
005D'  21 013C'                 lxi     h,ncond
0060'  CD 019C'                 call    prout           ; turn off condensed mode
0063'  11 00CC'                 lxi     d,mcond
0066'  0E 09                    mvi     c,pstring
0068'  CD 0005                  call    bdos
006B'  CD 01AC'                 call    answer
006E'  C2 0077'                 jnz     askpap
0071'  21 0144'                 lxi     h,cond
0074'  CD 019C'                 call    prout
0077'                   askpap:
0077'  21 0141'                 lxi     h,paper         ; set paper warning
007A'  CD 019C'                 call    prout
007D'  11 00DE'                 lxi     d,mpaper        ; ask
0080'  0E 09                    mvi     c,pstring
0082'  CD 0005                  call    bdos
0085'  CD 01AC'                 call    answer
0088'  C2 0091'                 jnz     settab
008B'  21 013E'                 lxi     h,npaper        ; reset paper warning
008E'  CD 019C'                 call    prout
0091'                   settab:
0091'  21 0123'                 lxi     h,tabs
0094'  CD 019C'                 call    prout
0097'  C3 0000                  jmp     boot
                        ;
009A'  0D 0A 64 6F      menhan: db      cr,lf,"double vertical? y/n $"
009E'  75 62 6C 65
00A2'  20 76 65 72
00A6'  74 69 63 61
00AA'  6C 3F 20 79
00AE'  2F 6E 20 24
00B2'  0D 0A 64 6F      mempha: db      cr,lf,"double horizontal? y/n $"
00B6'  75 62 6C 65
00BA'  20 68 6F 72
00BE'  69 7A 6F 6E
00C2'  74 61 6C 3F
00C6'  20 79 2F 6E
00CA'  20 24
00CC'  0D 0A 63 6F      mcond:  db      cr,lf,"condensed? y/n $"
00D0'  6E 64 65 6E
00D4'  73 65 64 3F
00D8'  20 79 2F 6E
00DC'  20 24
00DE'  0D 0A 73 75      mpaper: db      cr,lf,"suppress paper out bell? y/n $"
00E2'  70 70 72 65
00E6'  73 73 20 70
00EA'  61 70 65 72
00EE'  20 6F 75 74
00F2'  20 62 65 6C
00F6'  6C 3F 20 79
00FA'  2F 6E 20 24
00FE'  0D 0A 73 77      mswap:  db      cr,lf,"swap list and punch function? y/n $"
0102'  61 70 20 6C
0106'  69 73 74 20
010A'  61 6E 64 20
010E'  70 75 6E 63
0112'  68 20 66 75
0116'  6E 63 74 69
011A'  6F 6E 3F 20
011E'  79 2F 6E 20
0122'  24
0123'  1B 44 08 10      tabs:   db      esc,"D",8,16,24,32,40,48,56,64,72,80,88,96,114,122,130,00,"$"
0127'  18 20 28 30
012B'  38 40 48 50
012F'  58 60 72 7A
0133'  82 00 24
0136'  1B 48 24         nenhan: db      esc,"H$"
0139'  1B 46 24         nempha: db      esc,"F$"
013C'  12 24            ncond:  db      18,"$"
013E'  1B 38 24         npaper: db      esc,"8$"
0141'  1B 39 24         paper:  db      esc,"9$"
0144'  0F 24            cond:   db      0fh,"$"
                        ;
0146'  1B 47 24         enhan:  db      esc,"G","$"
0149'  1B 45 24         empha:  db      esc,"E","$"
                        ;
014C'                   input:  ds      80
                        ;
019C'                   prout:
019C'  5E                       mov     e,m
019D'  7E                       mov     a,m             ; get byte
019E'  0E 05                    mvi     c,list          ; to list device
01A0'  FE 24                    cpi     "$"
01A2'  C8                       rz                      ; done
01A3'  E5                       push    h
01A4'  CD 0005                  call    bdos
01A7'  E1                       pop     h
01A8'  23                       inx     h               ; next character
01A9'  C3 019C'                 jmp     prout
                        ;
01AC'                   answer:
01AC'  11 014C'                 lxi     d,input         ; place for answer
01AF'  3E 50                    mvi     a,80            ; maximum length
01B1'  32 014C'                 sta     input           ; set up for 'read'
01B4'  32 014E'                 sta     input+2         ; make sure answer is not "y"
01B7'  0E 0A                    mvi     c,rstring       ; read string function
01B9'  CD 0005                  call    bdos
01BC'  3A 014E'                 lda     input+2         ; get answer or 80
01BF'  FE 79                    cpi     "y"
01C1'  C8                       rz
01C2'  FE 59                    cpi     "Y"
01C4'  C9                       ret
                        ;


     MACRO-80 3.4   01-Dec-80       PAGE    1-3


                                    end     start
```

**End Listing**

# Small-C Expression Analyzer

## Patching the Address Arithmetic

**by J. E. Hendrix**

*J. E. Hendrix, Rt. 1, Box 74-B-1, Oxford, MS 38655.*

For those who are already using small-c, here are some patches to the expression analyzer. In its original form, small-c doesn't always properly scale values involved in address arithmetic. The following list describes its shortcomings:

1. Pointer values and array names should receive identical treatment. They can be used interchangeably, except that an array name, being a constant, cannot be used as an lvalue. In particular, values added to an array name should be properly scaled according to the type of the array, just as in the case of pointer arithmetic. The expression *(intarray+3) should refer to the fourth integer in the array. Small-c properly scales values added to and subtracted from pointers, but not values added to array names. It is meaningless to subtract a value from an array name.

2. There is no requirement in "C" that expressions such as (pointer+integer) be written with the pointer on the left side of the operator. Small-c, however, will scale "integer" only if the expression is written in that form.

3. Standard "C" permits address expressions to have more than two terms. That means that the expression analyzer should keep track of the identity (address or object) of intermediate results. Displacing an address (pointer value or array name) by an integral value should yield a new address. Subtracting one address from another of the same type should yield the integral number of objects by which they are displaced. All other arithmetic operations involving addresses are meaningless. The expression (address1-address2) has meaning only when both addresses refer to objects in the same array. Small-c makes no attempt to track the identity of intermediate results and, therefore, does not always properly scale values added to or subtracted from intermediate values.

The revisions needed to correct these problems are too extensive and fragmented to describe in detail, so I am including a complete listing of the revised expression analyzer. All new code is enclosed in boxes. I leave it to the reader to work out the theory.

My complements to Ron Cain for his work in producing a great little compiler. After using it for a year, I am still enthusiastic about it. We owe Ron a debt of gratitude for placing it in the public domain.

DDJ

While your manuscript may be too technical for some other micro publications, it may be "just what the doctor ordered" for our audience. Let us take a look at it, won't you? You can reach us at:

**Dr. Dobb's Journal**
**P.O. Box E**
**Menlo Park, CA 94025**
**(415) 323-3111**

# Small-C Expression Analyzer (Text begins on page 40)

```
/*
** lval[0] - symbol table address, else 0 for constant
** lval[1] - type indirect object to fetch, else 0 for static object
** lval[2] - type pointer or array, else 0
*/
expression()
  {
  int lval[3];
  if(heir1(lval))rvalue(lval);
  }
heir1(lval)
  int lval[];
  {
  int k,lval2[3];
  k=heir2(lval);
  if (match("="))
    {if(k==0){needlval();return 0;}
    if (lval[1])push();
    if(heir1(lval2))rvalue(lval2);
    store(lval);
    return 0;
    }
  else return k;
  }
heir2(lval)
  int lval[];
  {int k,lval2[3];
  k=heir3(lval);
  blanks();
  if(ch()!='|')return k;
  if(k)rvalue(lval);
  while(1)
    {if (match("|"))
      {push();
      if(heir3(lval2)) rvalue(lval2);
      pop();
      or();
      }
    else return 0;
    }
  }
heir3(lval)
  int lval[];
  {int k,lval2[3];
  k=heir4(lval);
  blanks();
  if(ch()!='^')return k;
  if(k)rvalue(lval);
  while(1)
    {if (match("^"))
      {push();
      if(heir4(lval2))rvalue(lval2);
      pop();
      xor();
      }
    else return 0;
    }
  }

heir4(lval)
  int lval[];
  {int k,lval2[3];
  k=heir5(lval);
  blanks();
  if(ch()!='&')return k;
  if(k)rvalue(lval);
  while(1)
    {if (match("&"))
      {push();
      if(heir5(lval2))rvalue(lval2);
      pop();
      and();
      }
    else return 0;
    }
  }
heir5(lval)
  int lval[];
  {
  int k,lval2[3];
  k=heir6(lval);
  blanks();
  if((streq(line+lptr,"==")==0)&
    (streq(line+lptr,"!=")==))return k;
  if(k)rvalue(lval);
  while(1)
    {if (match("=="))
      {push();
      if(heir6(lval2))rvalue(lval2);
      pop();
      eq();
      }
    else if (match("!="))
      {push();
      if(heir6(lval2))rvalue(lval2);
      pop();
      ne();
      }

    else return 0;
    }
  }
heir6(lval)
  int lval[];
  {
  int k,lval2[3];
  k=heir7(lval);
  blanks();
  if((streq(line+lptr,"<")==0)&
    (streq(line+lptr,">")==0)&
    (streq(line+lptr,"<=")==0)&
    (streq(line+lptr,">=")==0))return k;
    if(streq(line+lptr,">>"))return k;
    if(streq(line+lptr,"<<"))return k;
  if(k)rvalue(lval);
  while(1)
    {if (match("<="))
      {push();

      if(heir7(lval2))rvalue(lval2);
      pop();
      if(lval[2] | lval2[2])
        {ule();
        continue;
        }
      le();
      }
    else if (match(">="))
      {push();
      if(heir7(lval2))rvalue(lval2);
      pop();
      if(lval[2] | lval2[2])
        {uge();
        continue;
        }
      ge();
      }
    else if((streq(line+lptr,"<"))&
      (streq(line+lptr,"<<")==0))
      {inbyte();
      push();
      if(heir7(lval2))rvalue(lval2);
      pop();
      if(lval[2] | lval2[2])
        {ult();
        continue;
        }
      lt();
      }
    else if((streq(line+lptr,">"))&
      (streq(line+lptr,">>")==0))
      {inbyte();
      push();
      if(heir7(lval2))rvalue(lval2);
      pop();
      if(lval[2] | lval2[2])
        {ugt();
        continue;
        }
      gt();
      }
    else return 0;
    }
  }
heir7(lval)
  int lval[];
  {
  int k,lval2[3];
  k=heir8(lval);
  blanks();
  if((streq(line+lptr,">>")==0)&
    (streq(line+lptr,"<<")==0))return k;
  if(k)rvalue(lval);
  while(1)
    {if (match(">>"))
      {push();
      if(heir8(lval2))rvalue(lval2);
```

*(Listing continued on next page)*

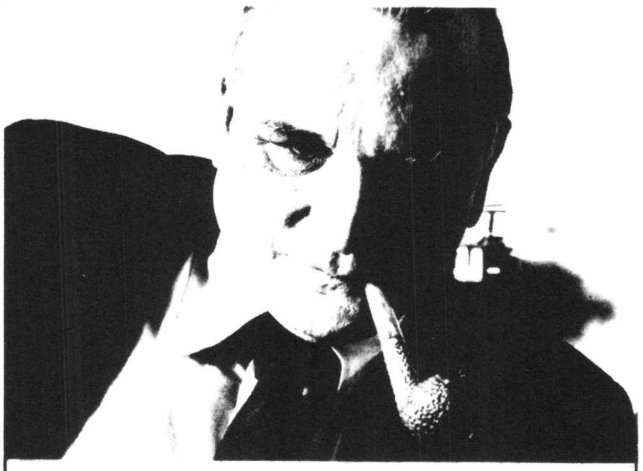
# Small-C Expression Analyzer (Listing continued, text begins on page 40)

```
175       pop();
176       asr();
177       }
178     else if (match("<<"))
179       (push();
180       if(heir8(lval2))rvalue(lval2);
181       pop();
182       asl();
183       }
184     else return 0;
185     }
186   )
187 heir8(lval)
188   int lval[];
189   {
190   int k,lval2[3];
191   k=heir9(lval);
192   blanks();
193   if((ch()!='+')&(ch()!='-'))return k;
194   if(k)rvalue(lval);
195   while(1)
196     (if (match("+"))
197       (push();
198       if(heir9(lval2))rvalue(lval2);
199       if(dbltest(lval,lval2))
200         doublereg();
201       pop();
202       if(dbltest(lval2,lval)) {
203         swap();
204         doublereg();
205         swap();
206         }
207       add();
208       result(lval, lval2);
209       }
210     else if (match("-"))
211       (push();
212       if(heir9(lval2))rvalue(lval2);
213       if(dbltest(lval,lval2))
214         doublereg();
215       pop();
216       if(dbltest(lval2,lval)) {
217         swap();
218         doublereg();
219         swap();
220         }
221       sub();
222       if((lval[2]==cint)&(lval2[2]==cint)) {
223         swap();
224         immed();ol("1");
225         asr();  /** div by 2 **/
226         }
227       result(lval, lval2);
228       }
229     else return 0;
230     }
231   )
232 heir9(lval)

233   int lval[];
234   {
235   int k,lval2[3];
236   k=heir10(lval);
237   blanks();
238   if((ch()!='*')&(ch()!='/')&
239     (ch()!='%'))return k;
240   if(k)rvalue(lval);
241   while(1)
242     (if (match("*"))
243       (push();
244       if(heir9(lval2))rvalue(lval2);
245       pop();
246       mult();
247       }
248     else if (match("/"))
249       (push();
250       if(heir10(lval2))rvalue(lval2);
251       pop();
252       div();
253       }
254     else if (match("%"))
255       (push();
256       if(heir10(lval2))rvalue(lval2);
257       pop();
258       mod();
259       }
260     else return 0;
261     }
262   )
263 heir10(lval)
264   int lval[];
265   {
266   int k;
267   char *ptr;
268   if(match("++"))
269     (if((k=heir10(lval))==0)
270       (needlval();
271       return 0;
272       }
273     if(lval[1])push();
274     rvalue(lval);
275     inc();
276     if(lval[2]==cint)
277       inc();
278     store(lval);
279     return 0;
280     }
281   else if(match("--"))
282     (if((k=heir10(lval))==0)
283       (needlval();
284       return 0;
285       }

286     if(lval[1])push();
287     rvalue(lval);
288     dec();
289     if(lval[2]==cint)
290       dec();

291     store(lval);
292     return 0;
293     }
294   else if (match("-"))
295     (k=heir10(lval);
296     if (k) rvalue(lval);
297     neg();
298     return 0;
299     }

300   else if(match("*"))
301     (k=heir10(lval);
302     if(k)rvalue(lval);
303     if(ptr=lval[0])lval[1]=ptr[type];
304     else lval[1]=cint;
305     lval[2]=0;  /**** flag as not pointer or array ****/
306     return 1;
307     }
308   else if(match("&"))
309     (k=heir10(lval);
310     if(k==0)
311       (error("illegal address");
312       return 0;
313       }
314     ptr=lval[0];
315     lval[2]=ptr[type];
316     if(lval[1]) return 0;
317     /* global & non-array */
318     immed();
319     outstr(ptr);
320     nl();
321     lval[1]=ptr[type];
322     return 0;
323     }
324   else
325     (k=heir11(lval);
326     if(match("++"))
327       (if(k==0)
328         (needlval();
329         return 0;
330         }
331       if(lval[1])push();
332       rvalue(lval);
333       inc();
334       if(lval[2]==cint)
335         inc();
336       store(lval);
337       dec();
338       if(lval[2]==cint)
339         dec();
340       return 0;
341       }
342     else if(match("--"))
343       (if(k==0)
344         (needlval();
345         return 0;
346         }
347       if(lval[1])push();
348       rvalue(lval);

349       dec();
350       if(lval[2]==cint)
351         dec();
352       store(lval);
```

```
353       inc();
354       if(lval[2]==cint)
355         inc();
356       return 0;
357       }
358     else return k;
359     }
360   }
361 heir11(lval)
362   int *lval;
363   {int k;char *ptr;
364   k=primary(lval);
365   ptr=lval[0];
366   blanks();
367   if((ch()=='[')|(ch()=='('))
368   while(1)
369     {if(match("["))
370       {if(ptr==0)
371         {error("can't subscript");
372         junk();
373         needbrack("]");
374         return 0;
375         }
376       else if(ptr[ident]==pointer)rvalue(lval);
377       else if(ptr[ident]!=array)
378         {error("can't subscript");
379         k=0;
380         }
381       push();
382       expression();
383       needbrack("]");
384       if(ptr[type]==cint)doublereg();
385       pop();
386       add();
387       lval[0]=0;
388       lval[1]=ptr[type];
389       k=1;
390       }
391     else if(match("("))
392       {if(ptr==0)
393         {callfunction(0);
394         }
395       else if(ptr[ident]!=function)
396         {rvalue(lval);
397         callfunction(0);
398         }
399       else callfunction(ptr);
400       k=lval[0]=0;
401       }
402     else return k;
403     }
404   if(ptr==0)return k;
405   if(ptr[ident]==function)
406     {immed();

407     outstr(ptr);
408     nl();
409     return 0;
410     }
411   return k;
412   }
413 primary(lval)
414   int *lval;
415   {char *ptr,sname[namesize];int num[1];
416   int k;
417   lval[2]=0;  /**** clear pointer/array type ****/
418   if(match("("))
419     {k=heir1(lval);
420     needbrack(")");
421     return k;
422     }
423   if(symname(sname))
424     {if(ptr=findloc(sname))
425       {getloc(ptr);
426       lval[0]=ptr;
427       lval[1]=ptr[type];
428       if(ptr[ident]==pointer) {
429         lval[1]=cint;
430         lval[2]=ptr[type];
431         }
432       if(ptr[ident]==array) {
433         lval[2]=ptr[type];
434         return 0;
435         }
436       else return 1;
437       }
438     if(ptr=findglb(sname))
439       if(ptr[ident]!=function)
440       {lval[0]=ptr;
441       lval[1]=0;
442       if(ptr[ident]!=array) {
443         if(ptr[ident]==pointer) lval[2]=ptr[type];
444         return 1;
445         }
446       immed();

447       outstr(ptr);nl();
448       lval[1]=lval[2]=ptr[type];
449       return 0;
450       }
451     ptr=addglb(sname,function,cint,0);
452     lval[0]=ptr;
453     lval[1]=0;
454     return 0;
455     }
456   if(constant(num))
457     return(lval[0]=lval[1]=0);
458   else

459     {error("invalid expression");
460     immed();outdec(0);nl();
461     junk();
462     return 0;
463     }
464   }
```

```
465 /*
466 ** true if val1 -> int pointer or int array and val2 not ptr or array
467 */
468 dbltest(val1,val2) int val1[], val2[]; {
469   if(val1[2]!=cint) return 0;
470   if(val2[2]) return 0;
471   return 1;
472   }
473 /*
474 ** determine type of binary operation
475 */
476 result(lval, lval2) int lval[], lval2[]; {
477   if(lval[2] & lval2[2]) {
478     lval[2]=0;
479     }
480   else if(lval2[2]) {
481     lval[0]=lval2[0];
482     lval[1]=lval2[1];
483     lval[2]=lval2[2];
484     }
485   }
```

**End Listing**

**Dr. Robert Smithdas is blind, nonhearing, and nonverbal. He is a published poet.**

President's Committee on
Employment of the Handicapped
Washington, D.C. 20210

Produced by The School of Visual Arts Public Advertising System

# Auto. Loading of Assembly Language Subroutines

**by John B. Palmer**

*John B. Palmer, P. O. Box 23, Boonville, CA 95415.*

Have you ever wanted a simple way to relocate an 8080 assembly code program into the top of the TPA in CP/M? Have you ever written a BASIC program that requires some machine code hidden in dumb DATA statements? There is a better way.

Why would anybody want to use assembly code and BASIC together to solve a problem? Well, there really are a few places where that seems to be the right approach. Let me give an example based on a real experience.

Let's imagine that you and I have been using BASIC and CP/M for some time on a project. Everything goes well until we wish to interface BASIC to a new analog-to-digital converter. At first we try to read the gadget with the PEEK function in BASIC. We find that some readings are not always true. The BASIC interpreter is too slow.

So we write machine code and place it in DATA statements, use POKE to put it into memory, and use CALL to transfer control to the machine code which in turn does a RET to BASIC. Later we need to make a minor change and have a lot of trouble understanding what those DATA statements mean. We conclude that while BASIC may be quick and easy for many jobs, BASIC does not provide good documentation when machine code is written into a group of DATA statements.

The better way is to use the utilities that come with CP/M. The ED program we shall use to edit a well-documented assembly language program. The ASM program will assemble the program and may report errors.

Once the program has been correctly assembled, it will take two or three steps to get both BASIC and the machine code running together. First we use the DDT program to load the resulting HEX file into the upper part of the TPA. After that we reboot and execute BASIC with a command line that limits the memory size and then give the commands to load and run the program in BASIC.

The only objection to this is the number of steps it takes to finally get into our BASIC program. If we make a mistake we may have to start all over again. Is it possible to write a simple COM file that will do everything? Yes, it is.

Quite a while back in *DDJ* (Vol. 3, No. 2) there was an article by Gary Kildall that explained how to write self-relocating code. By using this trick we can bypass DDT and have our code load itself in the upper part of the TPA.

We shall want to create a Submit file on the A drive that will start BASIC with the right memory size and load our BASIC program right away. Figure 1 is a partial example of how this could be done.

Briefly, here are the steps we need to take in order to create our COM file that will do everything we need with just one simple command. (We assume that the machine code to interface BASIC to our new toy is less than 256 bytes.)

Step 1. Write a memory move program that will move a block of code that starts at 0200h into a destination in the upper TPA. As each byte is moved,

Our high-quality editorial
and
your high-quality products
are made for each other!
Call (415) 323-3111 to reserve your space
in the next *Dr. Dobb's Journal*.

check to see if its contents are equal to the byte in the same relative place in a comparison block that starts at 0300 hex. If they are not equal, the byte is the high-order byte of an address; correct the address in the destination.

Step 2. Combine the memory move with the program in Figure 1 so that after the machine code is put in place a file called "$$$.SUB" is written onto the A drive.

Step 3. Assemble two copies of the desired BASIC-to-gadget machine code. Originate one copy at 0200 hex. Originate the other at 0300 hex to be the comparison block.

Step 4. Write a text file of type .SUB that has the proper command line to invoke BASIC, set memory size, and load and execute our BASIC program. It should be a single line something like this: BASIC ANALOG/M&H9000. Then try to execute it with the SUBMIT utility on the B drive. We get a file "$$$.SUB" on the B drive.

Step 5. Invoke DDT. Read in the memory mover HEX file. Read in the HEX code that originates at 0200. Read in the HEX code that originates at 0300. Read in the $$$.SUB file with offset so that it lands at 0400. Reboot. SAVE 4 Widget.COM.

Now then, anytime we give the simple command WIDGET the machine code will move into the upper TPA and BASIC will automatically be invoked with the proper command line.

This may seem to be overly complicated. Perhaps it is. But the steps outlined above need only be done once, and can themselves be made into a Submit file.

Now let's review what objectives have been met.

1. The end user can still use his BASIC interactive interpreter. No need to teach him a new language.
2. We have avoided the distasteful practice of placing a machine language program into a group of DATA statements.
3. The assembly language program can be edited and altered with very little impact on the main application program written in BASIC.
4. No new software was purchased. Everything was done with the existing interpreter and system utilities.

I hope this may be of some help to others who have had to face a similar problem with their microcomputer operations. ▶▶J

**Figure I.**

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*   MAKE A AUTO EXECUTE FILE ON THE 'A' DRIVE            *
;*   USING 0400H AS THE BLOCK OF CODE TO BE               *
;*   WRITTEN ONTO DISK AS '$$$.SUB'                       *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
CR      EQU     0DH
LF      EQU     0AH
MAKAUTO:
        MVI     C,26    ; SET DMA
        LXI     D,0400H
        CALL    05
        MVI     C,22    ; MAKE FILE
        LXI     D,FCB
        CALL    05
        INR     A       ; TEST FOR 0FFH
        JZ      ERROR
        MVI     C,21    ; WRITE
        LXI     D,FCB
        CALL    05
        MVI     C,16    ; CLOSE
        LXI     D,FCB
        CALL    05
        MVI     C,9
        LXI     D,WAITMESS
        CALL    05
        JMP     0       ; REBOOT CPM
WAITMESS:
 DB CR,LF
 DB 'PLEASE WAIT ONE MOMENT...
 DB CR,LF
 DB 'YOUR PROGRAM IS NOW LOADING...
 DB CR,LF,'$'
ERROR:
        MVI     C,9
        LXI     D,ERRMESS
        CALL    05
        JMP     0
ERRMESS:
 DB 'SOMETHING IS WRONG $'
FCB:
        DB      1       ; DRIVE A
        DB '$$$         ; NAME
        DB 'SUB'        ; TYPE
        DB 0,0,0,0,0
        DB 0,0,0,0,0,0,0,0,0
        DB 0,0,0,0,0,0,0,0,0
        DB 0            ; 33 BYTES
        END
```

# Index

# Index


2716 EPROM programmer [M-80] 189-190
6502
  Assembler for the 6502 369-373
  Byte-count routine 83
  Opcodes analysis [BASIC/Pet] 96-97
  Reader response 185
6809
  Multi-byte opcodes 279
  Tiny BASIC for the 6809, Santa Barbara 202-203
8080 (also see Z-80) 187
  Memory chip tester 533-534
  North Star DOSMOVER 124
  Z-80
    Cross assembler for 8080/Z-80 230-234
    Editor, assembler, disassembler in XL package 36-40
Adams, E. 514
Adven-80, adventure development system 468-473
Airy, B. 6, 7, 11
Algorithm to convert integer date [MP/M] 201
Algorithms [FORTH] 148-149
Apple II
  File system, Tiny 240-241
Arrays, cassette I/O routine [BASIC] 150-152
Artificial intelligence 419
  "Thinking" program, ERGO 100
Artificial intelligence language, LISP 526-530
Assemblers
  Assembler for the 6502 [FORTH] 369-373
  Cross assembler for 8080/Z-80 230-234
  Editor, assembler, disassembler in XL package 36-40
  Editor/Assembler Plus 123
BASIC
  Arrays, cassette I/O routine 150-152
  BASIC-FORTH 427-428
  MIN and MAX definition 312
  North Star
    Debugging concatenated programs 64
    North Star DOSMOVER 124
    Small VM development, portable software 492-495
    Terminal programs for North Star 487
    Text editor input routine 42-43
    Time keeping in microcomputers 287-289
  PET
    Analysis 6502 opcodes 96-97
    Speed reading using a computer 95
  POKE and PEEK, BASIC version 3.2 enhancement 62
  Polynomial evaluation 146-147
  Rubik's cube simulator 291-292
  Time delay circuit design 197
  Tiny BASIC for the 6809, Santa Barbara 202-203
    Bugs 419
  ZX80 Sinclair 60-61
BASIC-FORTH [FORTH] 427-428
BASIC-Plus 45
Bassett, S. 148
Binary to decimal conversion routine 271
  Readers' improvements 422
Blessing, R. 487
Bolton, B. 330
Brodgen, W. 237
Byte-count routine [6502] 83
Byte manipulation and pointers [Fortran] 103
C language (see also: Small-C compiler)
  Time functions 330-331
Cage, T. 124
Calendar routines 313, 516
Card shuffling 356
Caulkins, D. 63, 95
CBASIC-2
  DB.BAS package file maintenance 80
CLOAD Magazine, Inc. 122
COMAL-80 [CP/M] 235-236
  As an interpreter 235
Compal-80 102
Compiler writing tool, META4 321
  Implementing an M4 interpreter 329
  M4 language 325
  META4 language 322
Compilers
  Small-C compiler
    Error run-time library routine 53
  Small-C compiler bug fixes and enhancement 228
  Small-C cross-compiler implementation 66-67
  Small-C programming 504-507
  Tincmp, compiler for Pidgin 284-286
Computer conferencing 429-431
Conversion
  Binary to decimal conversion routine 271
  File conversion, H8 text editor to CP/M 239
  Submit file comments 200, 415
Cortesi, D. 200, 271, 312, 414, 422, 520
CP/M
  COMAL-80 235-236
  DDT, circumventing 200
  DUMP utility readability enhancement 93
  File conversion, H8 text editor to CP/M 239
  Keyboard input definition 194-196
  SCRIBE IX, word processing package 220-222
  Submit file comments 200
  Text editor WORD-MASTER with the Compal-80 102
  ZDM debugger 186
Cross assembler for 8080/Z-80 [8080/Z-80] 230-234
  XASM-86 features 232
  Z-88 project 230
Database design [FORTH] 398-401
Davis, D. 126
DB.BAS package file maintenance [CBASIC-2] 80
DDT, circumventing [CP/M] 200

Debuggers
DDT, circumventing 200
ZDM debugger for CP/M users 80-81
Debugging concatenated programs [BASIC/North Star] 64
Decompiler [FORTH] 403
Demas, J. 36
Dobyns, B. 178
DUMP utility readability enhancement [CP/M] 93
Implementation 227
Duncan, R. 69,140,187,403
Editors
Editor, assembler, disassembler in XL package [8080/Z-80] 36-40
Editor/Assembler Plus [TRS-80] 123
Program editors, reassessment of 237-238
Screen editor, MINCE 178
Screen-oriented editors 383
Text editor input routine [BASIC] 42-43
Text editor WORD-MASTER with the Compal-80 102
Underline modification 126
ZX80 Sinclair 60-61
Electronic phone book [PCNET project] 63,87,133,176
Embedding wide characters within an ordinary line 515
Faulkner, D. 15
Favitta, M. 80
Fibonacci numbers sequence [FORTH] 9,91
File comparator [Z-80] 15
File conversion, H8 text editor to CP/M [conversion] 239
File system, Tiny [Apple II] 240-241
Floppy disk test [Z-80] 140-145
FORTH
Algorithms 148-149
Assembler for the 6502 369-373
BASIC-FORTH 427-428
Database design 398-401
Decompiler 403
Description 411-413
Fibonacci numbers 9
Fibonacci numbers sequence 91
FORTH as language and environment 381-382
Overview 6,12
Construction of program-like commands 8
Metamorphic changes by concatenations of characters 10
Primitives 7
Reader response 137,185
Use by novices 12
Variables stack 7
Parameterized command concept 54-59
Reader response 186
Philosophy of 363-368
Software development methodology 364
Software tools 367
Screen-oriented editor 383
String routines 51
FORTH as language and environment [FORTH] 381-382
FORTH description 411-413
Fortran
Byte manipulation and pointers 103
Small-C cross-compiler implementation 66-67
Freed, E. 103
Gale, W. 240,282,321
Gordon, H. 6,54
Greenlaw, D. 88
H8 computer 239
Hacking 510-514
Hallem, R. 149
Harris, D. 63,177
Harris, K. 363
Hatfield, J. 122
Haydon, G. 398
Head, G. 80,220
Hendrix, J. 492
Hightower, M. 127
Hinds, J. 202
Hoffman, R. 146
Hypotenuse approximation on integer values 200
Reader input 312,359,520
James, J. 429
Johnson, J. 514
Kalish, M. 122
Kanefsky, R. 514
Keyboard input definition [CP/M] 194-196
Theory 195
Knippenberg, H. 197
Kudlac, S. 514
Language register 3
Lawrence, J. 45
Laxen, H. 383
LISP (see artificial intelligence language)
Loeliger, R. 381
M-4 interpreter 329
M-4 language 325
M-80
2716 EPROM programmer 189-190
Maggs, P. 127
Mailing list label printer [SLIC] 149
Memory chip tester [8080] 533-534
Memory-mapped video displays 16
Memory test [Z-80] 69
Meta 4 (see compiler writing tool)
Miller, E. 189
Mills, J. 230
MIN and MAX definition [BASIC] 312
Extended to other functions 354
MINCE, screen editor 178
Monitor MON 3 [TRS-80] 122
MP/M
Algorithm to convert integer date 201
Nicol, R. 123,526
N-Logs 53,281
North Star (see BASIC)
North Star DOS 36,270
North Star DOSMOVER [8080] 124
Terminal programs for North Star 487
Palmer, J. 60,194
PAN protocol revision [PCNET Project] 63
Parameterized command concept [FORTH] 54-59
Access 55
Conditional-jump primitive 57
Data structures 55
Notation 54-55
Reader response [FORTH] 186
Status-register 56
Pascal
Card shuffling 356
"Upside" down, nature of 4
Patches
POKE and PEEK, BASIC version 3.2 enhancement 62
PCNET project 176
Electric phone book 63,87,133,176
PAN protocol revision 63
PCNET protocol 63,177
Pehoushek, J. 291
Pelle, M. 235
Personal computer networks (see PCNET)
Peter, K. 514
Philosophy of FORTH [FORTH] 363-368
PHIMON subroutines 15
Pidgin 321
Tincmp, compiler for Pidgin 284-286

Pidgin, systems programming languages 282-284
Pocket computer, Radio Shack 88
POKE and PEEK, BASIC version 3.2 enhancement [BASIC] 62
Alternative patch 228
Polynomial evaluation [BASIC] 146-147
Portable software development, Small/VM 492-495
Printers
Printer toggle installation in North Star DOS 5.2 270
Program editors, reassessment of [editors] 237-238
Programmers
2716 EPROM programmer 189-190
Programming 237
Artificial Intelligence language, LISP 526-530
BASIC- FORTH 427-428
Pidgin, systems programming language 282-284
Small-C programming 504-507
Time functions 330-331
Programs
Adventure development system, Adven-80 468-473
Assembler for the 6502 369-373
BASIC- FORTH 427-428
COMAL-80 235-236
Compiler writing tool, META4 321
Cross assembler for 8080/Z-80 230-234
Debugging concatenated programs 64
Decompiler 403
DUMP utility readability enhancement 93
File comparator 15
File conversion, H8 text editor to CP/M 239
File system, Tiny 240-241
Floppy disk test 140-145
Keyboard input definition 194-196
Mailing list label printer 149
Memory test [Z-80] 69
North Star DOSMOVER 124
Pidgin, systems programming language 282-284
Rubik's cube simulator 291-292
Screen-oriented editor 383
Small-C programming 504-507
Speed reading using a computer 95
Terminal programs for North Star 487
"Thinking" programs, ERGO 100
Time delay circuit design 197
Time functions 330-331
Time-keeping in microcomputers 287-289
Tiny BASIC for the 6809, Santa Barbara 202-203
Turing machine simulation 45
Video drivers 16-17
Radio Shack (see TRS-80)
Radio Shack pocket computer 88
Ragsdale, W. 369
Rohde, A. 150
Roth, R. 504
Routines
Arrays, cassette I/O routine 150-152
Binary to decimal conversion routine 271
Byte manipulation and pointers 103
Calendar routines 313
Database design 398-401
DUMP utility readability enhancement 93
Polynomial evaluation 146-147
Small-C cross-compiler implementation 66-67
Text editor input routine 42-43
Text editor WORD-MASTER with the Compal-80 102
Rubik's Cube simulator [BASIC] 291-292
Rubin, S. 100
Santa Barbara (see Tiny BASIC for the 6809)
Scargill, P. 468
Scott, J. 239
Screen editor, MINCE 178
Addendum 227
Extended with Amethyst 181-182
Screen editors 178
Screen-oriented editor 102
[FORTH] 383
Scribe IX, wordprocessing package [CP/M] 220-222
Sinclair BASIC 61
Skjellum, A. 16,102,423
SLIC
Mailing list label printer 149
Small-C compiler
Bug fixes and enhancements 228
Error run-time library routines [compiler] 53
Modification 465
Small-C cross-compiler implementation [Fortran] 66-67
Bug correction 67
Bugs 228,277
Update 104
Small-C programming 504-507
Small VM development, portable software
[BASIC/North Star] 492-495
Smart, B. 270
Smith, G. 42,64
Softky, S. 100
Software reviews
CLOAD Magazine, Inc. 122
DB.BAS package file maintenance 80
Editor, assembler, disassembler in XL package 36-40
Editor/Assembler Plus 123
Monitor MON 3 122
Screen editor, MINCE 178
SCRIBE IX word processing package 220-222
Utility package, Disk Keyplus 122
ZDM debugger for CP/M users 80-81
Sonderstrup, S. 235
Speech synthesizer interface [TRS-80] 127-129
Speed reading using a computer [BASIC/Pet] 95
Using ED and PIP in CP/M 186
String routines [FORTH] 51
Submit file comments [CP/M / conversions] 200,415
Testing binary integer for single 1-bit 414
Tektronix 4010 simulator [Z-80] 423-425
Terminal programs for North Star [North Star] 487
Text editor
Input routine [BASIC/North Star] 42-43
WORD-MASTER with the Compal-80 [CP/M] 102
"Thinking" program, ERGO [artificial intelligence] 100
Documentation 319
Time-delay circuit design [BASIC] 197
Small error 420
Time functions [C language] 330-331
Time-keeping in microcomputers [BASIC/North Star] 287-289
Timin, M. 411
Tincmp, compiler for Pidgin 284-286
Ting, C.H. 427
Tiny BASIC for the 6809, Santa Barbara [6809] 202-203
TRS-80
Editor/Assembler Plus 123
Monitor MON 3 122
Snythesizer interface 127-129
Utility package, Disk Keyplus 122
Turing machine simulation [programs] 45
Underline modification 126
Universal language 45
Utility package, Disk Keyplus [TRS-80] 122
Van Iderstine, T. 62
VDM-1 124
Video drivers [Z-80] 16-17

Time-dependent coding 93
Wexler, S. 122
Whitlock, H. 96
Woods, P. 66, 104
Word processing package (see SCRIBE IX)
Yates, D. 287
Z-80 (see also 8080)
File comparator 15
Floppy disk test 140-145
Memory test 69
Tektronix 4010 simulator 423
Video drivers 16-17
Z-80 microprocessor, use and misuse 187-188
ZDM debugger for CP/M users [Z-80] 80-81
Inline assembler [CP/M] 186
ZSID bug and patch 519
ZX80, Sinclair [BASIC] 60-61
Reader response 137, 229

5493-9

# THE DR. DOBB'S SERIES

The People's Computer Company

A collection of fascinating bits of information for the home computer hobbyist. These works, from People's Computer Company, are not just "the best of . . . ," they include almost everything from all issues of *Dr. Dobb's Journal*. "There's more creativity, wild ideas, and raw enthusiasm here per page than you are likely to find anywhere else in the personal computing press." *Creative Computing*.

## DR. DOBB'S JOURNAL, Volume 1

Chronicles the advent of the microcomputer era in 1976. Always pertinent for bit crunching and byte saving, language design, home-brew computer projects, and the technical history of personal computing.

**Topics include:** Tiny BASIC, the first words on CP/M, Speech Synthesis, Floating Point Routines, 6502 Disassembler for the Apple, Timer Routines, and much more. #5475, paper, 368 pages.

## DR. DOBB'S JOURNAL, Volume 2

The small computer emerges as a powerful tool for the modern age in these issues from 1977.

**Topics include:** Lawrence Livermore Lab's BASIC, Dr. Starkweather's PILOT, Using a Modem, String Handling Techniques, Ciphers, Turtle Graphics, micro utilities, and much more. #5484, paper, 480 pages.

## DR. DOBB'S JOURNAL, Volume 3

Explores the foundations of the mass market computer industry and advances in microcomputer programming in 1978.

**Topics include:** Programming in BASIC, PILOT, and Pascal, RAM Memory Testers, Apple utility programs, S-100 Bus Standard, STRUBAL: A Structured BASIC Compiler, and much more. #5490, paper, 480 pages.

## DR. DOBB'S JOURNAL, Volume 4

Heralds the widespread acceptance of the microcomputer in America. Innovative ideas and articles guide the reader through the age of discovery in 1979.

**Topics include:** Programming problems for the popular micros, Selecting Business Software, Microcomputer Speech and Music, Printers and Plotters, Information Networks, interfacing techniques, and much more. #5491, paper, 480 pages.

## DR. DOBB'S JOURNAL, Volume 5

Focuses on the technological promise of the modern microcomputer and the creative challenges facing programmers in 1980.

**Topics include:** The revolutionary impact of CP/M, C programming and the UNIX operating system, survey of computer networks, graphics, software portability, introduction to FORTH, compiler writing, and much more. #5492, paper, 464 pages.

## DR. DOBB'S JOURNAL, Volume 6

The microcomputer enters the mainstream — these issues assess the revolutionary impact of microcomputers on the individual and on our society.

**Topics include:** Computer Conferencing, The Power of FORTH, 8- and 16-Bit Technology, Rubik's Cube Simulator, An Adventure Game Development System, and much more. #5493, paper, 576 pages.

Look for these volumes of *Dr. Dobb's* at your local computer store and bookstore. All volumes are also available from Hayden Book Company.



HAYDEN BOOK COMPANY, INC.
Hasbrouck Heights, New Jersey

ISBN 0-8104-5493-9