Dr. Dobb's Journal of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

A Reference Journal for Users of Home Computers

Volume Four

The People's Computer Company Series

Dr. Dobb's Journal

of Computer Calisthenics & Orthodontia,

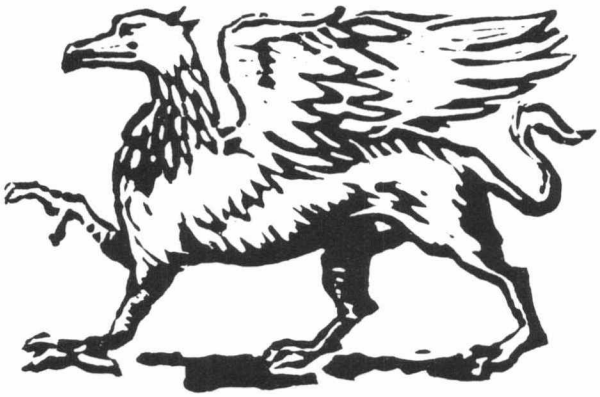
Volume Four

Dr. Dobb's Journal of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

A Reference Journal for Users of Home Computers

Volume Four



The People's Computer Company Series

HAYDEN BOOK COMPANY, INC.
Rochelle Park, 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, had their creators not felt that immersion in the critical caldron 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, coffee house synergisms, and occasional moments of genius. These Friends keep us sane, and make microcomputing human.

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

Book production: Wendy Palmer and Associates,
Palo Alto, California

Index by Julie Anton, Palo Alto, California

Full-page computer art by David Dameron

ISBN 0-8104-5491-2

Copyright © 1982 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

1 2 3 4 5 6 7 8 9 PRINTING

82 83 84 85 86 87 88 89 90 YEAR

Editor's Preface

And where will *it lead?*

That was the question posed in *Dr. Dobb's Journal,* Volume III. It was a time when a body of microprocessor knowledge had been accumulated, but the number of persons with access to and understanding of that knowledge was extremely small. Manufacturing techniques were being perfected, but costs were still high. The bandwagon was rolling, but no one was holding the reins.

At least, that's the way it appeared.

Dr. Dobb's Journal was born in a time of chaos. Some fantastic new machines had been bred in basement test tubes. The problem was how to optimize their potential? Then a handful of radical-thinking idealists produced Tiny BASIC and a publication in which to communicate the good news. More good things have been cropping up since then, and now the magazine serves as a broad forum for the micro community.

To answer the original question, the revolution in microcomputing will lead wherever we direct it. In the beginning, when there was very little momentum to the personal computing movement, a handful of movers and shakers could do most of the guiding. They saw to it that the power of small computers would indeed rest in the hands of individuals instead of with corporations or government.

The movement has, of course, gained gigantic proportions as more and more people have begun to grasp its significance and to reap its benefits. With its present inertial force, many more guiding hands are needed to keep it on track, to preserve the original dream. *Dr. Dobb's Journal* has kept pace by opening its pages to new authors, more free thinkers and guiding lights.

The proliferation of hardware and software has required that we grow to meet the challenge presented by our own idealism. In order to keep alive the concept of personal computing power, we must not only keep stride with technology, we must shape the future. We must be in the forefront, developing systems tools and refining the programming languages, tecnniques and very concepts which will influence what small computers will be able to do, and for whom.

Lest we become too grandiose, too filled with the importance of our mission, we have Doc Dobb's humbling influence. Always ready with a bit of irony and keen wit, Doc sees to it that the pages of his journal remain open as a forum and that they do not become a soapbox dominated by any particular group or prejudice. His philosophical outlook is often expressed in such homilies as "Let a thousand lilies bloom . . ." and "If it's grist for the mill, we'll grind it." That view is tempered by another of Doc's sayings, "The quick brown fox keeps his tail out of the lazy dog's mouth."

That is Doc's way of reminding us not to be influenced by any special interests. Its broad, objective approach has enabled *Dr. Dobb's Journal* to serve the world as the leading publication in its field. But more than a publication, it is a style of thinking which makes technology work for humans — and not the other way around.

Marlin Ouverson
Editor

Contents

VOLUME 4, NUMBER 31, 1979

2 Programming Pastimes and Pleasures Charles Wetherell
4 CHAOS: An Interactive Timeshared Operating System for the 8080 Jeff Levinsky
12 TUFIVE: An Improvement on Hexadecimal Keneth A. Simons
14 SAM76 Language Update Ancelme Roichel
18 Two Additions to the Z-80 RAM Test Ronald Derynk and Bock W. Lee
21 The Mystery of Ackermann's Function Mike Gabrielson
22 PET BASIC Renumber Bill Seiler
26 SAM76 Utility Function (And a Script Called Monkeys) Claude Kagen
30 Add a Trap Vector for Unimplemented 6502 Opcodes Carl W. Moser
32 A Mastermind Player for the National Semiconductor SC/MP Microprocessor Charles Kapps
34 Performing a Cost-Benefit Analysis of an Improvement Designed to Reduce Home Fuel Costs Michael Trombetta
36 A Primitive Report Generator Implemented in Zilog System Commands W. M. McKeeman
38 Memory Map Program for the Z80/8080 Robert Alkire

VOLUME 4, NUMBER 32, 1979

48 A Voyage to Antares Robert Burt, Judith Wasserman, Ramon Zamora
58 An Unusual Pseudorandom Number Generator Program H. T. Gordon
62 Let Your Computer Speak ASCII G. Gaugler
64 Genie, An Immodest Proposal David Chapman
73 Pascal Bibliography Mike Gabrielson
75 Features for TDL Editor, Assembler and Text Output Processor R. J. Long
78 A 2708 PROM Programmer for the 6800 E. Insam
80 Programming Pastimes and Pleasures Charles Wetherell
89 The Largest Known Prime Curt Noll

VOLUME 4, NUMBER 33, 1979

92 Falconer Floating Point Arithmetic, Part I Charles B. Falconer
103 CONVPATB: Palo Alto Tiny BASIC to ASCII Paul Holliday
112 QUOLST Gary D. Gaugler
113 Sorting Sequential Files in Place David Burris and Wayne Byrd
114 Vice Versa: Pencil to CPM and Reverse Jim Gagne
118 Build a Self-Learning No-Programming Computer with Your Microprocessor Klaus Holtz
126 A Closer Look at the MC6800 Curtis Roads
128 Label BASIC, a BASIC Preprocessor Sylvan Rubin
129 PIAs for KIM Joel Swank

VOLUME 4, NUMBER 34, 1979

140 WP-68 Word Processor Gary D. Gaugler
152 Falconer Floating Point Arithmetic Charles B. Falconer
162 Bringing up UCSD Pascal Without CP/M Mike Gabrielson
164 Unspecified 8085 Op Codes Enhance Programming Wolfgang Dehnhardt and Villy M. Sorenson
166 Conversion: 8-Bit Memories to 16-Bits George Lyons
171 A Disk for Frits Pac Dale Puckett
173 OSI BASIC for the KIM-1 Jonathan M. Prigot
176 Increase your Poly 88 Pleasure David L. Johnson
177 Product Evaluation: PERCOM LFD-400 Douglas K. Beck

VOLUME 4, NUMBER 35, 1979

188 The Software Tools FORMAT Program: An 8080 Version Mike Gabrielson
199 Changing Bases Theodore C. Hines, Rosann Collins, Jerry Russell, Linda Spencer
200 A Z-80/ZAP DISASM B. W. Lee
209 Patches to SWTPC BASIC: Change Output Format of Numbers Robert L. Pigford
210 Programming Pastimes and Pleasures Charles Wetherell
212 Patching the SWTPC Co-Resident Editor-Assembler Geoffrey A. Gass
221 Tiny-C Interpreter on CDOS Ray Duncan
224 ODT User's Manual Gary D. Gaugler
228 Software Standards, Who Needs Them? Tom Pittman

VOLUME 4, NUMBER 36, 1979

236 Discovering the 26th Prime Curt Noll
239 XS-0: A Self-Explanatory School Computer J. Nievergelt, H. P. Frei, H. Burkhart, C. Jacobi, B. Plattner, H. Sugaya, B. Weibel, J. Weydert
255 Selecting Business Software for Microcomputers Greg B. Scott
259 What's a COSMAC? Karl Mosgofian
260 EXOS – A Software Development Tool Kit for the 6500 Microprocessor Family Don Colburn
263 How To Make Your Computer Talk Charles M. Phelan
264 A Look at Computer Music Curtis Roads
268 Growing, Pruning and Climbing Binary Trees with tiny-c Les Hancock
273 Programming Pastimes and Pleasures Charles Wetherell
274 Further Thoughts on Ackermann's Function Lloyd Rice

VOLUME 4, NUMBER 37, 1979

282 On Computer Crime Bill S240 Jon K. Taber
288 Programming Pastimes and Pleasures Charles Wetherell
290 UCSD Pascal to CP/M File Transfer Program Ronald G. Parsons
295 A Floating Point Subroutine Package for the 1802 Paul Wasserman
298 Those All-Important Extras Don Colburn
305 Context Switch for the 6800 M. Mickelsen and N. Worth
306 Diskette Copying on the INTEL Single Density DOS Bill Collis
308 Hardware Relative Addressing for the 8080 John Beetem

VOLUME 4, NUMBER 38, 1979

324 An Electromechanical Household Servant D. J. Reynolds
335 A Fast Repertory Telephone Dialer Program S. Cash Olsen
336 A General Purpose Data Compression Program Graham K. Jenkins
339 A Potpourri of Utility Functions for LISP Paul Tarvydas
344 Machine Language Program Displacement Table Colin Keay
346 MITS Sequential File Dump A. L. Bender
350 A Proposed Standard for a 16 Bit System Data Exchange Charles B. Falconer
354 A Z80 Micro-Disassembler Holger Petersen
357 Modular Programming with the APPLE II R. F. Zant

VOLUME 4, NUMBER 39, 1979

366 Tiny BASIC for the F8 Jerry D. Fox
381 Page List for the APPLE Robert D. Diaz
384 Just Poking Around with my PET Gary Ratliff
386 Quick and Dirty Routines for the Sweet-16 Steve Wheeler
388 Producing Pictures on Your Computer With a Diablo Printer
Jay C. Bowden and Anna K. Scharschmidt
392 David and Goliath: The Little Guy is a Challenger! David Morganstein
394 Programming Pastimes and Pleasures Charles Wetherell
396 Main Software Teletype Terminal Interface Donald Faulkner
400 Common Instructions of the 6800 and 6502 B. T. G. Tan

VOLUME 4, NUMBER 40, 1979

408 Some Improvements on 8080 PILOT Tim Scully
422 A District Attorney Responds to Jon Taber Donald J. Ingraham
423 Jon Taber Responds to Donald Ingraham Jon Taber
424 North Star Disk STOIC, An Introduction B. W. Lee
430 DAL: Data Acquisition Language Gregg Marshall
437 Complex Pseudorandom Sequences from Interlaced Simple Generators H. T. Gordon
440 Dumping Files Under FLEX Freeman L. Moore
442 Prelim Programming Specs VDM-2/Graphic Display Lee Felsenstein
446 An Interactive Heath H-8 Disassembler Ronald T. Borochoff
450 Interfacing My Home Computer To a Large Scale System Glenn Story
457 PRTECT Your Space L. H. Reid

465 Index

About People's Computer Company

People's Computer Company, the publisher of *Dr. Dobb's Journal,* was founded in 1972 to demystify computers by teaching people how to use them. It is still carrying on that crusade through magazines, books, and informal classes.

As founder Bob Albrecht explained 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."

Since then, as a non-profit, educational corporation, PCC has published a newspaper, three magazines, and several books on home/personal computing including the popular book of games, *What To Do After You Hit Return*, now in its sixth printing. It has also operated a neighborhood computer center, a mail-order bookstore, and a computer van.

People's Computer Company sprang from the same roots as the *Whole Earth Catalog.* Both were offshoots of the Portola Institute of Menlo Park, an organization dedicated to investigating and promoting the idea of alternative technology. For the founders of PCC, that meant learning to use computers as a liberating force in society.

PCC has concentrated on small, personal computers – the kind anyone can learn to use and many families can afford. In PCC's view, familiarity with computers breeds confidence and greater control over one's own life. PCC recently initiated ComputerTown, USA!, which sponsors weekly computer nights at the Menlo Park Library. Sponsored by a grant from the National Science Foundation, ComputerTown, USA! is a grass roots, economical model showing how to offer *everyone* in a community of 27,000 the opportunity to use a microcomputer. It utilizes the resources of the community – public library, schools, community center, local businesses, donations of hardware and volunteers – to bring direct, hands-on experience with microcomputers to children and adults. ComputerTown, USA! publishes a monthly newsletter and provides a design of microcomputer courseware.

People's Computer Company currently publishes two magazines for two distinct audiences. The older, *Recreational Computing*, is aimed at the computer novice and at the intermediate hobbyist. *RC* started out as a newspaper called *People's Computer Company*, evolved into *People's Computers*, and took its present name with the January-February 1979 issue. The magazine focuses on the kind of educational experience one gains while having fun.

"*Recreational Computing* is focused more on the beginner in an informal learning environment, particularly the home," says publisher Michael Madaj. "We've been pointing the way for computer users for nearly ten years and now the computer revolution is upon us!"

Dr. Dobb's Journal, PCC's second publication, had a humble beginning in January, 1976, as a short-term mimeographed forum for the newly written Tiny BASIC language. Reader response was so strong that the *Journal* went into steady publication. Most *DDJ* readers describe themselves as having intermediate to extensive computer experience.

Dr. Dobb's Journal is a reference journal read worldwide. Ten percent of the circulation is outside the United States. The magazine contains sophisticated discussions of programming techniques and hardware modifications. About half of each issue is devoted to free software. *DDJ* appears monthly, and has a very loyal following. Volumes one through five of *DDJ* cover its five years of publication (1976-1980), and are available in attractive book form from Hayden Book Company (Rochelle Park, New Jersey).

A third PCC publication, *Computer Music Journal*, has been acquired by The MIT Press. *CMJ*, now in its fifth year of publication, and highly respected in the computer music field, is under the fine editorship of Curtis Roads. PCC is happy and proud to have made such a publication possible, and expects it to continue to grow and flourish in its new home.

PCC's newest project is PCNET (The Personal Computer NETwork), a developmental telecommunications project. Software (PAN) is now available from PCC which will turn any PET into a computer mail device. PCC has also put together a directory of Computerized Bulletin Board Systems. These computerized bulletin boards work just like ordinary bulletin boards except that instead of paper and thumbtacks they use a terminal, a modem, and the telephone network. The directory is regularly published in *Dr. Dobb's Journal* and *Recreational Computing.*

People's Computer Company occupies the second floor of a small office building in downtown Menlo Park, California. The staff members have professional interests in education, art, musical composition, computer science, psychology, and journalism.

People's Computer Company
1263 El Camino Real
P. O. Box E
Menlo Park, CA 94025

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 31 | January 1979 | Volume 4, Issue 1

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

Programming Pastimes and Pleasures *Charles Wetherell* **4**
A new monthly column

CHAOS *Jeff Levinsky* **6**
An interactive timeshared operating system for the 8080

Performing a Cost-BenefitAnalysis *Michael Trombetta* **36**
An improvement designed to reduce home fuel costs

A Primitive Report Generator *W.M. McKeeman* **38**
Implemented in Zilog System Commands

Add a Trap Vector *Carl W. Moser* **32**

TUFIVE: An Improvement on Hexadecimal *Keneth A. Simons* **14**

SAM-76 Language Update *Ancelme Roichel* **16**

The Mystery of Ackermann's Function *Mike Gabrielson* **23**

PROGRAMMING PASTIMES AND PLEASURES

© 1978 CHARLES WETHERELL

Welcome to the first edition of *Programming Pastimes and Pleasures*, a column I hope will become a permanent feature of *Dr. Dobb's*. Together we will explore the prosaic and fantastic possibilities of programming. Sometimes I will pose a problem to solve; sometimes I will discuss a programming technique or idea you may not have heard of; at other times I will tell you about a game you can play with your computer. At times I will be serious, and at other times I will be off-the-wall. Along the way, you will learn things you didn't know before, and you will find wonderful new ways to waste time with your computer. Most of all, I hope you will have fun.

Before we begin, you ought to know a little about me. I began programming in 1961. Since then I have had all the academic tickets punched, right up to the Ph.D. To support myself, I have done business programming, applications programming and systems programming. Right now I teach at the University of California at Davis and also work as a systems programmer. I've spent a lot of time lately working on standardizing extensions to FORTRAN, about as dull a job as you can imagine. My first publication was a computerized referee for *Kriegspiel*, a double blind version of chess. Since then, I have wasted more than 500 hours of big computer time simulating blackjack get-rich-quick schemes. Lately, Prentice-Hall published my book of computer problems *Etudes for Programmers*, which I know you will want to run out and buy today. I am looking forward to meeting or hearing from many of *Dr. Dobb's* readers.

This Month's Problem

Early this month I went on business to Lake Tahoe. While there, I did a little research into experimental probability distributions (if you don't have a research grant, this is known as gambling). While wandering about the casinos, I came across a bank of poker machines. Of course, they don't play real poker, but since I won $10.00 after betting only 75¢, the machines seemed pretty interesting to me. Besides the payoff, I noted some possible computer problems.

But first, how do the machines work? Take your quarter and drop it in the slot. The machine will deal you five cards. When you see the hand you may discard zero, one, two, or up to all five cards. The machine then deals enough new cards to replace the ones you discarded. The payoff depends on the quality of your final hand. For example, you pay 25¢. The machine deals

♠ 2 ♦ A ♥ 7 ♣ 7 ♦ 5 •

You decide to hold onto the pair of 7's. The machine deals three new cards to replace your discards.

♦ 6 ♠ 7 ♥ 7 ♣ 7 ♦ K •

What do you know—three of a kind and 75¢ tumbles into the payoff slot! Better quit while you're ahead, buddy..

The payoff table is straightforward. For each 25¢ bet, you receive back

$125	for a royal flush
$25	for a straight flush
$10	for four of a kind
$2.50	for a full house
$1.75	for a flush
$1.25	for a straight
$.75	for three of a kind
$.50	for two pair

and tough luck for anything less. Notice the payoff *includes* your original bet.

Now, it's pretty obvious that if you're dealt a pat hand (that is, one that is already a winner), you will discard only useless cards (like the fifth card from two pair). Similarly, you would almost certainly throw away the odd card from any four flush (that is, a hand with four cards of one suit and a fifth card from another suit) hoping to pick up a flush for $1.75. But it's not so clear what to do when you are dealt a mess like

♦ 2 ♣ 6 ♥ 5 ♣ J ♠ 9 •

Should you throw the whole hand back and get five new cards? Should you hold the two clubs as a start on a flush? How about the 2, 5, and 6 as part of a straight? Maybe you should take up a simpler game like matching pennies against your four-year-old cousin . . . .

Now comes the computer problem. First, what are all the possible combinations of cards you can be dealt? We are interested in any combination like a four-flush, a bob-tail straight (a straight of four cards missing one card from either end), an inside straight (missing one card in the middle of four), or one pair which might turn into a payoff. Suits don't matter except for flushes and denominations matter only for straights (two's and three's are as good as aces and kings in this game).

Second, once you have the list, what is the probability that you will draw each kind of hand? Third, with each kind of hand, what cards should you discard and what are your probabilities of getting one of the winning hands? Fourth, how much do you expect to win for each quarter you put in the slot?

I can think of at least two ways of solving these problems. But, so as not to spoil your fun, I'm not going to write about them now. Instead, when you have a good solution, send it to me in care of *Dr. Dobb's*. In a future issue, I will describe the best solution methods sent to me and give the author's names in this column.

Postscript: In an ordinary poker game, discards can not be redealt to fill out the hand on the second round. Does it make a difference to your answers to questions three and four if your discards can be reused? How much?

CHAOS

An Interactive Timeshared Operating System for the 8080

BY JEFF LEVINSKY
3632 Governor Drive
San Diego, CA 92122

CHAOS, an acronym for the Clairemont High Advanced Operating System, and its successor, CHAOS II*, are timeshared interactive systems which operate on the 8080 microprocessor. The following provides an overview of the goals of the system, examples of how CHAOS is used, and a description of the internal structure of the system, with a brief discussion of the implementation of directories and command processing. For further information, references to other articles and texts are included.

Origins of CHAOS

The initial design for CHAOS was based upon the needs of a secondary school computer course. Typical computer resources of many high schools consist of a single *port* connected to a district-owned BASIC timesharing computer or to a state or university consortium. Since only one student at a time may use the computer, each student in a class of 30 receives under nine minutes of time per week. Expansion by purchasing additional ports is limited by the high cost ($7500 is a usual price for a single port). Alternatively, microprocessor systems with multiple floppy disk drives can be obtained for under $3000, and with CHAOS, additional users can be added for about $1000 each. Moreover, an onsite computer system can be tailored to better fit the educational needs of the users. For example, the primary concern in the design for CHAOS was the availability of several languages (BASIC, 8080 assembly/machine, a system command (job control) language, PASCAL, FORTRAN, and COBOL for business-oriented students) and individual accounts for students as well as mechanisms for shared files and an 'explorable' structure. This last goal was especially chosen for the highly motivated and gifted students who frequently wish to delve into the workings of the commands and internals of the system—such activity is, in terms of CHAOS, to be encouraged, as the result is a source of knowledgeable system operators and valuable insight into system deficiencies. Of course, CHAOS is protected from non-privileged users. Security is partly maintained by the directory system, and teachers using the system to handle grading or other confidential files can elect to use a separate floppy disk for total protection. In practice, each class also has a separate floppy disk, in order to make better use of disk drives, which appear to be the most costly unit. Finally, the presence of a complete computer at the school proves far more stimulating to students, especially those interested in hardware, than does a simple remote terminal.

The UNIX™ system, developed by Bell Labs for the PDP-11 computer series, served as the external model for CHAOS. Internally, UNIX contains some well-conceived and relatively simple features that CHAOS was unable to emulate, due to the limitations of the 8080 processor and the inefficiency of floppy disk storage. To produce a smaller and faster system, the majority of CHAOS is written in assembly language, as opposed to a high level language such as C. CHAOS II was developed with both hard disks and 16-bit processors such as the Z8000 in mind, so that performance can be upgraded without extensive software revision. CHAOS, like UNIX, is designed to be simple for the novice to use as well as a powerful and flexible tool in the hands of the experienced. Unlike bare-bones operating systems and exclusively high level systems, such as CP/M and UCSD PASCAL, respectively, CHAOS attempts to provide a full spectrum of computer capabilities.

A Sample Lesson

Let us imagine a student, named Fred, approaching CHAOS to write a program that will average together numbers stored in a file for one or more files. First, Fred must log onto the system (system responses are in boldface):

:login: fred
password:
CHAOS II
System backup at 3 pm.
You have mail.
%

After the password has been correctly entered (it does not echo), some system messages are printed and then the percentage sign prompt is given. Mail may be obtained by:

% mail
From ethel 9:30 AM Tue Sept 5 1978
Club meeting after school.
%

'Mail' is an example of a *command* and is actually a non-resident program brought temporarily into Fred's work area.

Since the averaging program will be written in BASIC, that language is entered:

% basic
OK

CHAOS uses MITS BASIC Version 4.1 which provides a full set of string and file capabilities. The BASIC is actually always resident—the command 'basic' merely routes the user into it. The substance of the program is now entered:

*Since the layout of CHAOS and CHAOS II, as opposed to the actual code, is essentially the same, 'CHAOS' will be used when no distinction between the two systems is necessary.

```
10      REM     PROGRAM TO PRODUCE FILE AVERAGES
20      SUM=0           'SUM OF ENTRIES
30      COUNT=0         'COUNT OF ENTRIES
100     REM     ACCEPT FILE NAME
110     LINEINPUT "FILE?"; FILE$
120     IF FILE$="" THEN END
130     OPEN "I",1,FILE$
200     REM     SUM AND COUNT ENTRIES
210     IF EOF(1)  THEN CLOSE : GOTO 300
220     INPUT #1, NUM
230     SUM=SUM+NUM : COUNT=COUNT+1
240     GOTO 200
300     REM   OUTPUT
310     IF COUNT=0 THEN PRINT "NO ENTRIES" ELSE PRINT SUM/COUNT
320     GOTO 100
```

and tested:

```
RUN
FILE? DATA1
32.778
FILE? DATA2
29.14
FILE?
OK
```

Now, the program is stored (in Fred's directory by default) and BASIC is exited:

```
SAVE "AVERAGE"
OK
CHAOS
%
```

Fred can now check that the file indeed is in his directory via the command 'dir':

```
% dir
average
data1
data2
master
stuff
%
```

More information about the contents of the directory could be obtained by

```
% dire -al
```

where the 'a' and 'l' are flags interpreted by 'dir' as a signal to provide special options. Commands may inspect flags and other arguments typed upon the command line at will. In this case, 'a' causes all filed in the directory to be listed normally files with names beginning with a period are not printed out) and 'l' causes length, access, and disk information to be given for each file. At any rate, Fred can run his new program without re-entering BASIC:

```
% AVERAGE
FILE? DATA2
29.14
%
```

Here, AVERAGE was run just as a command—which is not surprising given that the commands 'mail,' 'basic,' and 'dir' are all actually written in BASIC.

The argument facilities exist as *system calls* and are available in BASIC to privileged users. CHAOS offers an all-or-none policy in this area: either one has or does not have access to BASIC commands such as POKE and OUT and the ability to make system calls. The right to use these is known on CHAOS as 'FRIBL' (pronounced 'fribble'). A CHAOS command such as 'dir' may have 'FRIBL' although the user who executes the command does not. One way that Fred may obtain 'FRIBL' is to move from his directory into the directory 'master,' in which a special version of the command 'basic' exists. Since Fred has a pointer to 'master' (see the results of 'dir' above), he transfers directories by:

```
% chdir master
```

and then enters the version of BASIC which gives him 'FRIBL.' Although in BASIC, Fred can still transfer back to his own directory, which is what ' . . . ' refers to here*:

```
% frbasic
OK
chdir " . . . "
OK
```

The modified version of AVERAGE that uses the argument facility is:

```
20      SUM=0           'SUM OF ENTRIES
30      COUNT=0         'COUNT OF ENTRIES
40      ARG=2           'ARGUMENT NUMBER
50      DIM M(5)        'ARRAY FOR SYSTEM CALLS
60      FRIBL           'THIS COMMAND NEEDS FRIBL
100     REM     ACCEPT FILE NAME
120     GOSUB   9000

320     END

9000    REM     READS ARGUMENT #ARG INTO FILE$
9010    M(0)=36         'SYSTEM CALL NUMBER
9020    M(1)=ARG
9030 X=USR0(VARPTR(M(0) ) )
9040 X=1  :  FILE$= " "
9050 FILE$=FILE$ + CHR$(PEEK(M(4) )  AND 127)
9060 IF(PEEK(M(4) )  AND 128) = 0 THEN M(4)=M(4)+1 : GOTO 9050
9070 RETURN
```

Lines 130 through 310 are the same as before. The subroutine at 9000 reads in the argument ARG from the command line into the string FILE$. There are several ways to accomplish this. Here the pointer to the start of the argument returned by the system call in M(4) is used to PEEK out the characters (the most significant bit being on only in the last character of the argument). FRIBL at line 60 can only be entered here because Fred already has FRIBL.

*Paths are dynamic in CHAOS, as opposed to static on UNIX.

Again, the program is stored and run back at the system level:

```
SAVE "AVERAGE"
OK
CHAOS
% AVERAGE DATA2
29.14
%
```

Now, however, averaging is only accomplished for a single file. To permit several files to be averaged, one could write a loop in BASIC to increment ARG (null arguments can be tested for), but CHAOS itself provides an alternative method. This consists of writing a file of commands known as a *"shell"* file (the command line processor is known as the *"shell"*) to make use of some argument processing commands. In this case, Fred produces the shell file 'AVERAGER' which contains the following text lines:

```
:loop
if $2a a  ; exit
echo $2
average $2
shift ; goto loop
```

When the shell file is executed, each command inside is executed in turn. The first command here begins with a colon, and serves only as a place holder for the label 'loop.' The 'if' command in the next line compares two strings/arguments for equality, in this case the '$2a' against the 'a.' A dollar sign followed by a number is a macro, and so '$2a' will be replaced by the second argument on the command line with AVERAGER with an 'a' appended at the end. If this is a null argument, then '$2a' will match 'a' and thus the exit command on the same line will be executed causing the whole shell file to terminate. If some argument '$2a' does exist, then 'exit' will not be executed. On the next line, 'echo' is used simply to print the second argument out, and after that, the average of the file is printed. 'Shift' causes arguments to be shifted to the left, so that what was the third argument to AVERAGER will become the second. Finally, the 'goto' will cause a jump back to the first line for another averaging.

To use AVERAGER, Fred merely says:

```
% AVERAGER DATA2 DATA1
DATA2
29.14
DATA1
32.778
%
```

At this point, Fred may wish to have AVERAGE and AVERAGER moved into the root directory so that all users can access them. Then, by typing a control -d, Fred exits from the system:

```
% d
:login:
```

A few observations can be made here. From the user's viewpoint, the command developed above is convenient as the standard system-wide format for the arguments is easy to remember. Having developed a command that becomes available to all users, Fred is motivated to continue to produce and to upgrade. Meanwhile, other programmers can readily incorporate AVERAGE or AVERAGER into their commands using the capabilities of either BASIC or the SHELL. CHAOS also permits commands to be written in native code, and these may also invoke other commands and make system calls. UnFRIBLed users may run and debug 8080 machine code, but only under the watchful eye of a simulator which prevents system crashes. Finally, if Fred is on a multi-user version of CHAOS, other users may at the same time be running and debugging commands/programs independently. Of course, the system has safeguards to prevent file modification from occurring while others are using the same file.

The Structure of CHAOS II

CHAOS II consists of about 10K of code which resides in the lower 32K of memory alongside the 20K MITS BASIC. The system is both conceptually and physically divided into a series of nested levels, which are diagrammed in Figure 1. The contents of a given level may call upon an inner level for assistance, but the reverse is not permitted. For example, the lowest level is the 8080 hardware: this is used by all other levels but cannot itself use any of the software in those levels. One may be reminded of a black hole, into which any object may be sent, but from which no object will emerge. Each level other than level 0 consists of a number of carefully defined system calls, such as the one used above to obtain a pointer to a shell argument. Typically, each level is written, tested, and documented by one person. Briefly, these levels are:

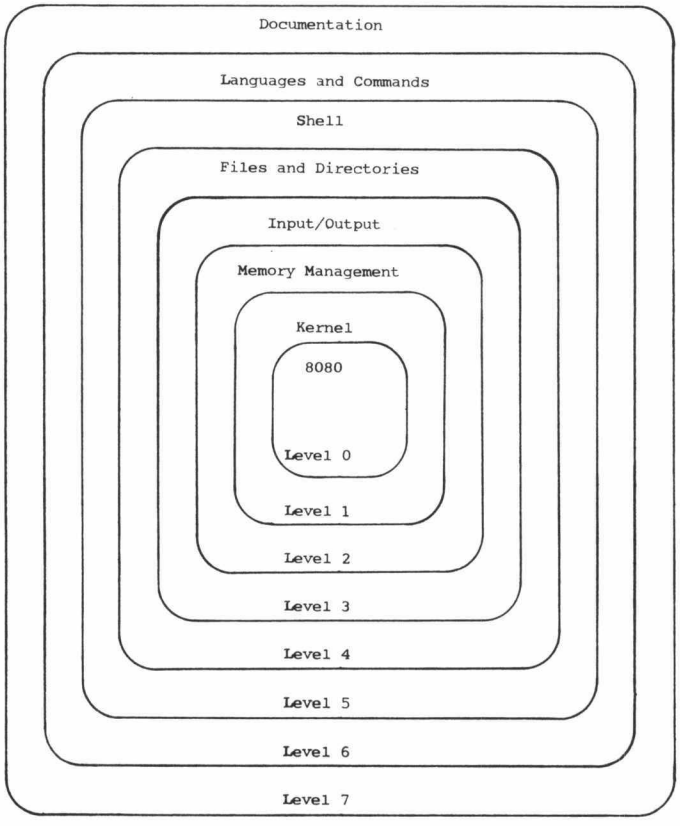


Figure 1. CHAOS System Structure

Level 0: The 8080 hardware required to run CHAOS can consist of no more than a standard MITS 8080 floppy disk system with about 40K of memory, if only a single-user version is required. The system is configured as shown in Figure 2 with BASIC and CHAOS in the low 32K, the user's system variables in the 33rd K of memory, and with the remainder serving as the user's workspace.

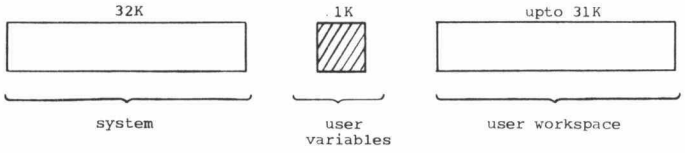


Figure 2. Single User Configuration

If multiple users are to be supported, then one of two configurations is possible. In the first, shown in Figure 3, a separate 1K variable area is provided for each user and all workspaces are located in the single upper block of memory. All of the variable areas are addressed the same but only one may be active at a given time.

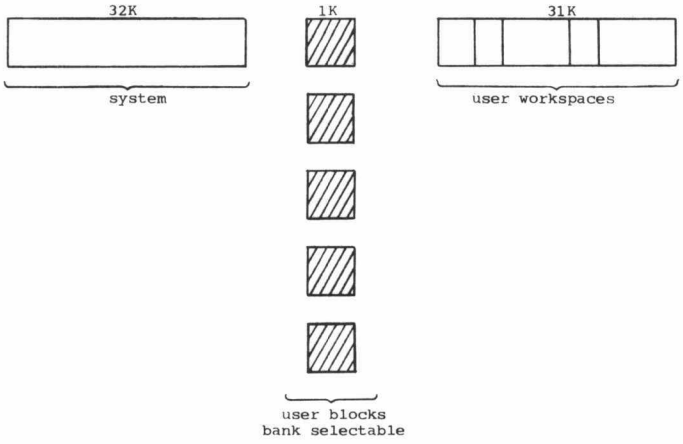


Figure 3. Multi-user configuration

This is controlled by a signal output to a parallel port which in turn selects one of eight 'banks' from a memory board, as is shown in Figure 4. The front board in that figure is active only when the low 1K of the boards is being addressed – otherwise the rear board is enabled.

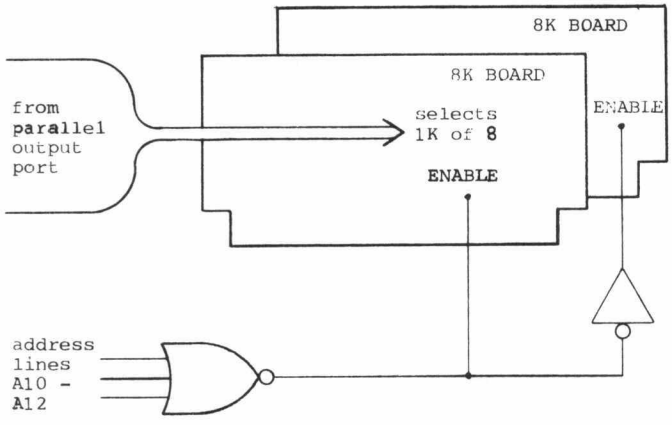


Figure 4. Multi-user control hardware

When large workspaces are required, the configuration given in Figure 5 is used. Under this method, both the 1K variable spaces and the workspaces are selected by the signal sent to the port. The hardware to implement this is shown in Figure 6, and consists only of a decoder connected so as to disable all but one set of 16K boards, i.e., all but those of the active user.

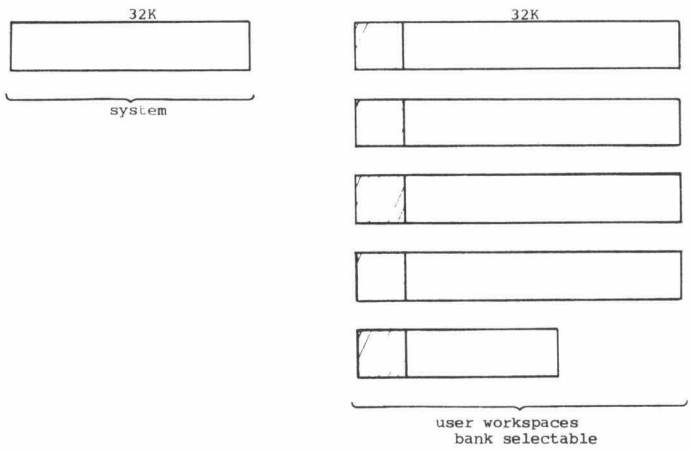


Figure 5. Multi-user configuration

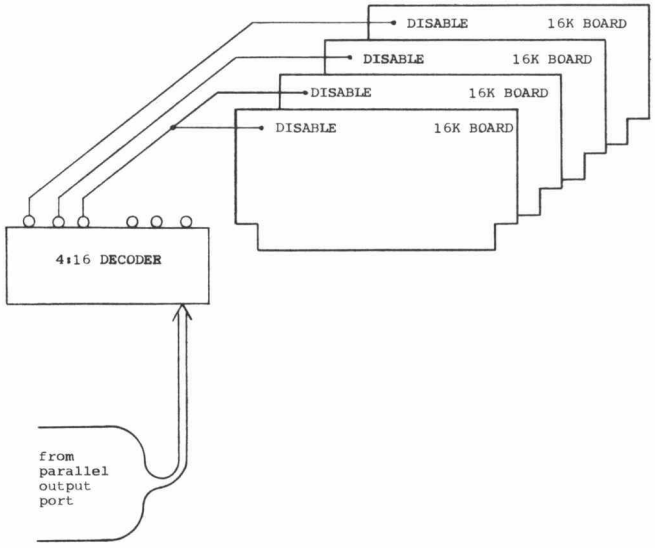


Figure 6. Multi-user control hardware

Level 1: The *kernel* provides each user/process with a slice of computer time, maintains the system clock, and provides some interprocess synchronization. The kernel handles interrupts and controls the memory switching described above for multi-user systems. Within the kernel, a circular queue of active processes is kept for use in scheduling. If a process requests a semaphore (a P operation) when the semaphore is currently unavailable, the process is suspended from the circular queue and placed upon a FIFO queue associated with the semaphore. When another process then releases the semaphore (a V operation), a process from the corresponding semaphore queue is then restored to the active queue.

Level 2: Memory management consists of the allocation of *segments* within each user's workspace. The layout of the typical workspace is shown in Figure 7: a stack of segments grows upwards from the bottom of the workspace while the 8080 stack grows downwards from the top. Each segment is either a program/command or a special data area required by a program. The segments are handled similarly to 'activation records' in programming languages and system calls exist to create, modify, and remove these.

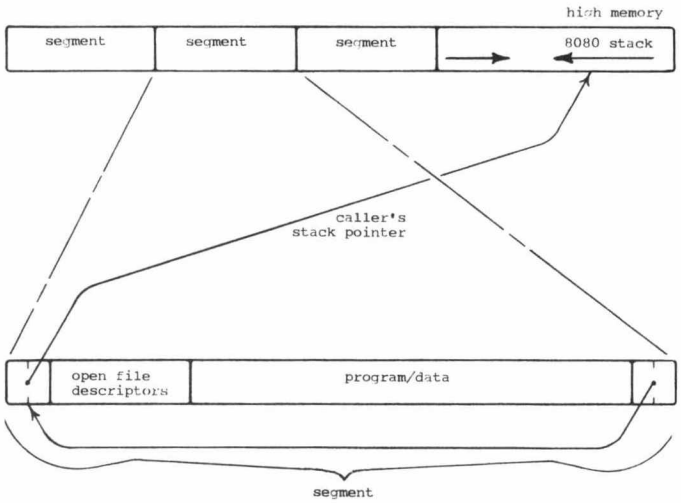


Figure 7. User's workspace and segments

Level 3: Input/output constitutes a minor part of CHAOS as complex buffering and device drivers are avoided. At least in secondary schools, students enter data relatively slowly and do not want blindingly fast screen listings. Therefore, no tremendous efficiency is required. In the multi-user version of CHAOS II, vectored interrupts can be added to achieve somewhat smoother input/output, especially as disk head motion need not lock onto the processor.

Level 4: Files and directories are quite elaborate in CHAOS as evidenced by the example given above. Essentially, a directory is simply a file of pointers to other files, and therefore directories may point to other directories (as 'fred' pointed to 'master'). In CHAOS II, file and directory functions are primarily handled by modified code from the BASIC.

Level 5: The Shell was already introduced as the program responsible for processing command lines. How it operates is explained more thoroughly below, but its position in the hierarchy is clear here, for it must rely upon input/output to receive the commands from the user, upon memory management to create and remove the segments to hold the commands, and upon the file and directory system to locate and to read in the commands. The Shell is probably the most independent of all the levels described so far, and certainly other shells which act in quite different ways can be used. In fact, halfway through the debugging of the original CHAOS, one shell was pulled out and another substituted in for it without more than a handful of errors.

Level 6: Languages and commands are properly one level outside of the Shell, although parts of BASIC occur at inner levels as much of the BASIC code is used for input/output and for files. Several options exist for other languages: a Tiny-LISP and the 8080 simulator fit into a user's workspace under all configurations, while large languages require a full 31K workspace as do most business application packages.

Level 7: Documentation is of crucial importance on CHAOS since users have a variety of backgrounds and interest. After the UNIX fashion, each command and system call is given a one or two page summary which provides adequate information in most cases. Commonly used programs, such as the text formatter (word processor) and the BASIC are also described by tutorials which emphasize examples that users may actually type in and try. On a larger scale, the system is described by a set of charts which show the flow of control in operations such as argument parsing and directory searching. Finally, a current project is the completion of a manual detailing the overall operation of the system through diagrams and text, of which a condensed and simplified version will be provided to users seeking only general knowledge. CHAOS itself is expected to assist in documentation both by facilitating the production of manuals and updates and by providing a complaint facility through which frustrated command users may communicate with those responsible for maintaining the system.

The CHAOS Directory System

Each file on CHAOS is described by a *file pointer* which can be defined in PASCAL as:

```
type
   ftype = (basic, text, assembly, random, directory);
   rights = (w, o, r, e);
   filepointer = record
      name:          array[1..8] of char;
      origin:        integer;    [sector and track #]
      access:        set of rights;
      owner:          filepointer:
      lastwritten:   date;
      case oftype:   ftype of
         basic, text, assembly, random: (length: integer)
         directory:                     (quota: Integer)
      end;
```

With this, we might define a directory as:

```
   directory = set of filepointer;
```

(although this isn't legal PASCAL). How directories are actually implemented is not of importance: in CHAOS both files of descriptors and a common single disk directory (volume table of contents) have been used.

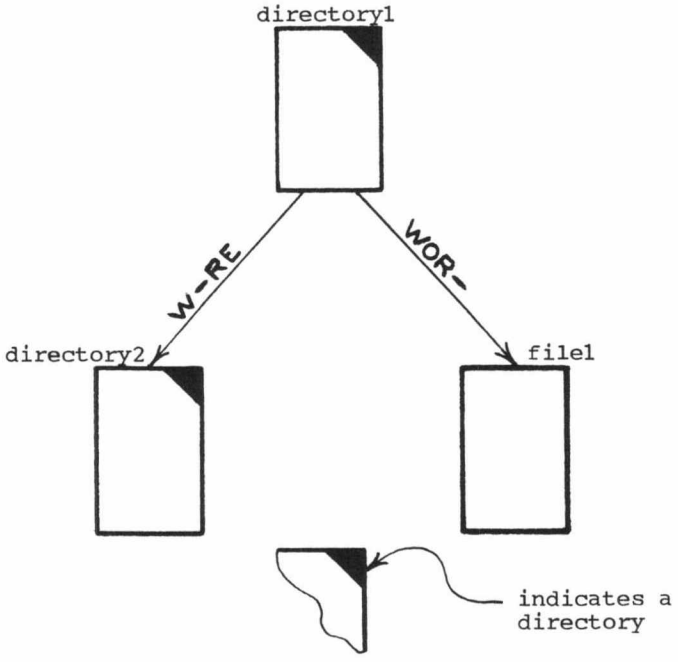


Figure 8. Simple Directory Structure

Mainly to introduce some notation, Figure 8 shows 'directory1' which owns two entries: 'directory2' and 'file1'. On each arrow are the access rights for the entry. These govern what operations a user in 'directory1' can perform upon the entries. The four possible access rights are:

w = write access
r = read access
e = execute access
o = override access

Having override access permits any operation to be performed. Access rights are also associated with a user in a given directory. To actually compute what access is permitted for an entry, the access rights associated with the owning directory are *anded* with those associated with the entry itself. This is best illustrated by example. Suppose that a user has r and e access when in 'directory1'. If that user accesses 'file1', then the rights are computed as:

```
--re
 wor-
--r-
```

The file can only be read. On the other hand, if the user has just o access for 'directory1', then the rights computed for 'file1' are:

```
-o--
wor-
-o--
```

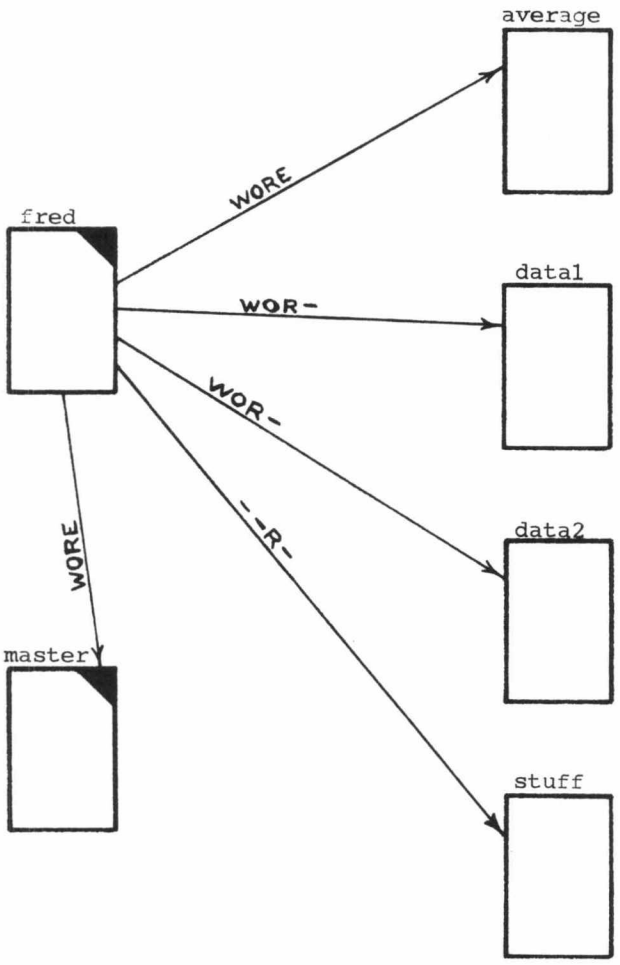


Figure 9. Directory structure for 'fred'

Here, since override access exists, all operations can be performed upon the file. When a user elects to change from one directory to another, the user's rights for the new directory are computed in the same manner. Supposing that a user has o and r access in 'directory1', the access rights for the user upon transfering into 'directory2' will be computed as:

```
-or-
w-re
--r-
```

When the user returns to 'directory1', the o and r status will be restored.

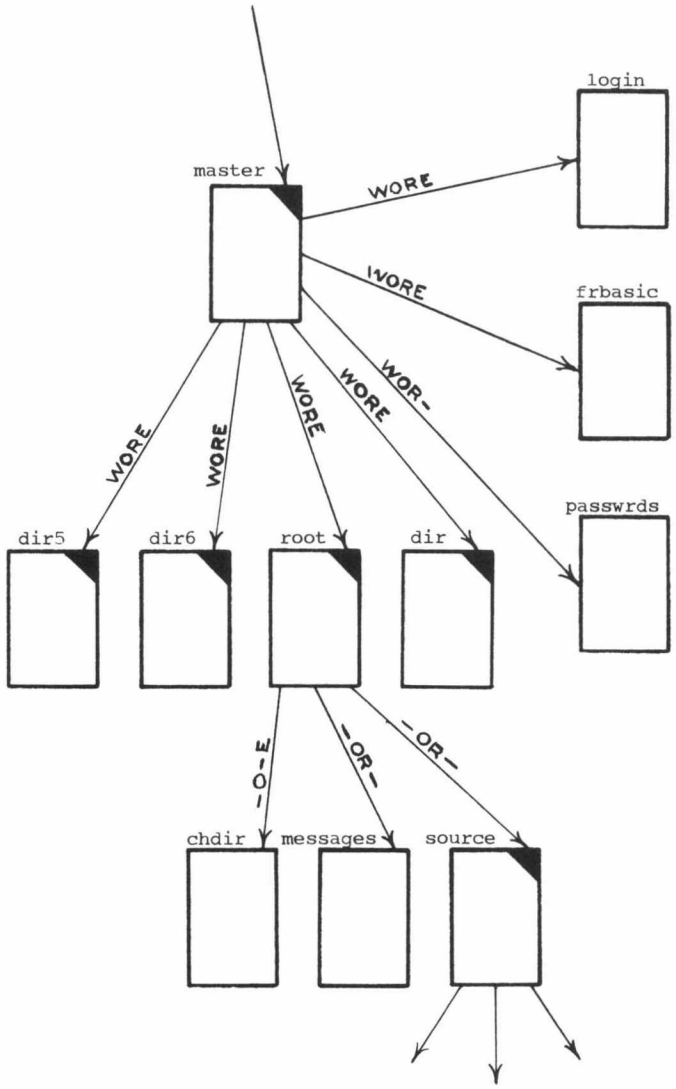


Figure 10. Directory structure for 'master' and 'root'

Figure 9 illustrates the directory 'fred' used in the sample session above. Since the data files are not generally executable, they are by default not given e access. Assuming that Fred had o access when in his directory, those data files could still have been executed by him—in which case they are treated as shell files. When Fred moves into 'master', he retains all possible rights. Figure 10 shows the typical structure of 'master'. The program 'login' controls the actual logging-on of users into CHAOS and accesses the passwords file. The directory 'master' then also contains pointers to all directories into which users may intially enter.

Since file pointers are constrained to point only to files on their disk (to allow single disks to be unmounted). 'master' is actually spread across all disks. One other directory of importance is 'root' as it contains all of the typically used system commands. The Shell first searches a user's directory for a file to execute and then, if the file is not found, the 'root' directory. Each user has an implicit pointer to 'root' with w, r, and e access. Hence, a user referencing the command 'chdir' when not in 'root' will have access rights computed as:

```
w-re
-o-e
---e
```

and will only be able to execute. In this example, the user will have only r access to the directory of source code and to the messages file. On the other hand, a user transferring into or logging into 'root' via 'master' will have full access rights due to the presence of the override access in all of the pointers within 'root.' Presumably, if a user turns off all rights on a pointer to a directory and then transfers into that directory, no operations will be possible. For this reason, the implicit access to 'root' is fixed rather than computed: the user will always be able to call 'chdir' in order to return to the original directory.

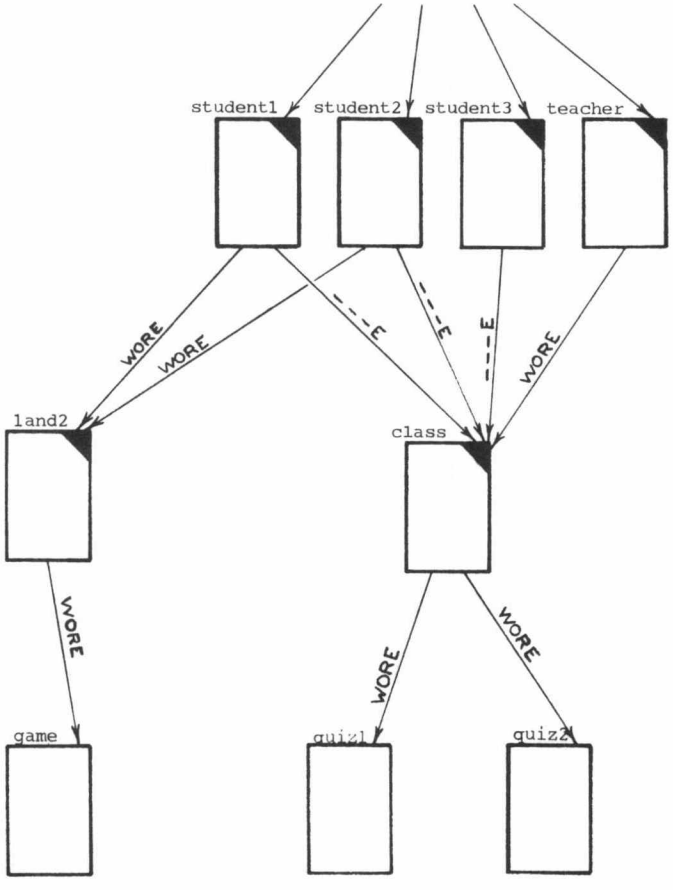


Figure 11. Directory structure for a class

In a classroom situation, access rights and directories become quite useful, especially as multiple pointers to directories may exist (multiple pointers to regular files are not typically permitted, to avoid the issue of choosing which owner is to be charged for the file's length). A common directory for a class is shown in Figure 11. The directory 'class' is akin to 'root' in that the students may only execute while the teacher has full access to the quiz files. The directory 'land2' exists so that two students may both work on a game program that is protected from other users. Of course, a user with access to 'master' has access to all directories under normal circumstances.

Implementation of the Shell

No doubt the most interesting feature of CHAOS, due to its command and argument processing abilities, is the Shell. Strictly speaking, the Shell is a function which processes one or more command lines to obtain a boolean result indicating the success or failure of the commands. The 'if' command presented earlier is capable of using the boolean value returned as a conditional, as are assembly language programs. A few variants of the function shell exist: shellf executes a shell file, and shelli produces an interactive shell (one that prints prompts and accepts command lines from the terminal—exiting only after a control-d is typed).

Since a shell file may, for example, invoke another shell file or an interactive shell, the shell function is recursive. The command lines and/or pointers into them are kept within a segment created for a specific invocation of the shell function. A segment associated with an interactive shell is dynamic in size as command lines inputted vary in length. A typical interactive shell segment is shown in Figure 12. Shell segments also contain a list of processed arguments from the command line currently being executed. These arguments can be accessed in various ways by the system calls contained in the Shell level of the system.

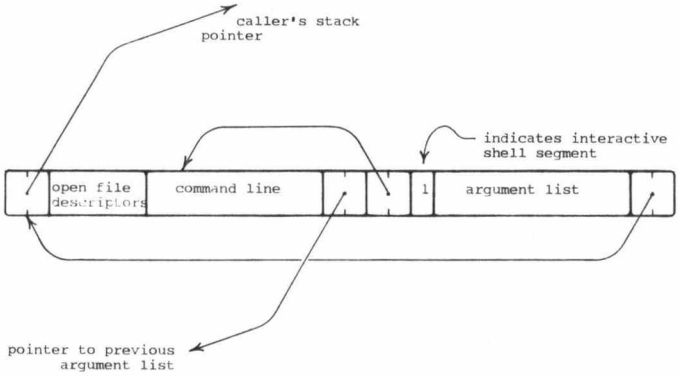


Figure 12. Interactive shell segment

The general algorithm for the shell function is very straightforward:

```
function shell (commandline: array [1..n] of char):
 boolean;
   status: boolean;
begin
   status:=true;
   repeat
      process command line;
      if legal then status:= shell(command)
         else status:=false
   until endoffile or (status = false);
   shell:=status
end;
```

The function shelli is slightly different in that it prints a prompt and requests a new line whenever an end-of-line is reached or an error in a command line is found. Non-interactive shells return upon an error: therefore when a shell file is found to be incorrect, execution of it will terminate and control will be returned to an interactive shell or machine language program. This may cause intermediate shell files to also terminate—the flow of control is handled automatically by the shells.

About half of the Shell is devoted to command line argument processing and especially to the generation of argument lists. In general, an argument is a string of characters delimited by blanks or other special characters. The command 'chdir master' consists of two arguments: the first argument is taken of course to be the name of the file to be executed. Arguments may contain special characters through the use of quoting, which entails either surrounding the argument with single or double quote marks or by turning off the special meaning of a character by preceding it by a backslash. For example, to run the command whose name is ';', one might say:

% ';' or % ";" or % \;

however '% ;' would not work as the semicolon would be taken as a command separator (in this case separating two null commands).

In addition to quoting and macros, CHAOS' Shell permits what is often called argument list generation or wild-character file names. Whenver certain characters are found (unquoted) in an argument, the Shell no longer treats the argument literally but instead takes it to be a pattern to be matched against the names of all files in the current directory. Those names which match are then included into the argument list (in alphabetic order). The special characters and their meanings are:

? matches any single character.

[defines the beginning of a character class.] defines the end. Characters appearing between the square brackets are all potential matches for a single character in a file name. Hence '[abcd]' would match a single 'a', 'b', 'c', or 'd'. A shorthand for this is '[a-d]'. If the character directly following the '[' is a dash, then the character class matches any character *but* those appearing inside (i.e., the intial dash negates the meaning). Since '[]' matches no character, '[-]' matches all and is equivalent to '?'.

* signifies that zero or more of the previous character or character class may be matched.

Some examples may clarify how file names can be selected using these patterns. Say that the current directory contains the files 'data1','data2', 'data33', and 'datum'. Then typing:

% print data?

is equivalent to typing:

% print data1 data2

as the question mark matches the '1' and then '2'. Neither 'datum' nor 'data33' was included in the list because the 'u' or the final '3' did not match. On the other hand,

% print dat??

is equivalent to:

% print data1 data2 datum

To get all file names of the form data⟨digit⟩, one might use 'data[0-9]'. This would not match a file named 'datam' for example. To also match 'data33', one might use 'data [0-9]*'. The asterisk applies in this case to the character class and causes zero or more digits to be matched, that is, both 'data' and 'data1234567' will also match this pattern. More generally, '?*' will match any file name (the question mark is optional in this case).

To process a command line, Shell repeatedly calls a scan function which in turn examines the command line for a single argument, appends one if found to the argument list at the end of the shell segment, and then reports back via a flag whether or not the end-of-line has been located. The scan function identifies quoted arguments, tracks down macros, and supervises argument expansion. The actual procedure to compare a pattern to a file name is somewhat complex, as recursion is required to handle patterns such as 'a*b*', and is based upon programs published in *Software Tools* (see References).

Finally, given the directory structure and shell calls outlined above, the actual external appearance of the system can be very easily described. The 'login' command in the master directory prints the request for a directory name and a password, and then determines whether such a directory exists and if the password is correct—using the pointers from 'master' to establish the user's original directory and access rights. Thus, each user is initialized to run the following:

```
program chaos;
   const        crash=false;
begin
   repeat
      shell("login");
      shelli
   until crash
end
```

References

For information on CHAOS and CHAOS II, write to Mary Elizabeth Kroening, 3632 Governor Drive, San Diego, California, 92122. Persons interested in specific applications of CHAOS II, such as in education, medical research, or business should ask for details on the high-level system packages being developed for these areas.

Brooks, Frederick P., Jr. *The Mythical Man-Month*. Menlo Park: Addison-Wesley Publishing Company, 1975.

Gries, David. *Compiler Construction for Digital Computers*. New York: John Wiley & Sons, 1971.

Kernighan, Brian W., and Plauger, P.J. *Software Tools*. Menlo Park: Addison-Wesley Publishing Company, 1976.

Ritchie, Dennis M., and Thompson, Ken. "The UNIX Timesharing System." *Communications of the ACM*, XVII 7 (July, 1974), 365-375.

Shaw, Alan C. *The Logical Design of Operating Systems*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1974.

Wirth, Niklaus, and Jensen, Kathleen. *PASCAL User Manual and Report*. New York: Springer-Verlag, 1975.

TUFIVE:

AN IMPROVEMENT ON HEXADECIMAL

BY KENETH A. SIMONS, CONSULTANT
2035 Willowbrook Drive
Huntingdon Valley, Pa. 19006

Note: Mr. Simons is constantly attempting to improve the field of mathematics. In a future issue, we will present an article by Mr. Simons detailing the history of numeric symbols for numbers *and a concept called OGS which converts all numbers into integers.* *–SMR*

The Need for More Compact Expression Of Numbers:

One of the first things a computer enthusiast learns is that, although the computer itself uses only binary–usually represented by 0 and 1–there are many possible ways in which numbers can be expressed at the input to the computer or from its output. There is, of course, *decimal*, based on the number 10; *binary*, based on 2; *octal*, based on 8; and *hexadecimal*, based on 16.

Each number base has its advantages. Decimal is our everyday number language; quantities expressed in this form are instantly recognizable. It is, however, a relatively complicated procedure to convert a given number from decimal to binary, or the reverse. The Octal base has two big advantages: 1) conversion to binary is simple; 2) it uses familiar symbols–the first 9 digits of the decimal number system including 0. Its disadvantage is that, for many numbers, more digits are required to express it in octal than in decimal. The decimal number 855, for example, becomes 1527 when expressed in octal. To circumvent this it has become increasingly common to use hexadecimal, based on 16, which has the single advantage of being more compact–the decimal number 8,987,605, for example, becomes 8922D5 when expressed in hexadecimal. The disadvantage is obvious: this bewildering combination of letters and numbers defies any quick interpretation (at least without a great deal of practice!) and one must rely on the computer itself to do the conversion to more familiar forms.

Why Stop Here?

Once we have taken the step of intermixing number and letter symbols to represent a number based on a power of two, it is not at all obvious why we should stop with the base 16. There are more letter symbols available, enough to allow a system based on 32, using only the capital letters (omitting O, I, and Z to avoid confusion with 0, 1 and 2 respectively). If we allow the use of both upper and lower case letters, as well as a few other symbols available on the ASCII keyboard, we could even go to a system based 64! Just for fun let's explore the possibilities.

First we need names, because it's awfully hard to talk about a number system without having a name for it, and let's see if we can do better than a mouthful like "hexadecimal," which is confusing anyway because it seems to imply a relation of some kind to the decimal system. How about TUFIVE, meaning, a system based on two raised to the fifth power, for the system based on 32. Following the same logic we would have TUSIX for a 64-based system, and we might even want to go back and reduce hexadecimal to TUFOR.

Now let's see how it might work. Table I shows a comparison of the decimal numbers from 0 to 64 with the same numbers expressed in binary, octal, TUFIVE and TUSIX (with one possible choice of symbols for the last two). The advantage of compactness can best be seen by expressing a large number in each system. Each of the following expresses the same number:

Binary	Octal	Decimal
100010010010001111010101	42221725	8987605
Hexadecimal	**Tufive**	**Tusix**
8922D5	8J8WM	cJFM

Conversion Between Number Systems

The process of converting between a binary number and any number system based on an integral power of two is about equally simple for all systems and follows the same rules. To convert from a given binary number to a number system based on the "nth" root of 2, break the binary number into groups of "n" digits, working from right to left, and write, in order, the single symbol in the "nth power" system corresponding to each group. Suppose, for example, we wish to convert the binary number shown in the preceding section to its equivalent in TUFIVE. First break it up into groups of five digits, starting from the right. Then write, below each group, the single tufive symbol representing that group (as found, for example, in Table I):

```
00100   01001   00000   11110   10101
  ↓       ↓       ↓       ↓       ↓
  4       9       0       W       M
```

so the TUFIVE number is 490WM.

Conversely if the "nth Power" system number is known, the corresponding binary number is found by writing the corresponding group of "n" digits below each symbol. To convert cJFM, in TUSIX, to binary, for example:

```
  c       J       F       M
  ↓       ↓       ↓       ↓
010001  001000  001111  010101
```

Table I
A COMPARISON OF NUMBERS WITH VARIOUS POSSIBLE BASES

Binary	Octal	Decimal	Hexadecimal	Tufive	Tusix
0000000	0	0	0	0	0
0000001	1	1	1	1	1
0000010	2	2	2	2	2
0000011	3	3	3	3	3
0000100	4	4	4	4	4
0000101	5	5	5	5	5
0000110	6	6	6	6	6
0000111	7	7	7	7	7
0001000	10	8	8	8	8
0001001	11	9	9	9	9
0001010	12	10	A	A	A
0001011	13	11	B	B	B
0001100	14	12	C	C	C
0001101	15	13	D	D	D
0001110	16	14	E	E	E
0001111	17	15	F	F	F
0010000	20	16	10	G	G
0010001	21	17	11	H	H
0010010	22	18	12	J	J

Table I (cont'd)

Binary	Octal	Decimal	Hexadecimal	Tufive	Tusix
0010011	23	19	13	K	K
0010100	24	20	14	L	L
0010101	25	21	15	M	M
0010110	26	22	16	N	N
0010111	27	23	17	P	P
0011000	30	24	18	Q	Q
0011001	31	25	19	R	R
0011010	32	26	1A	S	S
0011011	33	27	1B	T	T
0011100	34	28	1C	U	U
0011101	35	29	1D	V	V
0011110	36	30	1E	W	W
0011111	37	31	1F	X	X
0100000	40	32	20	10	a
0100001	41	33	21	11	b
0100010	42	34	22	12	c
0100011	43	35	23	13	d
0100100	44	36	24	14	e
0100101	45	37	25	15	f
0100110	46	38	26	16	g
0100111	47	39	27	17	h
0101000	50	40	28	18	i
0101001	51	41	29	19	j
0101010	52	42	2A	1A	k
0101011	53	43	2B	1B	m
0101100	54	44	2C	1C	n
0101101	55	45	2D	1D	o
0101110	56	46	2E	1E	p
0101111	57	47	2F	1F	q
0110000	60	48	30	1G	r
0110001	61	49	31	1H	s
0110010	62	50	32	1J	t
0110011	63	51	33	1K	u
0110100	64	52	34	1L	v
0110101	65	53	35	1M	w
0110110	66	54	36	1N	x
0110111	67	55	37	1P	y
0111000	70	56	38	1Q	z
0111001	71	57	39	1R	!
0111010	72	58	3A	1S	"
0111011	73	59	3B	1T	#
0111100	74	60	3C	1U	$
0111101	75	61	3D	1V	%
0111110	76	62	3E	1W	&
0111111	77	63	3F	1X	*
1000000	80	64	40	20	10

SAM76 Language Update

No. 4 - August 1978

BY ANCELME ROICHEL

This update will describe how the user may define a text to contain an assembly language program that will be executed when the text is fetched. In addition a number of anomalies and errors will be discussed and suggested corrections where practical will be shown.

This opportunity is also taken to extend my appreciation to the several users who have taken the time to feed back comments and pointed out errors or anomalies in both the text material as well as in the object code.

In any correspondence please give your phone number and an indication that you will accept collect calls. I can generally be reached at home on weekends and after 5p.m. on weekdays - phones are: 609-466-1129 and -466-1130. Additionally I will continue to send out preprints of upcoming updates, notices and articles to users who send me a self adressed stamped envelope containing one dollar to cover xeroxing and possible additional postage. If satisfied with what is received instead of cursing - merely "recurse" to get the next one. This operation is not a commercial one, the "Inc." in the name notwithstanding, and the financial situation is marginal at best.

It is the purpose of this section to correct a few errors and to explain certain anomalies or ambiguities in the text book section that contains the function definitions; appropriate changes will be made in subsequent reprintings.

Errors and anomalies in manual

The suggested corrections shown below do not help the user to locate precisely the section of code in the overall program. There are at least two ways of doing this: use a monitor search capability and/or use the %xqf,function/ to locate the first adress of the function program. The changes themselves may be made either using SAM76 procedures, or perhaps more simply with your monitor or debugger. The adresses of labelled locations used should be determined by examination of those lines of code which make use of these adresses. With the exception of the fix for the EP function, the corrections result in a requirement for equal if not less space.

The special notation shown is used to facilitate transfer of the system to other machines. Jumps shown as .JRxx are assembled as relative jumps on a z80 and absolute jumps on an 8080 system; the "xx" signifies the condition. The JRxx and JSxx notations mean respectively JMP to xx and CALL xx.

On z80 systems where the RST locations may be used as a means of having one byte subroutine calls, six of the JSxx functions have been defined as follows:

JSAE	RST 1	JSAS	RST 2	JSWA	RST 3
JSRB	RST 4	JSER	RST 5	JSLK	RST 6

AB - Alphabetic Branch — Change, the word "lesser" to read "greater" - that is to say that the word "DOG" is (USASCII wise) greater than "CAT"; the function operates properly in the interpreter.

EA - Erase All — The definition allows arguments signifying names of texts not to be erased; this feature has not yet been implemented.

ED - Extract D char. — There is a coding error that is readily corrected as shown below.

In addition the definition should be changed to include the following explanation:

"The text divider is assumed to be of zero width, and consequently the first character to the right of the divider is stipulated when D1 is equal to zero. Negative values of D1 bring characters from the left of the divider; it is thus possible to get some characters from each side of the divider if D2 is greater than D1 when D1 is negative".

```
;[$207/ED] ... new
$207:   JSCFGT
        LHLD    PAL2
        DAD     B
        NOP
        SHLD    PAL2
        XCHG
        JSGE
        XCHG
        .JRNZ   $2074
$2071:
;
```

```
;[$207/ED] ... old
$207:   JSCFGT
        LHLD    CHA1
        DAD     B
        DCX     H
        SHLD    PAL2
        XCHG
        JSGE
        XCHG
        .JRNZ   $2074
$2071:
```

EP - Erase Partition — Error in code causes erasure of all partitions, even if arguments are specified.

A simple correction is shown below which will permit one argument to be specified.

```
;[$151/EP] ... new
$151:   JSCFGT
        ORI     N200
        MOV     E,A
        LHLD    PAL2
$1511:  JSRB
        .JRC    $EP12
        .JRZ    $1512
        MOV     A,E
        CMP     C
        .JRZ    $1511
        ANI     N177
        .JRZ    $1511
$1512:  INX     H
        MOV     M,C
        .JMPR   $1511
$EP12:
;
```

```
;[$151/EP] ... old
$151:   JSCF
;
;
        LHLD    PAL2
$1511:  JSRB
        .JRC    $EP12
        .JRNZ   $1511
;
;
;
;
;
        INX     H
        MOV     M,C
        .JMPR   $1511
$EP12:
```

FDM and FL — An error in the code causes the search to go to the right instead of the left if the substring sought exists to the right.

This is a tricky fix that involves several subroutines and is not described at this time. If the substring sought only exists to the left then all is well.

FF - Fetch Field — It was observed that this function operates properly in that the correct element is returned, but that the text divider is not moved.

It is felt that this is probably the correct type of action and consequently the definition should be changed to delete the specification that the text divider is moved. This is not a copout, a fix to move the divider is very simple. In most cases where this function is used the purpose is to extract a "record" identified by a leading partition of some defined value, and thus provides a simple means of resequencing or reformatting a series of records stored in a single text. Allowing the function to move the divider would complicate this process.

HP - How Many Partions Due to error in code this function does not reveal the count of a stipulated partition value; the fix is quite simple and is shown below.

```
;[$149/HP] ... new      | ;[$149/HP] ... old
$149:  MOV     D,A     | $149:  JSCFGT
       MOV     E,A     |        XCHG
       JSCFGT          |        MOV     H,A
       XCHG            |        MOV     L,A
       .JMPR   $1491   |        .JMPR   $1491
$1492:                 | $1492:
                       |
```

RA - Return Argument Due to coding error wrong argument is returned as the value; the very simple fix shown below will solve this problem.

```
;[$215/RA] ... new      | ;[$215/RA] ... old
$215:  JSGE            | $215:  JSGE
       JSMSP           |        JSMSP
$2151: DCX     B       | $2151: JSAS
       MOV     A,B     |        JSAE
       ORA     C       |        .JRC    $2152
       JZ      MRAD    |        DCX     B
       JSAS            |        MOV     A,B
       JSAE            |        ORA     C
       .JRNC   $2151   |        .JRNZ   $2151
       NOP             |        JMP     MRAD
       JSMRP           | $2152: JSMRP
       JRAC            |        JRAC
;                      |
```

[expression] This syntax is not currently implemented and there is no easy way in current releases for the user to make use of this feature.

Version 24 will provide a linkage capability to user programs and thus allow all kinds of good stuff to be done inside the square brackets such as "BASIC", "APL", "PASCAL", "COBOL" &c...

"TEXTS" containing executable machine code

Version 23 of the SAM76 language interpreter allows the user to create and place in the text area modules of executable object code - obviously executable only in a particular machine. The SAM76 interpreter is able to differentiate between texts created using the "dt" define text function and machine code through the use of the "protection class" functions implemented in Version 23.

When the protection class of a particular text in the text area is set to be ^H80, either using the "cpc" change protection class function, or during the creation of this type of text, then the usual text manipulation functions have no effect on that text. Furthermore any type of fetch, that is to say explicit or implied, as well as active or neutral will cause the interpreter to execute a call to the first location after the text name which is embedded in the text itself.

At that time the H & L registers contain the actual adress of that particular point so that the user program may make use of this information as required to modify or relocate adresses in the object code of the "text". Additionally the zero/non zero flag bit is set if the function was invoked neutrally (non zero), and the A register contains a zero. The user program in the text is then executed and a return to the SAM76 language system is achieved by doing any type of return instruction; that is true if the user was careful in his stack manipulation.

Alternatively a return to the SAM76 scanner may be achieved by jumping to BEGIN+6 or to the appropriate vector location in the system subroutine vector table.

It is obvious that the variability of the location of texts in the text area requires that any object code to be executed using this system be written carefully to be absolutely position independent, either through appropriate use of position independent instructions, or through relocation program provided by the user. There are approximately fifty subroutines in the SAM76 interpreter that are accessible to the user to do a variety of tasks. These will be defined in detail in later publication. Access to these is in one of several ways - RST-6 and an index number, a call to the point in the vector table created at initialization at location ASV, or a call to BEGIN+9 followed by the index number of the desired subroutine.

An assembly language program designed to provide required pointers and to correctly organize a unit that may reside in the text area is shown in figure 1. This program may be assembled using some appropriate macro assembler and then injected into the text area.

Another way of creating an executable object code text is to use the "xu,dt" Xperimental User define text function which is defined as follows:

|xu,dt,n,x1,x2,...,xn| Execution of this function creates in the text area a text with a protection class of ^H80.

This text contains in successive locations following its name the binary equivalent of the numbers symbolized in the current "X" base by x1,x2,...,xn. This text is intended to contain executable machine code, and execution of %ft,n, ... / or %n, ... / &c. will cause a "CALL" to be made to the location containing "x1".

Examination of an executable object code text may be achieved by use of the "xu,ft" Xperimental User fetch text function:

|xu,ft,n,sØ|

The value of this function is the binary contents of the text whose name is symbolized by "n", each binary value is preceded by the delimiter symbolized by "sØ" and is represented in the current "X" base.

Disk Functions

There are a number of disk functions in the CP/M compatible overlay which were not mentioned in the previous update. These will be revealed from time to time. In particular the next update will describe the functions used to acces large files which are not in the SAM76 format.

I/O steering

A future update will describe how you may steer or switch data flow in and out of the SAM76 functions to a variety of peripheral devices without having to go back to the monitor. Such exotic assignments as Console to Punch, or Disk to Console are feasible with a family of "Select Input and Select Output" functions.

This new enhanced version is available on paper tape or TDL cassette as follows for a postpaid price of $8.00 - ($2.00 credit will be given to original purchaser of earlier version returned with order); POLY-88 version for 8080 only on cassette $8.00; CPM version on standard diskette $15.00.

Availability

Addresses in table below are all in HEX.

Type	ASV	VAR	TAM	START	END
Z80 High	0100	0400	7FFF	8000	9FFF
8080 High	0100	0400	7FFF	8000	A3FF
Z80 Low	0100	3400	FFFF	1000	2FFF
8080 Low	0100	3400	FFFF	1000	33FF
8080 Poly-88		4800	5000	FFFF	2000	43FF

Notes: VAR = START - OPV; all version with ASV at 0100 are CPM compatible; Z80 versions make use of RST0 thru RST6 while processor is running and restores RST0 on exit; TAM=FFFF is theoretical maximum reduced by size of operating system used; CPM diskette contains low versions; the disk overlay is at C00, and other overlays are at 400 and 800; it is possible in the CPM system to run programs which are smaller than 2K without affecting the SAM76 processor.

Text book describing the language in simple terms is available for a price of $12 - postpaid. (Specify bound or loose leaf). System loader booklet appropriate to object code format is $2.

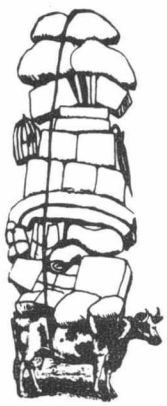
Source for CP/M Disk functions

The source program for the disk functions as implemented in the CP/M version will be published in Dr. Dobbs Journal, however copies may be purchased for the implementation of the moment - no guarantee that it will match the object code furnished - but to serve as a model for implementation in other disk systems; price is $10.00 postpaid; available as xerox printed copy or in machine readable form on standard diskette or paper tape [TDL macro format].

SAM76 Inc., Box 257, R.R.1, Pennington, N.J., 08534
Phone (609) - 466 - 1129/1130

```
.RADIX  8       ; added by editor
;
;
.TITLE  /Sample Program/
;
Z80=0
RSTINT=1
$COM=000400     ; ^h100
;
;******** start of boiler plate ******
.XLIST
;
;[DEUQ]
;
.SYN    RC,JRQC
.SYN    RZ,JRQZ
.SYN    RNC,JRQNC
.SYN    RNZ,JRQNZ
.SYN    RM,JRQM
;
;
;[DEUM]
;
.DEFINE LINKJ[ADR]=
[       JMP     $COM+3*ADR]
;
.DEFINE LINKC[ADR]=
[       CALL    $COM+3*ADR]
;
.DEFINE SKIP1=
[       .BYTE   376]
.DEFINE BTAB[A,B,C,ADR]=
[.WORD  ((B&7)<15)+((C&37)<10)+((A&37)<2)+((B&30)>3)
.IFNDEF ADR,[
        .WORD   0]
.IFDEF ADR,[
        .WORD   ADR]]
.DEFINE MTAB[TAB]=
[       LXI     H,TAB]
.DEFINE TTAB[M1,M2,ADR]=
[       .BYTE   (M1 & N137)
        .BYTE   (M2 & N137)
        .WORD   ADR]
.DEFINE DINX[REG]=
[       INX     REG
        INX     REG]
.DEFINE DDCX[REG]=
[       DCX     REG
        DCX     REG]
.DEFINE JCLA=
[       XRA     A]
.DEFINE DSUB[SBT]=
[       XCHG
        LHLD    SBT
        CALL    MSUB]
.DEFINE TSTA=
[       ORA     A]
.DEFINE MJST2[PTR]=
[       LHLD    PTR+2
        XCHG
        LHLD    PTR
        CALL    MST2]
.DEFINE MBTEST[TEMP]=
[       LDA     TEMP
        TSTA]
.DEFINE MMOVE[PTR]=
[       LHLD    PTR+2
        XCHG
        LHLD    PTR]
.DEFINE SKIP2[REG]=
[.BYTE  001![[[REG]&6]<3]]
;
.DEFINE MJSSR[TO,FRM,FRME]=;def
[       LHLD    FRM
        MOV     B,H
        MOV     C,L
        LHLD    TO
        XCHG
        LHLD    FRME
        CALL    MSHR
        SHLD    FRME]
;
;
;[DEUZ1]
;
.SYN RET,JRQ
.IFE Z80,[
.SYN JMP,.JMPR
.SYN JNC,.JRNC
.SYN JC,.JRC
.SYN JNZ,.JRNZ
.SYN JZ,.JRZ
;
.DEFINE MJSSA[FROM,TO,LNGTH]=
[       LXI     B,LNGTH
        LXI     H,FROM
        LXI     D,TO
;
        CALL    MSSA]
.DEFINE .DJNZ[ADR]=
[       DCR     B
        JNZ     ADR]
;
.DEFINE MDSBCD=
[       MOV     A,L
        SUB     E
        MOV     L,A
        MOV     A,H
        SBB     D
        MOV     H,A]]
;
;
.IFG Z80,[
.SYN JMPR,.JMPR
.SYN JRNC,.JRNC
.SYN JRC,.JRC
.SYN JRNZ,.JRNZ
.SYN JRZ,.JRZ
;
.DEFINE MJSSA[FROM,TO,LNGTH]=
[       LXI     B,LNGTH
        LXI     H,FROM
        LXI     D,TO
        LDIR]
.DEFINE .DJNZ[ADR]=
[       DJNZ    ADR]
.DEFINE MDSBCD=
[       JCLA
        DSBC    D]]
;
;
$$SAE=  ^D1
$$SAS=  ^D2
$$SWA=  ^D3
$$SRB=  ^D4
$$SER=  ^D5
$$SLK=  ^D6
$$SRXQ= ^D7
$$SGT=  ^D8
$$SXX=  ^D9
$$SRQQ= ^D10
$$SNQ=  ^D11
$$SRA=  ^D12
$$SRVA= ^D13
$$SHK=  ^D14
$$SPK=  ^D15
```

```
$$SLF=  ^D16
$$RNA=  ^D17
$$RNH=  ^D18
$$SAT=  ^D19
$$SFA=  ^D20
$$SDS=  ^D21
$$SDD=  ^D22
$$SWS=  ^D23
$$SAN=  ^D24
$$SFS=  ^D25
$$SGV=  ^D26
$$SSB=  ^D27
$$SPA=  ^D28
$$SRQ=  ^D29
$$SPL=  ^D30
$$RAC=  ^D31
$$RAD=  ^D32
$$RAY=  ^D33
$$RAZ=  ^D34
$$STO=  ^D35
$$STI=  ^D36
$$STL=  ^D37
$$SPR=  ^D38
$$SHR=  ^D39
$$SGQ=  ^D40
$$SXB=  ^D41
$$SXC=  ^D42
$$SUP=  ^D43
$$SMN=  ^D44
$$SDV=  ^D45
$$SUB=  ^D46
$$SPO=  ^D47
$$SRI=  ^D48
$$SXL=  ^D49
$$SLD=  ^D50
.LIST
;
;******** end of boiler plate ********
;
;[GUIDE]
;
; special commands generally have the following form:
;
;       JSxx    Means   CALL    MSXX
;
;       JRxx    Means   JMP     MRXX
;
;       Usually one can return to the scanner when in a
;       function program (with no intervening PUSHes)
;       by doing a JMP to any of the "MSXX" type labels.
;
;       The MRXX type labels always return
;       to the scanner whether called or jumped at.
;
;
.PAGE
;***** Pointer to Text Area End *****
;
TALsym= 032236  ; ^h349E ; CP/M
;
;
;***** Arbitrary Load Point *****
;
ENTLOC= 040000  ; ^h4000
;
;
;user program specials
;
.DEFINE .DIMEN[ARG]=
[ .WORD [[[ARG]<^D8]&177400] ! [[[ARG]>^D8]&377]]
;
;
;***** Origin statement for Loading *****
;
        .LOC    ENTLOC
;
;*****  Once only to move in place *****
;
        LXI     H,TALsym;change to suit
        PUSH    H
        MOV     E,M
        INX     H
        MOV     D,M
        LHLD    TRPNT
        MOV     C,L
        MOV     B,H
        LXI     H,LOCTOP
;
MOVE:   MOV     A,M
        STAX    D
        DCX     H
        DCX     D
        DCX     B
        MOV     A,C
        ORA     B
        JNZ     MOVE
        POP     H
        MOV     M,E
        INX     H
        MOV     M,D
        JMP     0; or as desired
;
;
;text format position independent code
;
;*****  REVERSE POINTER and NAME *****
;
TRPNT:  .DIMEN  LOCPFN+2 - .
TNAME:  .ASCII  PROGRAM
;
;*****  OBJECT CODE SECTION *****
;normal entry point
LOCGOT:
;
;address of "locgot" should be in H & L
;
;
;***** Position Independence *****
;
STRT:   PUSH    H
        LXI     B,TABLE-LOCGOT
        DAD     B
        XCHG
        POP     H
        LXI     B,LOOP-STRT
        DAD     B
;
;
;***** SAM76 Housekeeping *****
;
        LINKC   $$SER
;
;*****  Program Loop *****
;
LOOP:   XCHG
        MOV     A,M
        TSTA
        JRQZ
        MOV     C,A
        LINKC   $$SWA
        INX     H
        XCHG
        PCHL
;
;
TABLE:  .BYTE   74
        .ASCII  program
        .BYTE   76
        .BYTE   0
;
;
;*****  Text Header Information *****
;
LOCPFI: .WORD   0
LOCPFF: .DIMEN  LOCPFI-LOCGOT
LOCPFN: .BYTE   LOCGOT-TNAME
        .BYTE   200     ; ^h80
;
;
LOCTOP:
;
;
        .END    ;editor added
;
;
```



Two Additions to the Z-80 RAM Test

BY RONALD DERYNK AND BOCK W. LEE

Dear Dr. Dobb's: *Received: 78 Aug 14*

A couple of months ago I was just completing construction of an S.D. Sales Expandoram 16K memory when I saw John MacDougall's Z-80 RAM TESTER program in DDJ *Volume 3, Issue 2. This program appeared to be just what I needed to check out the new memory in my Z-80 system. As I studied the listing I noticed that John used three subroutines from the TDL ZAPPLE monitor. Unfortunately he did not give any indication as to what these routines were supposed to do. Since I do not have the ZAPPLE montior, I almost gave up trying to get the memory tester to work in my system.*

However, because the program was just what I needed I decided to see if I could figure out what each of the ZAPPLE subroutines was intended to accomplish. The results of this work are included with this letter along with the routines developed to take their place. I have also enclosed a memory dump (note my version of the program is assembled to run at location 0100H) and a sample run.

I would like to suggest that anyone submitting programs for publication which utilize montior routines also include a brief description of the purpose and register usages of these routines. This will make it much easier for people not possessing a particular monitor, such as ZAPPLE, to adapt these programs to run on their machine.

Yours truly,
Ronald Derynk
#28,336 Rundlehill Dr., N.E.
Calgary, Alberta
Canada T1Y 2Y2

First, the 3 ZAPPLE monitor routines will be discussed. The descriptions were arrived at by determining what was required to make the Z-80 RAM Tester program work and not be studying the ZAPPLE monitor. Therefore the descriptions are probably incomplete.

TDL ZAPPLE routine at F4FF

Purpose: Gets the start and end address from the user and returns with HL=start address and DE=end address.

TDL ZAPPLE routine at F561

Purpose: Increments the current test address contained in HL then checks to see if HL is greater than the end address contained in DE. If HL is less than DE, this routine returns to the caller. If HL is greater than DE, the stack pointer is incremented twice before returning. Incrementing the stack pointer effectively pops the return address off the stock. Therefore control is returned to the program which called the subroutine that contained the call to F561.

TDL ZAPPLE routine at F470

Purpose: On entry HL=address that has a defective memory location, and A=the test pattern. This routine prints the address of the bad memory location.

The following subroutines are designed to replace the TDL ZAPPLE monitor routines required by the Z-80 RAM TESTER program written by John MacDougall that appeared in Vol. 3, No. 2 of *Dr. Dobb's Journal*, page 42.

GETAD replaces the TDL routine called F4FF

PRTER replaces the TDL routine called F470

CHKEND replaces the TDL routine called F561

FINUP replaces the jump to the TDL monitor with a jump to my own Z-80 MONITOR, but first prints a message indicating that the test is complete.

The following listing references 5 subroutines which are part of my Z-80 MONITOR program. The purpose of each of these is described below:

IN2AD (C27E) on entry HL points to a message to be printed on exit HL=start add. DE=end add. entered by the user. (see DSPMSG for message format) IN2AD prompts the user by printing START and END as required.

DSPMSG (C4E6) on entry HL points to a message to be printed. The first byte of the message is a character count of the number of characters to be printed by DSPMSG. The rest of the message is stored immediately after the count in ASCII.

ADDOUT (C545) outputs the contents of HL as 4 hex digits.

PUTBIN (C1BD) outputs the contents of C as 8 binary digits. Bit 7 is printed first and bit 0 last.

SYSDSP (ED03) outputs the contents of A to the system terminal (TTY or VDM). SYSDSP also interprets any control characters and takes appropriate action depending on terminal type.

In general I have stuck to the ZILOG mnemonics except for some pseudo ops and address references. I don't believe these will cause any problems for those familiar with the Z-80 but you may have to modify the program slightly to conform to your assemblers syntax. At any rate the machine langauge codes are directly useable.

```
                ;THIS ROUTINE PRINTS THE TITTLES AND HEADINGS PLUS
                ;GETS THE START AND END ADDRESS FROM THE CONSOL
                ;AND RETRNS WITH HL=START AND DE=END
                ;
Ø177 21 86 Ø1   GETAD   LDHL,ADMSG1  PRINT TITTLE AND
Ø17A CD 7E C2           CALL IN2AD   GET START AND END ADDRESSES
Ø17D E5                 PUSH HL
Ø17E 21 94 Ø1           LDHL,ADMSG2  PRINT HEADINGS
Ø181 CD E6 C4           CALL DSPMSG
Ø184 E1                 POP HL       RESTORE REG.
Ø185 C9                 RET          AND RETUURN TO CALLER
Ø186 ØD 8D CD   MSG1    ASCII "(CR)MEMORY TEST(CR)"
Ø189 C5 CD CF
Ø18C D2 D9 AØ
Ø18F D4 C5 D3
Ø192 D4 8D

Ø194 15 C1 C4   MSG2    ASCII "ADD   TEST BIT   MEM(CR)"
Ø197 C4 AØ AØ
Ø19A AØ D4 C5
Ø19D D3 D4 AØ
Ø8AØ C2 C9 D4
Ø1A3 AØ AØ AØ
Ø1A6 CD C5 CD
Ø1A9 8D

                ;THIS ROUTINE IS CALLED WHEN A BAD MEMORY CELL IS
                ;FOUND. IT PRINTS THE ADDRESS OF THE BAD CELL (IN
                ;HL ON ENTRY). THE TEST PATERN (IN A) AND THE
                ;MEMORY CONTENTS ARE PRINTED IN BINARY FORMAT
                ;
Ø1AA F5         PRTER   PUSH AF      SAVE REG.
Ø1AB CD 45 C5           CALL ADDOUT  PRINT CURR. ADD IN HL
Ø1AE C5                 PUSH BC
Ø1AF 4F                 LDC,A        PUT TEST PATTERN IN C
Ø1BØ CD BD C1           CALL PUTBIN  AND PRINT IT IN BINARY
Ø1B3 3E AØ              LDA,"SP"     OUTPUT AN ASCII SPACE
Ø1B5 CD Ø3 ED           CALL SYSDSP  HERE
Ø1B8 CD Ø3 ED           CALL SYSDSP  AND ONCE MORE
Ø1BB 4E                 LDC,(HL)     GET THE BAD MEMORY DATA
Ø1BC CD BD C1           CALL PUTBIN  AND PRINT IN BINARY
Ø1BF 3E 8D              LDA,"CR"     NOW WE NEED A CARR. RET.
Ø1C1 CD Ø3 ED           CALL SYSDSP  SEND IT
Ø1C4 C1                 POP BC       RESTORE
01C5 F1                 POP AF       REGS AND
Ø1C6 C9                 RET          RETURN TO CALLER

                ;THIS ROUTINE INCREMENTS THE CURRENT ADDRESS AND
                ;CHECKS TO SEE IF THE NEW ADDRESS IS GREATER THAN
                ;THE END ADDRESS. IF IT IS THE RETURN ADD. IS
                ;POPPED OFF THE STACK.
                ;
Ø1C7 23         CHKEND  INC HL       INCREMENT CURR ADD IN HL
Ø1C8 D5                 PUSH DE      SAVE END ADD
Ø1C9 37                 SCF          CLEAR THE
Ø1CA 3F                 CCF            CARRY FLAG
Ø1CB EB                 EX DE,HL     SUBTRACT CURR. ADD.
Ø1CC ED 52              SBC HL,DE      FROM END ADD.
Ø1CE EB                 EX DE,HL     RESTORE CURR. AND
Ø1CF D1                 POP DE         END ADDRESSES
Ø1DØ FØ                 RET P        RETURN UNLESS PAST END
Ø1D1 33                 INC SP       FAKE STACK TO POINT TO
Ø1D2 33                 INC SP         SUBROUTINE ON NEXT HIGHER
Ø1D3 C9                 RET              LEVEL AND RETURN THERE

Ø1D4 21 DD Ø1   FINUP   LDHL,ADMSG3  PRINT END OF TEST
Ø1D7 CD E6 C4           CALL DSPMSG    MESSAGE AND
Ø1DA C3 ØF CØ           JP MON           GO TO MONITOR PROG.

Ø1DD ØD 8D C5   MSG3    ASCII "(CR)END OF TEST(CR)"
Ø1EØ CE C4 AØ
Ø1E3 CF C6 AØ
Ø1E6 D4 C5 D3
Ø1E9 D4 8D
```

Example run of Z-80 MEMORY TEST program. Underscores indicate data typed into the computer. A 1K block of memory starting at A000 is tested. Note that location A400 is the first address in an unpopulated memory block and it therefore always reads out as being all 1's. Also note that one of the 2102's in this memory block has a bad location (A146) which inverts the data stored there. Printing the test pattern and memory data in binary makes it obvious which chip is bad (bit 7 in this case). I have used the program to check all my memories and discovered that there were a couple of bad IC's which needed replacement. After this was done a lot of "funny" things that had been happening to my programs stopped occurring. These bad IC's were not part of the S.D. Sales Expandoram, the 8K dynamic RAMs for this kit all checked out perfectly.

Dear Dr. Dobb's: *Received: 78 Sept 18*

You published a Z-80 memory test program by J. MacDougall in the February issue which referenced an article of J. Butterfield (August 1977) and an algorithm of Knaizuk and Hartmann (IEE Transactions on Computers, April 1977). While MacDougall's program ran faster than any other I had used before, it was still an "over implementation" of the K & H algorithm in that a total of 14 x 2↑N memory accesses were employed per pass as compared to 4 x 2↑N.

Enclosed is an assembly listing of a program, Z80MT, V1.1, which implemented the K & H algorithm with only 6 x 2↑N memory accesses per complete pass. (A later program requiring exactly 4 x 2↑N memory accesses was written, but the difference in speed was not significant and the program was not fully developed.) In addition, the program included provisions for repetitively cycling over the memory test range with a cyclic, "walking" test byte, in order to enhance its power for testing cross-over type faults within bytes. A control B entered at the console device will terminate the multipass cyclic mode and return the program to "Begin," awaiting specification of another memory test range. A control C

entered at the console will abort and cause the program to return to the Zapple monitor.

In completion of each pass, "Pass Completed. # Errors = xx" is printed, where xx is the number of errors in Hex (modulo 256) encountered during the pass. Also, as each error byte is encountered during a pass, the address location (4 Hex) or the error byte and the result of exclusive or-ing the test byte and the error byte is printed. The latter can be effectively used to identify the error bits within the byte.

Bock W. Lee
150 East Standard Ave.
Richmond, CA 94804

```
                        .TITLE  'Z-80 K & H MEMORY TEST PROGRAM V 1.0'
                        .SBTTL  'B. W. LEE, SEPT 7,1978'
                        ;
                        ;
                        ;                  ++++++++++++++
                        .RADIX 16       ;+   Z80MT     +
                        ;                  +   V 1.1    +
                        ;                  ++++++++++++++
                        ;
                        ;
                        ;
                        ;THIS PROGRAM IS AN IMPLEMENTATION OF THE
                        ;K&H OPTIMAL ALGORITHM FOR TESTING STUCK-
                        ;AT-ONE OR STUCK-AT-ZERO TYPES OF FAULTS
                        ;IN IC MEMORY SYSTEMS. IT IS WRITTEN IN TDL'S
                        ;Z-80 ASSEMBLY LANGAUGE. FAULTS IN THE
                        ;ADDRESSING,DECODING,AND MEMORY ARRAY AREAS
                        ;ARE DETECTED IN ONLY 6*(2^N) MEMORY ACCESSES.
                        ;
                        ;
                        ;EQUATES:
F597                    LBYTE   =       0F597   ;PRINT BYTE IN 2 HEX
0000                    W0      =       00      ;TEST BYTE W0
F576                    HILOX   =       0F576   ;INC HL & CHK FOR END
F514                    EXLF    =       0F514   ;GET & PRINT 2 ADDRS
F485                    LFADR   =       0F485   ;PRINT ADDR IN 4 HEX
F009                    CO      =       0F009   ;OUTPUT TO CONSOLE
F012                    CSTS    =       0F012   ;CONSOLE STATUS
F519                    CRLF    =       0F519   ;CR & LF TO CONSOLE
000D                    CR      =       0D
000A                    LF      =       0A
F455                    TOM1    =       0F455   ;PRINT MSG & CSTS
                        ;
                        ;
                        ;
                        ;
0000' C3 0030'          ENTRY:  JMP     BEGIN   ;PROG BEGIN HERE
                        ;
                        ;
0003' 0D0A0000          MSGON:  .BYTE   CR,LF,W0,W0
0007' 5A38304D54                .ASCII  'Z80MT V'
000E' 312E30                    .ASCII  '1.0'
000E                    MSGONL  =       .-MSGON
                        ;
0011' 0D0A0000          MPASS:  .BYTE   CR,LF,W0,W0
0015' 5041535320                .ASCII  'PASS COMPLETED.
0025' 2320455252                .ASCII  '# ERRORS = '
001F                    MPASSL  =       .-MPASS
                        ;
                        ;
                        ;
                        ;
0030' 21 0030'          BEGIN:  LXI     H,BEGIN ;RET HERE
0033' E5                        PUSH    H
0034' 21 0003'                  LXI     H,MSGON
0037' 060E                      MVI     B,MSGONL
0039' CD F455                   CALL    TOM1    ;PRINT SIGN-ON
003C' CD F519                   CALL    CRLF
003F' 0E24                      MVI     C,'$'
0041' CD F009                   CALL    CO      ;PRINT $
                        ;
                        ;
0044' 0E02              GPAR:   MVI     C,02    ;2 PARAMETERS
0046' CD F514                   CALL    EXLF    ;GET THEM
0049' 22 00A1'                  SHLD    START   ;STORE START
004C' ED53 00A3'                SDED    END     ;STORE END
                        ;
                        ;
0050' 0E00                      MVI     C,W0    ;START WITH TB=W0
0052' AF                MTEST:  XRA     A       ;CHEAP ZERO
0053' 32 00A6'                  STA     ERROR   ;CLR ERROR CT.
0056' 3E02                      MVI     A,02    ;2 PARTS PER PASS
0058' 32 00A5'                  STA     PART    ;SAVE IT
005B' CD 00A7'          MTLOP:  CALL    STUFF   ;STUFF MAJ ALL OVER
005E' 3E02                      MVI     A,02    ;SET 2 AS MINOR SET
0060' CD 00B0'                  CALL    STUFM   ;STUFF MINOR SET
0063' 3E02                      MVI     A,02    ;SET 2 AGAIN
0065' CD 00C4'                  CALL    CHECK   ;CK ALL OVER
0068' 3A 00A5'          PTCHK:  LDA     PART    ;WHERE ARE WE?
006B' 3D                        DCR     A       ;DEC PART
006C' 32 00A5'                  STA     PART    ;SAVE IT
006F' FE00                      CPI     00      ;FINISH THIS PASS?
0071' 280E                      JRZ     RECYCLE ;YES
0073' 3E01              CONT:   MVI     A,01    ;NO:SET 1 AS MINOR
0075' CD 00B0'                  CALL    STUFM   ;STUFF MINOR SET
0078' 79                        MOV     A,C     ;GET MAJ TB
0079' 2F                        CMA             ;COMPL IT
007A' 4F                        MOV     C,A     ;REPLACE AS MAJOR
007B' AF                        XRA     A       ;SET 0 AS MINOR
007C' CD 00C4'                  CALL    CHECK   ;CHK WITH INV MAJ
007F' 18DA                      JMPR    MTLOP   ;DO REST OF 2ND PT.
                        ;
                        ;
0081'                   RECYCLE:
0081' 79                        MOV     A,C
0082' 32 00A5'                  STA     PART    ;TEMP'Y SAVE INVERTED TB
0085' 21 0011'                  LXI     H,MPASS
0088' 061F                      MVI     B,MPASSL
008A' CD F455                   CALL    TOM1    ;PRINT PASS MSG.
008D' F5                        PUSH    PSW     ;SAVE AF
008E' 3A 00A6'                  LDA     ERROR   ;GET ERROR COUNT
0091' CD F597                   CALL    LBYTE   ;PRINT AS 2 HEX
0094' F1                        POP     PSW     ;RECOVER AF
0095' D8                        RC              ;RET TO 'BEGIN'
                        ;NOTE IN THE ABOVE: ENTERING CONTROL C AT
                        ;CONSOLE WOULD HAVE CAUSE AN 'ABORT' TO
                        ;ZAPPLE WITHOUT PRINTING 'ERROR COUNT'.
                        ;ENTERING CONTROL B OR CONTROL A WOULD
                        ;HAVE CAUSE AN 'ABORT' TO 'BEGIN'AFTER
                        ;FIRST PRINTING THE 'ERROR COUNT'.
                        ;
0096' 3A 00A5'                  LDA     PART    ;REC'VER INVERTED TB
0099' CB27                      SLAR    A       ;SHIFT TB WITH 0
009B' 2801                      JRZ     ..TB    ; OVER IF END OF SHIFT CYCLE
                        ;
009D' 2F                        CMA             ;UN-INVERT TB
009E' 4F                ..TB:   MOV     C,A     ;INSTALL NEW TB
009F' 18B1                      JMPR    MTEST   ;BACK FOR ANOTHER PASS
                        ;
                        ;
00A1'                   START:  .BLKW   1       ;RES FOR START ADDR
00A3'                   END:    .BLKW   1       ;RES FOR END ADDR
00A5'                   PART:   .BLKB   1       ;RES FOR PART
00A6'                   ERROR:  .BLKB   1       ;RES FOR ERROR COUNT
                        ;
                        ;
00A7' CD 00EA'          STUFF:  CALL    STASTO  ;LOAD ADDR
00AA' 71                DOIT:   MOV     M,C     ;STUFF MAJOR TB
00AB' CD F576                   CALL    HILOX   ;INC & CHK HL
00AE' 18FA                      JMPR    DOIT    ;NOT FINISH
                        ;
                        ;
00B0' CD 00EA'          STUFM:  CALL    STASTO  ;LOAD ADDR
00B3' 47                        MOV     B,A     ;MINOR PS'N IN B
00B4' FE00                      CPI     00      ;SET 0 IS MINOR?
00B6' 2005                      JRNZ    HIL     ;NO
00B8' 79                MINOR:  MOV     A,C     ;COMP MAJ TB
00B9' 2F                        CMA
00BA' 77                        MOV     M,A     ;STUFF MIN TB
00BB' 0603                      MVI     B,03    ;SET FOR NEXT MINOR
00BD' CD F576           HIL:    CALL    HILOX   ;INC & CHK HL
00C0' 10FB                      DJNZ    HIL     ;TILL B=0
00C2' 18F4                      JMPR    MINOR   ;TIME TO STUFF MINOR
                        ;
                        ;
00C4' CD 00EA'          CHECK:  CALL    STASTO  ;LOAD ADDR
00C7' 47                        MOV     B,A     ;MINOR PS'N IN B
00C8' FE00                      CPI     00      ;SET 0 IS MINOR?
00CA' 2007                      JRNZ    MAJR    ;NO
00CC' 79                MINR:   MOV     A,C     ;PREPARE COMPL
00CD' 2F                        CMA
00CE' BE                        CMP     M       ;READ & COMPARE
00CF' 0603                      MVI     B,03    ;SET FOR NEXT MINOR
00D1' 1802                      JMPR    CKEND
00D3' 79                MAJR:   MOV     A,C
00D4' BE                        CMP     M       ;COMPARE
00D5' C5                CKEND:  PUSH    B
00D6' C4 00F2'                  CNZ     ERR     ;PRINT ERROR IF NEC'Y
00D9' C1                        POP     B
00DA' CD F012                   CALL    CSTS    ;CHK CONSOLE
00DD' B7                        ORA     A
00DE' 2007                      JRNZ    ABORT   ;ABORT IF S'THING
00E0' CD F576                   CALL    HILOX   ;CONT. IF NOTHING
00E3' 10EE                      DJNZ    MAJR    ;MAJOR TILL B=0
00E5' 18E5                      JMPR    MINR    ;THEN MINOR
                        ;
00E7' F1                ABORT:  POP     PSW     ;RET ONE STEP BACK
00E8' 1897                      JMPR    RECYCLE ;ABORT THRU RECYCLE
                        ;
                        ;
00EA' 2A 00A1'          STASTO: LHLD    START
00ED' ED5B 00A3'                LDED    END
00F1' C9                        RET
                        ;
                        ;
00F2' F5                ERR:    PUSH    PSW     ;SAVE TB
00F3' 3A 00A6'                  LDA     ERROR
00F6' 3C                        INR     A
00F7' 32 00A6'                  STA     ERROR
00FA' CD F485                   CALL    LFADR   ;PRINT ERROR LOC
00FD' F1                        POP     PSW     ;RECOVER TB
00FE' AE                        XRA     M       ;MARK DIFF BITS
00FF' CD F597                   CALL    LBYTE   ;PRINT DIFF AS 2 HEX
0102' C9                        RET
                        ;
                        ;
0000'                   .END    ENTRY
```

The Mystery of Ackermann's Function

BY MIKE GABRIELSON
Box 2692
Stanford, Calif. 94305

I first encountered Ackermann's function several years ago while reading D. W. Barron's *Recursive Techniques in Programming*. Page five states:

> It is possible for both types of recursion [recursive definition and recursive use] to be present simultaneously. The most notorious example of this is Ackermann's function:
>
> A(m, n) = [m=0 → n+1,
> n=0 → A(m−1, 1),
> A(m-1, A(m,n−1))]
>
> (Remember that in evaluating a conditional expression the conditions are tested in turn, so that the second and third lines will only be reached if m≠0.) This is a deceptively simple function; the reader with five munutes to spare may care to compute A(2,3) from the definition.

Ackermann's function has been the focus of a surprising amount of attention in the computer field. The excellent May 1977 issue of *Computer* (devoted to stack machines that month) contained an article by R.P. Blake titled "Exploring a Stack Architecture." Blake reports:

> Because of its strongly recursive nature, Ackermann's function has long been used in the theory of algorithms and automata. It is of importance in computer design because it can serve as a standard method for evaluating an implementation of procedure calling. Sundblad used Ackermann's function to explore various implementations of Algol, PL/I, and Simula on the 360/75 and the CDC 6600. Wichmann has now reported on the performance of Ackermann's function on over 50 [hardware/software systems].

The notation used by Barron was taken from McCarthy. The following Algol W version may be a little clearer:

```
INTEGER PROCEDURE ACKERMANN (INTEGER
  VALUE M, N);
IF M = 0 THEN N+1
ELSE   IF N = 0 THEN ACKERMANN (M−1, 1)
       ELSE ACKERMANN (M−1, ACKERMANN (M,
         N-1) );
```

As Barron suggested, I took five minutes to laboriously calculate A(2,3) using pencil and paper. Unfortunately,the book didn't contain the answer. Unable to confirm my calculations, I wrote my result in the margin and forgot about it.

More recently, I implemented Ackermann's function on my 6800 system as an exercise in recursive programming:

m \ n	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10	11
2	3	5	7	9	11	13	15	17	19	21
3	5	13	29	61	125	253	509	1021	2045	4093
4	13									

Table 1: Ackermann's Function (decimal)

m \ n	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	A
1	2	3	4	5	6	7	8	9	A	B
2	3	5	7	9	B	D	F	11	13	15
3	5	D	1D	3D	7D	FD	1FD	3FD	7FD	FFD
4	D									

Table 2: Ackermann's Function (hexadecimal)

```
*************************************************
* Ackermann's Function - Mike Gabrielson - 12/4/77
* Accepts:  A = m, X = n
* Returns:  X = Ackermann(m, n)
*************************************************
ACKERMANN TSTA            m = 0?
          BNE RECALL      no
          INX             yes, return n+1
          RTS
RECALL    CPX #0          m = 0.  n = 0?
          BNE RERECALL    no
          DECA            yes, A := m-1
          LDX #1          X := 1
          BSR ACKERMANN   (m-1, 1)
          RTS
RERECALL  PSHA
          DEX             X := n-1
          BSR ACKERMANN   (m, n-1)
          PULA
          DECA            A := m-1
          BSR ACKERMANN   (m-1, Ackermann(m, n-1))
          RTS
```

I was able to confirm my hand-calculated answer from years earlier, but soon discovered that the function required huge amounts of stack space. Limited to 16 kilobytes for the stack, my system was only able to generate the results shown in Table 1. Table 2 contains the same results, but in hexadecimal; notice how it makes the patterns obvious.

It is easy to extrapolate values for $n > 9$. But what does Ackermann's function return when m is greater or equal to 4?

PET Basic Renumber

BY BILL SEILER
1717 Woodland #202
Palo Alto, CA 94303

PET Renumber occupies 384 bytes at the end of an 8K PET RAM memory. The POKE 134,128 and POKE 135,30 statements set the top end of PET BASIC working memory below the PET Renumber program. This protects PET Renumber from BASIC. The SYS7808 transfers control to the PET Renumber Program.

PET Renumber uses 1K of screen memory as a storage buffer for the old line numbers. It first copies the old line numbers to the screen buffer. On this first pass PET Renumber generates the new line numbers and inserts them at the beginning of each line.

On the second pass, PET Renumber fixes the THEN, GOTO, GOSUB, and RUN line number references. It looks for these tokens followed by an ASCII number. If there is a number, it searches the old line number in the screen buffer and generates its new number. If the number is found the new number is inserted after the token.

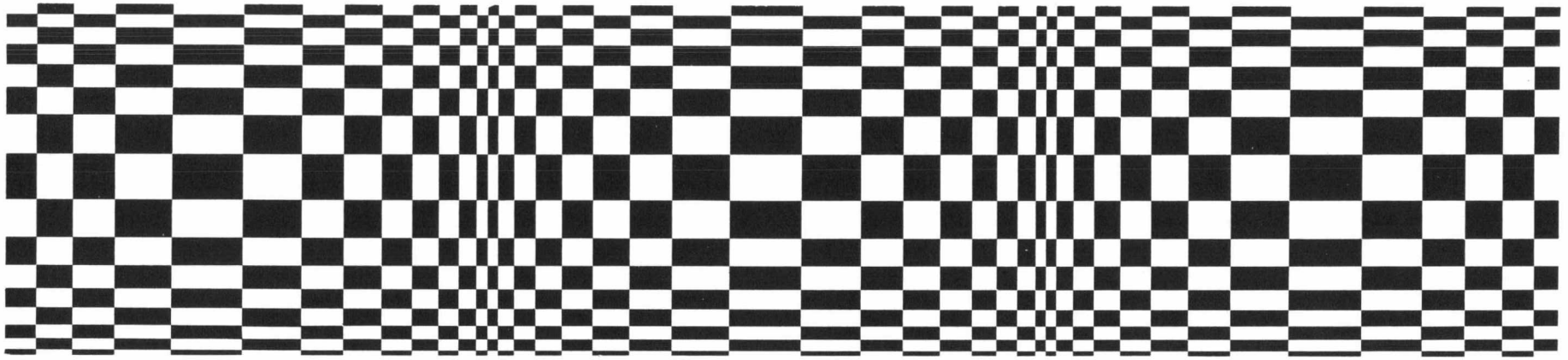
Sometimes while PET Renumber is inserting new line numbers which are larger or smaller, the source is expanded or compressed. This is done one byte at a time from where the number is being inserted.

After the second pass is completed PET Renumber jumps back into PET BASIC. It goes into a routine to fix up the line links and then goes back into BASIC.

Limitations

Pet Renumber uses PET's 1K TV screen RAM as a buffer. The old line numbers take two bytes each. Therefore, the screen buffer can only hold 511 old line numbers. Also PET Renumber occupies 384 bytes of the 7167 bytes free to the user. This limits the size of the program after renumbering to 6783 bytes.

```
                    USING  PET  RENUMBER  3.Ø

LOAD    PET RENUMBER
TYPE    "NEW"
POKE  134, 128  : POKE 135, 3Ø
LOAD     The program to be renumbered
TYPE     SYS 78Ø8
When cursor returns your program is renumbered
   Starting at 1ØØ  step by  1Ø


To change the start number
   POKE 78Ø9 , LLL     Lo   BYTE    Start # in
   POKE 7813 , HHH     Hi   BYTE    2 byte binary
TO change the step size
   POKE 7817, SSS      STEP SIZE <256
THEN DO       SYS78Ø8


ALL   THEN's, GOTO's, GOSUB's and RUN's will be renumbered.
  Also ON X GOTO's and ON X GOSUB's will be renumbered.
BRANCHES TO NON EXISTANT LINE NUMBERS WILL BE CONVERTED TO
BRANCHES TO LINE NUMBER 65535 (an illegal line number)


                    SAVING  PET  RENUMBER  3.Ø


LOAD    PET RENUMBER
LOAD AND RUN  MACHINE LANGUAGE MONITOR
TYPE    . S Ø1,PET RENUMBER 3.Ø,1E8Ø,2ØØØ
```



```
LINE # LOC     CODE       LINE

0002  0000              ;*********************************
0003  0000              ;
0004  0000              ;
0005  0000              ;  RENUMBER PET BASIC
0006  0000              ;
0007  0000              ;    (C) COPYRIGHT 1978
0008  0000              ;    BY B. SEILER
0009  0000              ;    ALL RIGHTS RESERVED
0010  0000              ;
0011  0000              ;
0012  0000              ;*********************************



0014  0000              BEGNUM=100
0015  0000              INCNUM=10
0016  0000              VARTAB=$007C
0017  0000              FACTO=$00B1
0018  0000              LINNUM=$0008
0019  0000              TXTTAB=$007A
0020  0000              TXTPTR=$00C9
0021  0000              INDEX1=$0071
0022  0000              INDEX2=$0073
0023  0000              STEP=$004D
0024  0000              BEGIN=$004E
0025  0000              NUMX=$0050
0026  0000              SCREEN=$8000


0028  0000              ;****** BASIC SUBROUTINES ******
0029  0000              CHRGET=$00C2            ;GETS CHR FROM TEXT
0030  0000              CHRGOT=$00C8            ;GET LAST CHAR
0031  0000              FINI=$C430              ;FIXES LINE LINKS
0032  0000              LINGET=$C863            ;MAKES ASCII INTO BIN
0033  0000              FLOATC=$DB1B            ;FLOAT A BIN #
0034  0000              FOUT=$DCAF              ;MAKE ASCII


0036  0000              ;*********************************
0037  0000              ;
0038  0000              ;
0039  0000              ; RENUMBER BASIC LINES
0040  0000              ;  STARTS AT BEGNUM
0041  0000              ;  AND NUMBERS BY INCNUM
0042  0000              ;  ALSO RENUMBERS THEN'S,RUN'S,
0043  0000              ;  GOTO'S,AND GOSUB'S.
0044  0000              ;  INCLUDING ON XX GOTO AND
0045  0000              ;  ON XX GOSUB STATEMENTS
0046  0000              ;
0047  0000              ;   ENTER BY SYS(7808),
0048  0000              ;   RETURNS TO BASIC
0049  0000              ;
0050  0000              ;


LINE # LOC     CODE       LINE

0051  0000              ;  DEFAULT :
0052  0000              ;   FIRST LINE NUMBER = 100
0053  0000              ;   STEP BY 10
0054  0000              ;
0055  0000              ;  FOR DIFFERENT START NUMBER
0056  0000              ;   POKE 7809,XXX  LO BYTE
0057  0000              ;   PE 7813,X  HI BYTE
0058  0000              ;
0059  0000              ;  FOR A DIFFERENT STEP
0060  0000              ;   POKE 7817,XXX  STEP SIZE <256
0061  0000              ;
0062  0000              ;
0063  0000              ;*********************************


0065  0000                      *=$1E80
0066  1E80              ;
0067  1E80  A9 64       START  LDA #<BEGNUM     ;DEFAULT START
0068  1E82  85 4E              STA BEGIN         ;  AT 100
0069  1E84  A9 00              LDA #>BEGNUM
0070  1E86  85 4F              STA BEGIN+1
0071  1E88  A9 0A              LDA #INCNUM       ;STEP BY 10
0072  1E8A  85 4D              STA STEP


0074  1E8C              ;FILL SCREEN BUFFER WITH OLD
0075  1E8C              ;  LINE NUMBERS AND RENUMBER
0076  1E8C              ;  THE BEGINNINGS ONLY
0077  1E8C              ;
0078  1E8C              SS=SCREEN-2
0079  1E8C  A9 FE       COPNUM LDA #<SS         ;SCREEN PTR - 2
```

```
LINE # LOC     CODE       LINE

0080  1E8E  85 73        STA INDEX2
0081  1E90  A9 7F        LDA #>SS
0082  1E92  85 74        STA INDEX2+1
0083  1E94  A5 7A        LDA TXTTAB      ;BEGINING OF TEXT
0084  1E96  85 71        STA INDEX1
0085  1E98  A5 7B        LDA TXTTAB+1
0086  1E9A  85 72        STA INDEX1+1
0087  1E9C  20 C1 1F     JSR SETFAC      ;SET NEW # CTR
0088  1E9F  A0 03     COP10  LDY #3      ;POINT AT LINE #
0089  1EA1  B1 71     COP20  LDA (INDEX1),Y  ;GET LINE #
0090  1EA3  91 73        STA (INDEX2),Y  ;SAVE IN BUFFER
0091  1EA5  B9 AE 00     LDA FACTO-3,Y   ;GET NEW #
0092  1EA8  91 71        STA (INDEX1),Y  ;REPLACE OLD
0093  1EAA  88           DEY
0094  1EAB  C0 01        CPY #1          ;POINTING AT LINK?
0095  1EAD  D0 F2        BNE COP20       ;NO-LOOP
0096  1EAF  B1 71        LDA (INDEX1),Y  ;YES-LAST LINE?
0097  1EB1  F0 10        BEQ COP80       ;YES-DONE!
0098  1EB3  20 E1 1F     JSR BUPX2       ;NO-BUMP PTR 2
0099  1EB6  AA           TAX             ;SAVE HI LINK IN X
0100  1EB7  88           DEY             ;POINT TO LO LINK
0101  1EB8  B1 71        LDA (INDEX1),Y  ;GET LOW
0102  1EBA  85 71        STA INDEX1      ;MAKE NEW PTR
0103  1EBC  86 72        STX INDEX1+1    ;  POINT AT NEXT LINE
0104  1EBE  20 CA 1F     JSR ADDSTP      ;GEN NEW NEW LINE #
0105  1EC1  D0 DC        BNE COP10       ;BRANCH ALWAYS
0106  1EC3              ;
0107  1EC3  A9 FF     COP80  LDA #$FF    ;MAKE LAST # BIGGEST
0108  1EC5  C8           INY
0109  1EC6  91 73        STA (INDEX2),Y  ;FINISH IN BUFFER
0110  1EC8  C8           INY
0111  1EC9  91 73        STA (INDEX2),Y
0112  1ECB  A5 7A     RENH   LDA TXTTAB  ;START CHRGET AT SOURCE
0113  1ECD  85 C9        STA TXTPTR
0114  1ECF  A5 7B        LDA TXTTAB+1
0115  1ED1  85 CA        STA TXTPTR+1
0116  1ED3  D0 03        BNE RN15        ;GET HI LINK
0117  1ED5  20 EB 1F  RN10   JSR GRAB    ;SKIP LO LINK
0118  1ED8  20 EB 1F  RN15   JSR GRAB    ;GET HI LINK
0119  1EDB  D0 03        BNE RN20        ;SKIP IF NOT ZERO
0120  1EDD  4C 30 C4     JMP FINI        ;FINISH IF ZERO
0121  1EE0  20 EB 1F  RN20   JSR GRAB    ;SKIP LO LINE #
0122  1EE3  20 EB 1F        JSR GRAB    ;SKIP HI LINE #
0123  1EE6  20 EB 1F  RN30   JSR GRAB    ;GET BASIC SOURCE
0124  1EE9  AA           RN35   TAX      ;IS THIS END OF LINE?
0125  1EEA  F0 E9        BEQ RN10        ;YES-LOOP
0126  1EEC              ;
0127  1EEC  A2 04     RN40   LDX #4      ;LOOK FOR 4 TOKENS
0128  1EEE  DD F5 1F  RN45   CMP TOKEN-1,X  ;IS THIS ONE
0129  1EF1  F0 05        BEQ RN50        ;YES-THEN FIX IT
0130  1EF3  CA           DEX             ;NO-TRY NEXT TOKEN IN TABLE
0131  1EF4  D0 F8        BNE RN45        ;NOT DONE-LOOP
0132  1EF6  F0 EE        BEQ RN30        ;TRIED ALL FORGET IT
0133  1EF8  A5 C9     RN50   LDA TXTPTR  ;YES-SAVE TEXT PTR
0134  1EFA  48           PHA
0135  1EFB  A5 CA        LDA TXTPTR+1
0136  1EFD  48           PHA
0137  1EFE  20 C2 00     JSR CHRGET      ;IS A # NEXT?
0138  1F01  B0 E6        BCS RN35        ;NO-LOOP
0139  1F03  20 63 C8     JSR LINGET      ;YES-MAKE A BIN #
0140  1F06  20 3A 1F     JSR FINUM       ;FIND OLD #, MAKE NEW #
0141  1F09  68           PLA             ;MOVE PTR TO # BEGIN
0142  1F0A  85 CA        STA TXTPTR+1
0143  1F0C  68           PLA
0144  1F0D  85 C9        STA TXTPTR
0145  1F0F  A0 00        LDY #0          ;SET INDEXS
0146  1F11  A2 00        LDX #0

LINE # LOC     CODE       LINE

0147  1F13  BD 01 01  RN60   LDA $0101,X  ;GET ASCII NEW #
0148  1F16  F0 0F        BEQ RN70        ;IF ZERO THEN DONE
0149  1F18  48           PHA             ;SAVE IT
0150  1F19  20 C2 00     JSR CHRGET      ;IS OLD CHR ASCII #?
0151  1F1C  90 03        BCC RN65        ;YES-SKIP TO REPLACE IT
0152  1F1E  20 71 1F     JSR MOVUP       ;NO-MOVE TEXT UP ONE
0153  1F21  68        RN65   PLA         ;GET SAVED NEW #
0154  1F22  91 C9        STA (TXTPTR),Y  ;REPLACE OLD
0155  1F24  E8           INX             ;NEXT ASCII IN NEW #
0156  1F25  D0 EC        BNE RN60        ;BRANCH ALWAYS
0157  1F27  20 C2 00  RN70   JSR CHRGET  ;ANY OLD ASCII LEFT
0158  1F2A  B0 08     RN75   BCS RN80    ;NO-FINISHED
0159  1F2C  20 93 1F  RN78   JSR MOVDWN  ;YES-MOVE TEXT DOWN ONE
0160  1F2F  20 C8 00     JSR CHRGOT      ;TEST NEW CHR IN OLD PLACE
0161  1F32  90 F8        BCC RN78
0162  1F34  C9 2C     RN80   CMP #',     ;IS THIS AN "ON"
0163  1F36  F0 C0        BEQ RN50        ;YES-DO THE REST
0164  1F38  D0 AF        BNE RN35        ;NO-THEN LOOP



0166  1F3A              ;FINDS OLD NUMBER IN BUFFER
0167  1F3A              ;  GENERATES A NEW LINE #
0168  1F3A              ;  PUT IT IN ASCII IN $0101
0169  1F3A              ;  ENDS IT WITH 00
0170  1F3A              ;  ENTER LINNUM=$0008 EQUAL
0171  1F3A              ;  TO OLD NUMBER
0172  1F3A              ;
0173  1F3A  20 C1 1F  FINUM   JSR SETFAC  ;BEGIN NUMBER
0174  1F3D  A9 00        LDA #<SCREEN    ;SET BUFFER PTR
0175  1F3F  85 73        STA INDEX2
0176  1F41  A9 80        LDA #>SCREEN
0177  1F43  85 74        STA INDEX2+1
0178  1F45  A0 01     FN10   LDY #1      ;START LINE # HI
0179  1F47  B1 73        LDA (INDEX2),Y  ;GET LINE #
0180  1F49  C5 09        CMP LINNUM+1    ;IS IT THE SAME?
0181  1F4B  F0 15        BEQ FN50        ;YES-MAYBE, CHECK LO
0182  1F4D  C9 FF        CMP #$FF        ;NO-DONE?
0183  1F4F  D0 18        BNE FN60        ;NO-INC NEW LINE #
0184  1F51  85 B1        STA FACTO       ;YES-DID NOT FIND IT
0185  1F53  85 B0        STA FACTO-1     ;MAKE # VERY BIG
0186  1F55  A5 B0     FN20   LDA FACTO-1 ;SWAP HI/LO FOR
0187  1F57  85 B2        STA FACTO+1     ;  CRAZY BASIC
0188  1F59  A2 90        LDX #$90        ;SET FOR FLOAT
0189  1F5B  38           SEC
0190  1F5C  20 1B DB     JSR FLOATC      ;FLOAT IT
0191  1F5F  4C AF DC     JMP FOUT        ;MAKE ASCII IN BUFFER
0192  1F62              ;
0193  1F62  88        FN50   DEY         ;CHECK LINE # LO
0194  1F63  B1 73        LDA (INDEX2),Y
0195  1F65  C5 08        CMP LINNUM      ;IS LO SAME?
0196  1F67  F0 EC        BEQ FN20        ;YES-GO FLOAT IT
0197  1F69  20 CA 1F  FN60   JSR ADDSTP  ;MAKE NEXT NEW #
0198  1F6C  20 E1 1F     JSR BUPX2       ;BUMP INDEX 2
0199  1F6F  D0 D4        BNE FN10        ;BRANCH ALWAYS



0201  1F71              ;MOVES TEXT UP ONE BYTE
0202  1F71              ;
0203  1F71  20 B0 1F  MOVUP   JSR SETPTR  ;SETUP
0204  1F74  A0 00     MU10   LDY #0
0205  1F76  B1 71        LDA (INDEX1),Y  ;GET BYTE TO MOVE
0206  1F78  C8           INY             ;POINT TO ONE HIGHER
0207  1F79  91 71        STA (INDEX1),Y  ;MOVE IT
0208  1F7B  20 D6 1F     JSR CMPX        ;ARE INDEX'S EQUAL
0209  1F7E  D0 08        BNE MU40        ;NO-SKIP
```

```
LINE # LOC     CODE       LINE

0210  1F80  E6 7C           INC VARTAB      ;YES-FINISHED BUT
0211  1F82  D0 02           BNE MU20        ;  TEXT IS ONE BYTE
0212  1F84  E6 7D           INC VARTAB+1    ;  LONGER NOW
0213  1F86  88        MU20   DEY             ;LEAVE WITH Y=0
0214  1F87  60              RTS
0215  1F88              ;
0216  1F89  A4 71     MU40   LDY INDEX1      ;DO A BORROW
0217  1F8A  D0 02           BNE MU60        ;NO
0218  1F8C  C6 72           DEC INDEX1+1    ;BORROW
0219  1F8E  C6 71     MU60   DEC INDEX1
0220  1F90  4C 74 1F        JMP MU10


0222  1F93              ;MOVES TEXT DOWN ONE BYTE
0223  1F93              ;
0224  1F93  20 B0 1F  MOVDWN JSR SETPTR      ;SETUP
0225  1F96  A0 01     MD10   LDY #1
0226  1F98  B1 73           LDA (INDEX2),Y  ;GET BYTE TO MOVE
0227  1F9A  88              DEY             ;POINT ONE LOWER
0228  1F9B  91 73           STA (INDEX2),Y  ;MOVE IT
0229  1F9D  20 D6 1F        JSR CMPX        ;ARE INDEX'S EQUAL
0230  1FA0  F0 05           BEQ MD30        ;YES-SKIP
0231  1FA2  20 E4 1F  MD20   JSR BUPX1      ;NO-SLIDE UP ONE
0232  1FA5  D0 EF           BNE MD10        ;BRANCH ALWAYS
0233  1FA7  A4 7C     MD30   LDY VARTAB      ;DO A BORROW
0234  1FA9  D0 02           BNE MD35        ;NO
0235  1FAB  C6 7D           DEC VARTAB+1    ;BORROW
0236  1FAD  C6 7C     MD35   DEC VARTAB
0237  1FAF  60        MD40   RTS


0239  1FB0              ;SETS PTRS FOR TEXT MOVES
0240  1FB0              ;
0241  1FB0  A5 7C     SETPTR LDA VARTAB      ;END OF TEXT
0242  1FB2  85 71           STA INDEX1
0243  1FB4  A5 7D           LDA VARTAB+1
0244  1FB6  85 72           STA INDEX1+1
0245  1FB8  A5 C9           LDA TXTPTR      ;START MOVE HERE
0246  1FBA  85 73           STA INDEX2
0247  1FBC  A5 CA           LDA TXTPTR+1
0248  1FBE  85 74           STA INDEX2+1
0249  1FC0  60              RTS


0251  1FC1              ;SETS UP FACTO AND FACTO-1
0252  1FC1              ;  TO GENERATE NEW LINE #'S
0253  1FC1              ;
0254  1FC1  A5 4E     SETFAC LDA BEGIN
0255  1FC3  85 B0           STA FACTO-1
0256  1FC5  A5 4F           LDA BEGIN+1
0257  1FC7  85 B1           STA FACTO
0258  1FC9  60              RTS


0260  1FCA              ;ADDS STEP TO FACTO-1 AND FACTO
0261  1FCA              ;
0262  1FCA  A5 B0     ADDSTP LDA FACTO-1     ;GET LO
0263  1FCC  18              CLC
0264  1FCD  65 4D           ADC STEP        ;ADD INCREMENT
0265  1FCF  85 B0           STA FACTO-1     ;SAVE LO
0266  1FD1  90 02           BCC ARTS        ;NO CARRY-SKIP
0267  1FD3  E6 B1           INC FACTO       ;ADD CARRY
0268  1FD5  60        ARTS   RTS
```

```
LINE # LOC     CODE       LINE

0270  1FD6              ;COMPARES INDEX1 AND INDEX2
0271  1FD6              ;  ZERO FLAG SET IF EQUAL
0272  1FD6              ;
0273  1FD6  A5 71     CMPX   LDA INDEX1      ;CHECK LO
0274  1FD8  C5 73           CMP INDEX2      ;EQUAL?
0275  1FDA  D0 04           BNE CRTS        ;NO
0276  1FDC  A5 72           LDA INDEX1+1    ;MAYBE-CMP HI
0277  1FDE  C5 74           CMP INDEX2+1
0278  1FE0  60        CRTS   RTS


0280  1FE1              ;INCREMENTS INDEX2 ONE OR TWO
0281  1FE1              ;
0282  1FE1  20 E4 1F  BUPX2  JSR BUPX1      ;BUMP INDEX2 TWO
0283  1FE4  E6 73     BUPX1  INC INDEX2      ;BUMP INDEX2 ONE
0284  1FE6  D0 02           BNE BRTS
0285  1FE8  E6 74           INC INDEX2+1    ;ADD A CARRY
0286  1FEA  60        BRTS   RTS


0288  1FEB              ;GET A CHAR  SET ZERO FLAG
0289  1FEB              ;
0290  1FEB  A0 00     GRAB   LDY #0          ;ZERO INDEX
0291  1FED  E6 C9           INC TXTPTR
0292  1FEF  D0 02           BNE GR10
0293  1FF1  E6 CA           INC TXTPTR+1
0294  1FF3  B1 C9     GR10   LDA (TXTPTR),Y
0295  1FF5  60              RTS


0297  1FF6  89        TOKEN  .BYTE $89       ;GOTO
0298  1FF7  8A              .BYTE $8A       ;RUN
0299  1FF8  8D              .BYTE $8D       ;GOSUB
0300  1FF9  A7        AAAA   .BYTE $A7       ;THEN
0301  1FFA                   .END

ERRORS = 0000


SYMBOL TABLE

SYMBOL    VALUE

 AAAA     1FF9    ADDSTP   1FCA    ARTS     1FD5    BEGIN    004E
 BEGNUM   0064    BRTS     1FEA    BUPX1    1FE4    BUPX2    1FE1
 CHRGET   00C2    CHRGOT   00C8    CMPX     1FD6    COP10    1E9F
 COP20    1EA1    COP80    1EC3    COPNUM   1E8C    CRTS     1FE0
 FACTO    00B1    FINI     0430    FINUM    1F3A    FLOATC   DB1B
 FN10     1F45    FN20     1F55    FN50     1F62    FN60     1F69
 FOUT     DCAF    GR10     1FF3    GRAB     1FEB    INCNUM   000A
 INDEX1   0071    INDEX2   0073    LINGET   C863    LINNUM   0008
 MD10     1F96    MD20     1FA2    MD30     1FA7    MD35     1FAD
 MD40     1FAF    MOVDWN   1F93    MOVUP    1F71    MU10     1F74
 MU20     1F86    MU40     1F8B    MU60     1F8E    NUMX     0050
 RENN     _ECB    RN10     1ED5    RN15     1ED8    RN20     1EE0
 RN30     1EE6    RN35     1EE9    RN40     1EEC    RN45     1EEE
 RN50     1EF8    RN60     1F13    RN65     1F21    RN70     1F27
 RN75     1F2A    RN78     1F2C    RN80     1F34    SCREEN   8000
 SETFAC   1FC1    SETPTR   1FB0    SS       7FFE    START    1E80
 STEP     004D    TOKEN    1FF6    TXTPTR   00C9    TXTTAB   007A
 VARTAB   007C
END OF ASSEMBLY
```

SAM76 UTILITY FUNCTION

(and a script called Monkeys)

BY CLAUDE KAGEN
SAM76 Inc.
Box 257 RR1
Pennington, NJ 08543

This article describes a set of "Utility Functions" written in the SAM76 language, whose purpose is to aid program development in that language by providing facilities for creating, displaying, and modifying texts. A familiarity with the language is for the most part assumed; the references at the end of the article will direct the curious but uninformed reader where to seek enlightenment.

Also included is a SAM76 script (= program) called MONKEYS, which uses a correlation matrix (sort of) to generate random words. The texts of this script can be used as raw material with which to try out the Utility Functions.

My version of SAM76 is the 8080 CP/M (Trademark of Digital Research) version. As far as I know, the only system-dependent feature in the Utility Functions is in the '$decode' Utility, where I check my console data port directly (port 13H) for a control-S in order to freeze the display—a handiness on a 9600 baud CRT terminal. At worst, this feature would simply be inoperative on another configuration: you might, however, want to modify text '$dec' so that it tests your console's data port.

List of Utility Functions

1.	$decode	decode a text (or the list of text names) into ASCII a character at a time
2.	$erase	delete all Utility Function texts in memory
3.	$find	display all texts with specified strings(s)
4.	$fix	modify contents of text(s)
5.	$list	list some or all texts on printer
6.	$make	create (& automatically partition) a text
7.	$upd	save text area on disk
8.	$view	display some or all texts on console

Some Conventions & Restrictions

1) All Utility Function text names contain a '$'. Those which are to be executed as functions begin with a '$'. Additionally, some temporary texts are created ($,$$,$$$, and the infamous 'Text With No Name'), which are normally deleted before control returns to you. If, however, you interrupt a Utility Function, some of these may be left lying around. It's a good idea to delete these, since otherwise the next invocation of a Utility Function might behave strangely.

2) In '$fix', partitions are replaced with " N ", where N represents the ordinal value of the partition. Therefore, when referencing partitions, be sure to use that representation. '$make' also assumes this representation. Both Functions (re—) partition the text before exiting.

3) Multi-partitions a blessing in what they make possible (and in fact easy) to code; they are a curse when you want to modify a text containing one or more of them. It is literally impossible (at least I found it so) to code '$fix' to handle multi-partitions, but I was able to work out a protocol which in part ameliorates the curse (for this and for many ideas sprinkled throughout my code I am indebted to the advice of Ancelme Roichel, creator and curator of SAM76: this one explicit acknowledgement of my indebtedness should be taken to represent many more implicit ones—perhaps on a recursive basis . . .). The S.O.P. is as follows: For any text which is to contain multi-partions, create an 'inactive' version whose name is the intended name bracketed by tilde (~) characters; wherever you want a multi-partition, represent it as you would for a partition, replacing carets with tildes, thus: "~N~". Turn to the explanation of the '$fix' Function for the answer to why you should do this, and what you should do next.

4) You can input ALMOST any string in answer to "FROM:" or "TO:" in '$fix', but there is one restriction: you may not enter a string which is unbalanced with respect to parenthesis pairs.

How To Use The Utility Functions

1. **$decode**

Say you have a text "A", with contents "ABCDEFHIJ" (you think!). Every time you display it, it beeps at you. Clearly, Watson, there's a non-printing BEL character—or characters—but the question is: where and how many? A "%$decode,a/=" will yield the following:

```
A – 41
B – 42
C – 43
D – 44
E – 45
F – 46
  – 7
H – 48
I – 49
J – 4A
```

showing that there is a BEL between the "F" and the "H" in your text (looks like bit-dropping: better run a memory test).

Or, you might somehow have created a text with non-printing characters in the name, and you can't erase it until you know its exact name. Just do "%$decode/=" (no argument), and the list of text names will be decoded in the above format, with text names delimited by periods.

If you are blessed with a CRT terminal, you can use " S " to freeze the display; type any other character to resume printing.

NOTE: this Function, as well as all others which generate a list of texts, first 'hides' all the Utility Function scripts, so that they won't show up in the output. This, however, assumes that you always bring the $UTIL.SAM file into the text area BEFORE fetching your own file.

2. $erase

This Function erases all of the Utility Function texts, except for 'u$' and '$upd', which you will probably want to use in building your own '$upd' text for the file you are working on.

3. $find

This is like '$view' (q.v.), except that the arguments are strings rather than text names. All texts will be displayed which contain ALL of the argument strings.

4. $fix

This is the handiest one of all—and the most complex. If you're at all like me, you'll be using this Function a dozen times for every time you use any other one.

'$fix' has two modes: Quickie and Extended. In Quickie mode, you supply all the information in the arguments to the call for a one-shot fix; Extended mode allows for making multiple fixes in multiple texts, plus a few more goodies.

The Quickie format is:

%$fix,TEXTNAME,FROM,TO,AFTER/=

For example, if you had a text JABBER with contents "'TWAS BRILLIG AND THE SLITHY TOVES", and you did the following:

%$fix,JABBER,TH,M/=
%$fix,JABBER,T,D,ME/=

you would wind up with "'TWAS BRILLIG AND ME SLIMY DOVES" as the new contents of text JABBER. Note that the first '$fix' made two changes, since there were two matches with the 'FROM' string; if you hadn't wanted "THE" changed to "ME", you could have avoided it by one of two techniques: 1) specifying an "AFTER" string (e.g., "SLI"); 2) specifying the "FROM" string more completely (e.g., "THY" instead of just "TH"—but notice that in this case you'll also have to change the "TO" string from "M" to "MY"). In the second example, the "T" in "'TWAS" is not changed to "D" because the "AFTER" string specifies that the search not begin until after the "ME".

Quickie mode automatically displays the text in its changed form before exiting.

Extended mode prompts you for each argument separately. The sequence of prompts is:

TEXT:
FROM:
TO:
AFTER:

Terminate each response with a 'Return'. A null response to the "TEXT:" prompt will exit the Function. A null response to "FROM:" will take you back to the "TEXT:" prompt. Otherwise, the prompts parallel the arguments in the Quickie mode, except that, after each fix, the Function loops back to the "FROM:" prompt; also, there are three subfunctions which may be executed at the "FROM:" prompt:

undo	–	cancels all modifications to the current text
view	–	displays the current text
mpart	–	(cf. CONVENTIONS & RESTRICTIONS, #3) creates a new text which consists of the current text name less the bracketing "~" characters, and multi-partitions this text

The first two subfunctions return you to the "FROM:" prompt; the third returns you to the "TEXT:" prompt, since you would normally create the multi-partitioned 'active' version of a text only after all necessary fixes had been made to the 'inactive' original.

A final nicety which is active only in Extended mode is the ability to shorten a long "FROM:" response by using 3 periods ("...") to indicate ellipsis. Thus, if you wanted to change text "RALLY", which contained "NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR PARTY" to "NOW IS THE SEASON FOR ALL BAD BOYS TO BE THE LIFE OF THE PARTY", you could do it by just entering "TIM . . . HEIR" for the "FROM:" response. Using "..." as the first 3 characters means to start at the beginning of the text, and "..." as the last 3 means: go to the end of the text. CAUTION: be sure you include enough information before and/or after the "..." to determine the "FROM:" string uniquely. However, if you do make a mistake, you can always "undo" it.

'$fix' can work with partitions up to ordinal level 20, and multi-partitions up to level 9.

Note that you can include the normal activator character ("=") as part of a response to a prompt, since the activator has been changed to the 'return' character. Also, you do not have to 'protect' your responses—as you do in Quickie mode—in order to prevent their evaluation; you still may not, however, enter a response with unbalanced parentheses, unless you are fond of unpredictable results, abnormal terminations, and suchlike.

5. $list

This function lists one or more (or all) texts on your printer—assuming you have one. The texts are listed in alphabetical order. "%$list/=" would cause all texts (except the Utility Function texts) to be listed on the printer, whereas "%$list, text1,text2, . . .textN/=" would cause only the texts whose names were supplied as arguments to be listed. In the latter case, this could (if you are the curious type) include texts which are part of the Utility Functions.

Before the listing starts, you will be asked for an "I.D. LINE", which will be printed at the top of the first page of the listing. It is suggested you include such things as the filename, date, the time, etc., in this line, which is terminated by the current activator character.

6. $make

Entering "%$make,NUTEXT/=" is an aid to creating new texts. It first spaces down two lines to give you some elbow room, then accepts input until you type the activator; this input then becomes the contents of text 'NUTEXT'. If you have toed the mark, protocol-wise, by representing all parti-

tions-to-be in the form "^N^", then the partitions will automatically be created for you. Notice also that you do not have to 'protect' the entire body of the text with "!. . ./" in order to prevent its evaluation: this can be a small but not insignificant blessing when you are at the end of a deeply nested text, and you are counting slashes and parens and hoping you have got it right.

7. $upd

What this does right now is save the contents of the text area onto disk as the files "$UTIL.SAM" and "$UTIL.SAK". What you want to do is create a new text with the same name —it has all of 10 characters—which does the same thing, only with the file name YOU want.

While you are in the throes of developing a script, I suggest you go ahead and save the Utility Functions right along with your texts; this way, when you start up next time, you merely have to bring in your own file. Then, when you have your program completely debugged (or thought you did), you can use the '$erase' Function to delete the Utility Function texts before saving your own script in its final form.

8. $view

This displays some or all of the texts on the console. As concerns arguments, the explanation of the '$list' Function applies totally. The difference is that this Function displays the texts on the console; it also pauses after printing each text: if you depress the space bar, it will display the next text; otherwise it will exit prematurely.

Next to each text name, in curly braces, is the number of characters in the text—including the text name itself. Before exiting, '$view' (and '$list') prints the total character count of all the texts listed, and the percentage of the available workspace that this represents.

References

1. The fountainhead for all information about SAM76 is:
   SAM76, Inc.
   Box 257, R.R. 1
   Pennington, NJ 08534
   Available are: manuals, paper tapes, diskettes, and advice.
2. Articles have appeared in the following issues of *Dr. Dobb's:*
   No. 21 (V.3/1) - complete language description (p. 18)
   No. 23 (V.3/3) - addenda to above (p. 46)
   No. 26 (V.3/6) - additional comments (p. 46)
3. In *Creative Computing:*
   May-June '78 (V. 4/3) - complete description (p. 28)
   Sept-Oct '78 (V.4/5) - addendum to above (p. 16)
4. The BASIC inspiration for the MONKEYS program came from reading an article in:
   The American Scientist, Nov-Dec '77 (V. 65), p. 694, W. R. Bennett, Jr.: "How Artificial is Intelligence?"
   This same information can also be found in a book by the same author, entitled:
   Introduction to Computer Applications for Non-Science Students, Prentice-Hall, 1976, p. 104

THE 'MONKEYS' PROGRAM

In the hallowed tradition of "Figger it out yerself: it'll put hair on your chest!", I'm not going to say a whole lot in explanation of this program. This program first accepts textual input from you in order to build up its repertoire of patterns, then it outputs random text using those patterns you input as a basis for selection of characters. In other words, if you give it Shakespearean input, you'll get back vaguely Shakespearean gobbledygook. By the same token, if you feed it text in German, it'll output pseudo-germanic-tongue-twisters.

The input-acceptance phase starts with the text "learn*"; the glossolalia mode is initialed with "create*". The pattern window size is determined by the value of "ws*". Have fun!

```
$UTIL.SAM -- 12 SEPT '78 -- 22:19

   $decode    (67)
   =======
   %cxb,16/%dt,,%i:([1]),,(%ht,u$/&lt,,/%ht/),(&ft,([1]))//#dec
   $:

   $erase     (169)
   ======
   %et,fin$,part$,vu$,cknul$,$view,li$,lv$,$find,$list,filp$,makp
   $,$erase,fxx2$,fixx$,fxx3$,$fix,err$,ell$,el2$,el3$,mpt$,$make,
   dec$,$decode,mpart$$$,view$$$,undo$$$/

   $find      (37)
   =====
   %ht,u$/%dt,,&lw,@,[#1]//%ht/#vu$#::

   $fix       (298)
   ===
   #ii,([1]),,(#fixx$:):^
   #tb,([1]),,(#err$, ([1]),[5]:):^
   %md,([1]),0/%filp$,([1])/^
   %ii,([4]),,,(%notex,#fr,([1]),([4]),(#err$, "AFTER",[5]:):)/
   ^
   %yt,$,...,(#el1$,([1]),[5]:)/^
   %ii,%hm,([1]),([2])%ii,([2]),,(&ft,$/)//,0,(#err$, "FROM",[5]:
   )/^
   %pt,([1]),([2])%ii,([2]),,(&ft,$/)//#md,([1]):%dt,([1]),&ft,([
   1]),([3])//^
   %makp$,([1])/%lv$[5],([1])/

   $list      (65)
   =====
   #cknul$[#1]:%os,
   I.D. LINE - /%os,
   &is/%soc,lst//#ii$%as,@,#:/:

   $make      (45)
   =====
   %dt,([1]),%os,

   /&is//#makp$,([1]):#lv$,([1]):

   $upd       (14)
   ===
   %u$,$UTIL/

   $view      (21)
   =====
   %cknul$[#1]/%vu$%.//

   cknul$     (56)
   ======
   %dt,,%ii,([1]),,(%ht,u$/&lt,@,/%ht/),(([#1]))//%zs,2,0/

   dec$       (96)
   ====
   %ii,&xi,11/,13,(%rr,(#dec$:))/^
   %dt,$,&fc,,(%et,,$/%ri/)//&os,
   &ft,$/ - %cx,,&ft,$///#dec$:
```

```
el2$    (57)
====
%ii,([2]),,,(%notext,&fr,([1]),([2]),(#err$, "FROM",[3]:)///)/


el3$    (82)
====
%md,$,0/`
%dt,$,&ft,$,%ii,([2]),,(&ft,([1])/),(&fr,([1]),([2]),(#err$, "
FROM",[3]:)/)///


el1$    (86)
====
%dt,$$,%ald,([1])//&pt,$,.../`
#el2$,([1]),&fe,$/,[2]:#el3$,([1]),&fe,$/,[2]:`
%md,([1]),#$$:/


err$    (79)
====
%md,&ft,/,0/#makp$,#ft,;:#et,,$,$$,$$$:#os,[2]*([1]) NOT FOUND
 *:%rr,(#fixx$:)/


filp$    (29)
=====
%dt,([1]),&ft,([1])#part$://


fin$    (106)
====
%et,/%os,









TOTAL CHARACTERS@: %zq,2/
WORKSPACE @% USED@: %ad,1,%di,%mu,100,%zq,2//,%qws////%zs,2,0/


fixx$    (131)
=====
#ca,#xc,D::#os,

 TEXT@: :#dt,,#is::#tb,#ft,;,,(#err$, #ft,;,
:):#ii,#ft,;,,(#et,,$,$$,$$$:%zs,2,0/%ri/):%ct,$$$,&ft,//#fxx2
$:


fxx2$    (93)
=====
%os,

 FROM@: :%dt,$,&is//%ii,&ft,$/,,(%rr,(#fixx$:)/)/`
#%tb,&ft,$/$$$,&ft,$/$$$,fxx3$/:


fxx3$    (71)
=====
#$fix,#ft,;,,#os,
   TO@: :#is;,#os,
AFTER@: :#is;,
:#os,
:#fxx2$:
```

```
li$    (48)
===
%ii,([1]),,(#fin$:%soc,con/),(#lv$,([1]):#li$[#2]:)/


lv$    (149)
===
&os,


([1])%tb,([1]),(
(%zs,1,%ad,%ad,%hc,([1])/,%ald,([1])//,%qrd,([1])///%zs,2,%ad,
%zq,2/,%zq,1///%zq,1/)
&ds,%hc,([1])/,=/),( * NOT FOUND *)//%VT,([1])/


makp$    (20)
=====
%pt,([1])#part$:/


mpart$$$    (35)
========
%pt,,~/#mpt$,&ft,/,&ft,,~/:


mpt$    (67)
====
#ct,([1]),([2]):#mt,([1]),~1~,~2~,~3~,~4~,~5~,~6~,~7~,~8~,~9~:
#fixx$:


part$    (96)
=====
,^1^,^2^,^3^,^4^,^5^,^6^,^7^,^8^,^9^,^10^,^11^,^12^,^13^,^14^,
^15^,^16^,^17^,^18^,^19^,^20^


u$    (62)
==
%ef,[1].SAM/%uf,[1].SAM/%bf,[1].SAM/%ef,[1].SAK/%uf,[1].SAK/%b
f,[1].SAK/


undo$$$    (28)
=======
%ct,&ft,/,$$$/#fxx2$:


view$$$    (25)
=======
#lv$,&ft,/:#fxx2$:


vu$    (54)
===
%ii,([1]),,(#fin$:),(#lv$,([1]):#vu$%ii,&ic/, ,,@,/[#2]:)/
```

```
TOTAL CHARACTERS: 2186
WORKSPACE % USED: 8
```

Continued to p. 37

ADD A TRAP VECTOR
FOR UNIMPLEMENTED 6502 OPCODES

BY CARL W. MOSER
3239 Linda Dr.
Winston-Salem, NC 27106

NOTE: Ideas in this article are at the present purely conceptual and have not been implemented. They are believed to be valid and are presented in this preliminary stage for feedback to aid in the ultimate implementation.

One feature the 6502, 6800, 8080, and most other microprocessors lack is hardware to "trap" unimplemented opcodes. This can aid in debugging a program in that the processor would never "hang up" as can now happen especially when executing software not fully debugged. In fact there is at least one unimplemented opcode in the 6800 that requires a power down procedure to reset the processor. Fortunately the 6502 does not have to be powered down for any of its unimplemented opcodes.

What if you could detect those unimplemented opcodes, what have you gained? Well, you've gained in that during debugging you could find out where the execution of the opcode occurred. This is very beneficial but you could also implement those opcodes in your own custom extension of the 6502 instruction set. How would you like a multiply or divide instruction, several one byte calls, auto incrementing and auto decrementing locations, block transfer, a whole group of 16 bit operations, etc. Before I go on, let's present the hardware which could detect an illegal opcode and if detected vector to high order memory for processing.

Trap Vector Circuitry

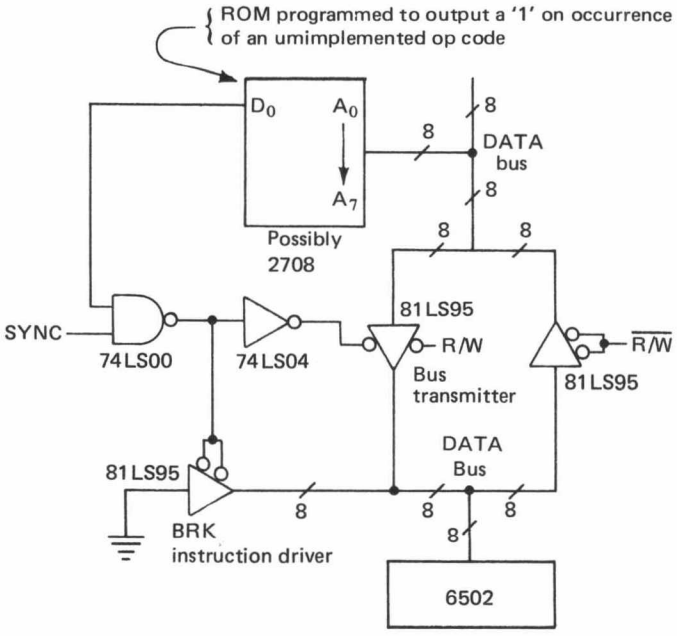
Figure 1 outlines the circuitry which will detect if the 6502 is trying to execute an illegal opcode. One characteristic of the 6502 is that it will output a SYNC pulse (pin 7) every time it reads the opcode portion of an instruction. The ROM of figure 1 has its address inputs connected to the 6502 data bus. This ROM is programmed to output a '1' every time a byte on the data bus corresponds to an illegal opcode.

Whenever the 6502 inputs an opcode, the SYNC will go to a '1', and the output from the ROM will be a '1' if the data on the bus is an unimplemented opcode. If both of these signals are logic 1, the bus transmitter is disabled, a bus driver is enabled, which forces 0's on the data bus. Well this is a break instruction (hex 00). The microprocessor will execute it and vector to FFFE for processing. This processing could obtain the address of the BRK instruction from the stack, input it, and test if it was actually a BRK instruction or an unimplemented opcode. You may be asking, why don't we get forced zeros again? The answer is that the SYNC pulse only occurs during the inputting of the opcode--not when inputting the second or third byte of the instruction or the loaded or stored data that occurs when executing the instruction.

A Possible 6502 Extended Instruction Set

What is the difference between a register and memory? Nothing except that with today's hardware technology, a few locations can economically be made faster than others. Thus execution is presently faster through a register than in ordinary memory. The result is to bottleneck data flow by channeling information through a small set of memory locations (registers). Also, why do we have a register designated as an accumulator? The answer is another hardware limitation dictated by economics.

The extended instruction set I will describe has no accumulators. In fact any memory location is equivalent to an accumulator. This instruction set has 30 autoincrementing index locations, and 32 autodecrementing index locations. The reason for specifying a designated set of index locations is to make the instruction set easier to implement.



If an unimplemented op code and sync occurs, a BRK (hex 00) instruction is forced on the data bus.

Figure 1. Trap vector detection hardware.

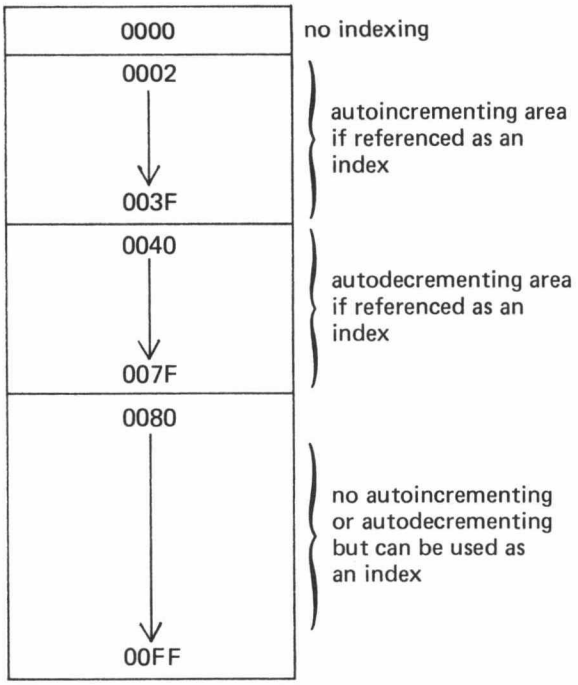


Figure 2. Memory map of index locations.

In this instruction set extension, any time a memory location from 0002 through 003F is specified as an index, autoincrementing will occur on the index contents. Also, locations 0040 through 007F will provide autodecrementing. If any other location is specified as the index, no autoincrementing or autodecrementing will occur. Functionally, this is equivalent to 30 autoincrementing and 32 autodecrementing index registers. This is illustrated in figure 2.

A possible instruction set extension is listed in table A. These instructions provide 16 bit operations and feature instructions to move a block of memory to another area (BLT), multiply (MUL and MUI), divide (DIV and DVI), and compare (CM and CMI) which sets the N, Z, and C flags plus sets the overflow flag (0) if FFFF (-1) occurs.

A typical instruction is: MOV Dlable (Id), Slabel (Is) This instruction should assemble as:

opcode
address lo } source
address hi }
Is — zero page address of 16 bit source index
address lo } destination
address hi }
Id — zero page addrs of 16 bit destination index

NO. BYTES	MNEMONIC		DESCRIPTION
6	BLT	Dlabel (#),Slabel	Block Transfer
7	MOV	Dlabel (Id), Slabel (Is)	Move - memory to memory
6	MVI	Dlabel (Id), immediate data	Move - immediate to memory
7	ADD		Add
7	SUB		Subtract
7	MUL		Multiply
7	DIV	Dlabel (Id), Slabel (Is)	Divide
7	XOR		Exclusive or
7	OR		Logical or
7	AN		Logical and
6	ADI		Add
6	SUI		Subtract
6	MUI		Multiply
6	DVI	Dlabel (Id), immediate data	Divide
6	XRI		Exclusive or
6	ORI		Logical or
6	ANI		Logical and
7	CM	Dlabel (Id), Slabel (Is)	Compare
6	CMI	Dlabel (Id), immediate data	Compare immediate
5	SHR	Dlabel (Id), #	Shift right a number of places
5	SHL	Dlabel (Id), #	Shift left a number of places

NOTE:
Dlabel = Destination address
Slabel = Source address
Id = Destination index
Is = Source index

Table A — Possible extension of the 6502 instruction set.

Continued to p. 41

A Mastermind Player

FOR THE NATIONAL SEMICONDUCTOR SC/MP MICROPROCESSOR

BY CHARLES KAPPS
Temple University
Philadelphia, PA 19122

The game of Mastermind is an adaptation of an old guessing game which has been marketed for the past several years by Invicta Plastics Ltd. The game comes with pegs in six different colors. Player number one chooses a combination of four pegs and player number two tries to guess the combination of colors chosen by player one. Each time player two guesses, player one scores the guess with black and white markers. Each black marker indicates that one peg matched the unknown pattern in both color and position. A white marker indicates that a color was right but was not in the right place. For example, if the hidden pattern was blue, white, blue, red, and the guess was red, blue, blue, blue, the score would be one black and two whites. Player two keeps guessing until he gets four black markers. The object is to finish with as few guesses as possible.

The SC/MP program which follows is designed to run on the $99 SC/MP evaluation kit with an input/output device such as a teletypewriter or video display. The program functions exclusively as player one. It produces a pattern which you have to guess. Instead of colored pegs, the guesses take the form of four digit base 6 numbers (digits 0 through 5). The score is shown as a string of B's followed by W's.

When you start the program, and whenever you finish a game, two exclamation points are printed. The program then goes into a random number generating loop which you stop by typing a character. You therefore get a different sequence every time you play. The program outputs carriage return/line feed, and the game begins. Make a guess of four digits 0 through 5 (repeats *are* allowed). As soon as you type the fourth digit your score will be posted followed by carriage return/line feed, and you guess again, continuing until you get four blacks. Everything will start all over again.

The following is a description of the accompanying program listing. This program listing was prepared using a cross assembler which runs on the PDP-11 computer. (My thanks goes to Tom Copera who developed the assembler.)

The program makes use of two system subroutines which are on the KITBUG ROM. These subroutines are GECO, which reads a character; and PUTC, which prints a character. Lines 16-23 set up calls to PUTC to print two exclamation points. Lines 24-34 repeatedly call the random number routine until a teletype space is received (indicating that somebody typed something). This guarantees a random seed for the random number generator, because the time it takes for a person to type a character is pretty random when measured in tenths of milliseconds.

Lines 35-50 then output a carriage return and line feed, and then generate four more random numbers placing them in ACSII form in the top four levels of the stack. (Note it is assumed that P2 initially contains 0; thus reverse auto-indexing of P2 uses the end of RAM as a stack.)

Lines 51-59 read four guessed digits, placing them and a copy of the machine's random choice on the stack. The reason a copy is made is that, when matches are made, they are scratched out and alternative matches will not be counted more than once. (Note that this is the Mastermind strategy of scoring, which differs from the scoring used in the Milton-Bradley COMP IV. The result is play strategies which vary slightly.)

Lines 70-89 test the guess for exact matches and output B's for each match. Whenever it finds a match, it garbages both the guess and the copy of the machine's secret number. If all four numbers match, the game ends and starts all over again. Lines 90-92 contain data locations. This is put in the

middle of the program so that it is readable from both ends of the program. Note that relative addressing is only good over a range of ±127 locations from an instruction.

Lines 93-128 form a double loop to search for out-of-line matches. W's are printed for each match, but not redundantly. In other words, if the number is 1212 and you guess 0111 your score is BW.

Lines 132-151 are the random number routine. This program simulates the fifteen bit shift register random number generator shown in the *TTL Cookbook* by Donald Lancaster. Note that P1 is used for calling this routine in order to avoid having to reset P3 to call PUTC at the beginning of the program.

There are several simple modifications to the program which can be made to make the game harder or easier. First the radix can be changed so that the numbers guessed are in a larger or smaller range. This number is found in location 227 and can be anything from 2 through A. This will make the guesses range from 0 through n-1. If you do not like 0 you can change the ORI instruction at 22c to F4 and ADI. The range will then go from 1 through n. If the 30 location 22D is changed to 40 as well, letters A, B, C, . . . will be used.

A more interesting set of changes is the number of digits in the guess. Changing this requires changing all the loop counters and address modifiers. There are fifteen of these in the program, and they are given by the following table:

	digits						
address	2	3	4	5	6	7	general
21F	02	03	04	05	06	07	DIGITS
235	FE	FD	FC	FB	FA	F9	-DIGITS
237	02	03	04	05	06	07	DIGITS
244	04	06	08	0A	0C	0E	2*DIGITS
246	02	03	04	05	06	07	DIGITS
255	02	03	04	05	06	07	DIGITS
25B	02	03	04	05	06	07	DIGITS
261	01	02	03	04	05	06	DIGITS-1
263	FE	FD	FC	FB	FA	F9	-DIGITS
267	01	02	03	04	05	06	DIGITS-1
26A	02	03	04	05	06	07	DIGITS
272	FE	FD	FC	FB	FA	F9	-DIGITS
278	06	09	0C	0F	12	15	3*DIGITS
27F	02	03	04	05	06	07	DIGITS
283	02	03	04	05	06	07	DIGITS

As far as adding complexity to the game, changing the radix has little effect. It just means that there are more things to try. However, changing the number of digits has a much more profound impact, and the game becomes noticeably harder when you step from five to six digits.

The next step is to program the machine to play against a person. This is probably beyond the capability of the basic SC/MP kit with its limited memory.

```
1                       TITLE  MASTERMIND PROGRAM
2          0006  RADIX=6
3          0004  DIGITS=4
4          0001  P1=%1
5          0002  P2=%2
6          0003  P3=%3
7          0186  GECO=0186
8          01C5  PUTC=01C5
9          0020  SP=20
10         000D  CR=0D
11         000A  LF=0A
12         0042  BEE=42
13         0057  UU=57
14         0021  EXPLA=21
15         0200  .=200
16 0200    C4    START:  LDI     ^L<PUTC-1>      ;LOAD ADDRESS
                   C4
17 0202    33             XPAL    P3              ;OF
18 0203    C4             LDI     ^U<PUTC-1>      ;PRONT ROUTINE
                   01
19 0205    37             XPAH    P3              ;INTO P3
20 0206    C4             LDI     EXPLA           ;GET EXPLAMATION POINT
                   21
21 0208    3F             XPPC    P3              ;AND PRINT
22 0209    C4             LDI     EXPLA           ;AND DO
                   21
23 020B    3F             XPPC    P3              ;AGAIN
24 020C    C4             LDI     ^L<RAND-1>      ;GET ADDRESS
                   B3
25 020E    31             XPAL    P1              ;OF
26 020F    C4             LDI     ^U<RAND-1>      ;RANDOM NUMBER
                   02
27 0211    35             XPAH    P1              ;ROUTINE (NOTE USE OF P1)
28                        ;
29                        ;RANDOM NUMBERS ARE GENERATED IN A LOOP UNTIL STOPPED MANUALLY
30                        ;
31 0212    3D    RANLP:  XPPC    P1              ;GET A RANDOM NUMBER
32 0213    06             CSA                     ;GET STATUS
33 0214    D4             ANI     20              ;AND SEE IF ANYTHING WAS TYPED
                   20
34 0216    9C             JNZ     RANLP           ;NO, KEEP LOOPING
                   FA
35 0218    C4             LDI     CR              ;OTHERWISE OUTPUT
                   0D
36 021A    3F             XPPC    P3              ;CARRIAGE RETURN
37 021B    C4             LDI     LF              ;AND
                   0A
38 021D    3F             XPPC    P3              ;LINE FEED
39 021E    C4             LDI     DIGITS          ;SET COUNT
                   04
40 0220    C8             ST      K               ;FOR NUMBER OF DIGITS
                   5A
41 0222    3D    GENLP:  XPPC    P1              ;GET RANDOM NUMBER
42 0223    02             CCL                     ;CLEAR CARRY
43 0224    DC             ORI     80              ;FORCE NEGATIVE
                   80
44 0226    F4    MODLP:  ADI     RADIX           ;ADD RADIX
                   06
45 0228    94             JP      GOON            ;UNTIL POSITIVE
                   02
46 022A    90             JMP     MODLP           ;FOR MODULO RADIX
                   FA
47 022C    DC    GOON:   ORI     30              ;MAKE INTO AN ASCII CHARACTER
                   30
48 022E    CE             ST      @-1(P2)         ;AND SAVE ON THE STACK
                   FF
49 0230    B8             DLD     K               ;DECREMENT K
                   4A
50 0232    9C             JNZ     GENLP           ;AND LOOP DIGITS TIMES
                   EE
51 0234    CE             ST      @-DIGITS(P2)    ;BUMP STACK FOR COPY SPACE
                   FC
52 0236    C4    NEXT:   LDI     DIGITS          ;SET COUNT TO DIGITS
                   04
53 0238    C8             ST      K               ;FOR READ LOOP
                   42
54 023A    C4             LDI     ^L<GECO-1>      ;LOAD ADDRESS
                   85
55 023C    33             XPAL    P3              ;OF
56 023D    C4             LDI     ^U<GECO-1>      ;READ ROUTINE
                   01
57 023F    37             XPAH    P3              ;IN P3
58 0240    3F    READLP: XPPC    P3              ;READ DIGIT
59 0241    CE             ST      @-1(P2)         ;AND SAVE ON STACK
                   FF
60 0243    C2             LD      2*DIGITS(P2)    ;GET ORIGINAL ANSWER
                   08
61 0245    CA             ST      DIGITS(P2)      ;AND COPY INTO SPACE ON STACK
                   04
62 0247    B8             DLD     K               ;DECREMENT COUNT
                   33
63 0249    9C             JNZ     READLP          ;AND LOOP
                   F5
64 024B    C4             LDI     ^L<PUTC-1>      ;LOAD ADDRESS
                   C4
65 024D    33             XPAL    P3              ;OF
66 024E    C4             LDI     ^U<PUTC-1>      ;PRINT ROUTINE
                   01
67 0250    37             XPAH    P3              ;INTO P3
68 0251    C4             LDI     SP              ;AND PRINT A
                   20
69 0253    3F             XPPC    P3              ;BLANK SPACE
70 0254    C4             LDI     DIGITS          ;SET COUNT TO DIGITS
                   04
71 0256    C8             ST      K               ;FOR LOOP
                   24
72 0258    C8             ST      B               ;AND SCORE
                   24
73 025A    C2    BEELP:  LD      DIGITS(P2)      ;GET COPY DIGIT
                   04
74 025C    E6             XOR     @1(P2)          ;COMPARE WITH GUESS
                   01
75 025E    9C             JNZ     NOBEE           ;SKIP AHEAD IF NO MATCH
                   0D
76 0260    CA             ST      DIGITS-1(P2)    ;OTHERWISE CLEAR COPY
                   03
77 0262    C6             LD      @-DIGITS(P2)    ;EXTEND STACK FOR SAFETY
                   FC
78 0264    C4             LDI     BEE             ;GET CHARACTER B
                   42
79 0266    CA             ST      DIGITS-1(P2)    ;USE IT TO WIPE GUESS
                   03
80 0268    3F             XPPC    P3              ;AND PRINT B
81 0269    C6             LD      @DIGITS(P2)     ;RESTORE STACK
                   04
82 026B    B8             DLD     B               ;DECREMENT SCORE
                   11
83 026D    B8    NOBEE:  DLD     K               ;DECREMENT LOOP COUNT
                   0D
84 026F    9C             JNZ     BEELP           ;LOOP DIGITS TIMES
                   E9
85 0271    C6             LD      @-DIGITS(P2)    ;RESTORE STACK
                   FC
86 0273    C0             LD      B               ;CHECK SCORE
                   09
```

Continued to p. 39

Performing A Cost-Benefit Analysis of an Improvement Designed to Reduce Home Fuel Costs

BY MICHAEL TROMBETTA
Department of Business
Queensborough Community College
Bayside, NY 11364

Contemplating an improvement in your home heating system—anything from a new burner to solar heating—in order to reduce your fuel costs? Naturally you want to know whether cost is justified by potential savings. To determine if the improvement is justified we calculate the cost and the benefits of the improvement; i.e., we perform a cost-benefit analysis.

The improvement cost is calculated one of two ways, depending on whether the improvement is financed. If we pay for the improvement in a lump sum from our savings, the cost of the improvement is simply the amount of that lump sum payment. If, on the other hand, we finance the improvement with a home improvement loan, the cost of the improvement is the sum of the present values of the payments we will make to amortize the loan. (The present value concept is the standard way of accounting for the time value of money. For example, if someone promises to pay me $100 a year from now, the present value of that $100 is only $95.24 since I could put $95.24 in a bank account which pays 5% interest and it would grow to $100 in one year. The present value depends on how far into the future the payment is made and what interest rate we can earn on our capital).

Each year during its useful life, the improvement will generate a fuel savings. The benefit of the improvement is the sum of the present values of these fuel savings. If the benefit of the improvement exceeds its cost, the improvement is justified.

So far we have described a standard cost-benefit analysis. Since we are performing our cost-benefit analysis on a computer, we can include additional factors which, because they complicate the analysis, are generally neglected. We shall find that these usually-neglected factors may have a profound effect on the cost-benefit analysis.

The first of these additional factors which is included in our analysis is taxes. Taxes enter the analysis in two ways: (1) because we must pay taxes, the actual rate of return on our capital is less than the nominal rate, and (2) the interest component of our loan payments is tax deductible. To include these factors in the analysis, we specify our marginal tax rate. (The marginal tax rate is the tax rate on the last dollar which is taxed). Usually the marginal tax rates on our income and our deductions will be equal, but they need not be. For example, if we don't itemize deductions, the marginal tax rate on our deduction will be zero. On the other hand, if we invest in tax-free bonds, the marginal tax rate on our income will be zero.

The other normally-neglected factor which is included in our analysis is inflation. If we are considering fuel savings realizable in the future, we must know future fuel cost.

Including both these factors, taxes and inflation, can have a profound effect on the cost-benefit analysis. In many cases the after-tax return on capital will be less than the inflation rate (this has been the usual experience over the past several years). Under these circumstances, the value of fuel savings increases faster than the discount rate, so the present value of fuel savings actually increases as we move further into the future. This kind of behavior is never obtained in a standard cost-benefit analysis in which taxes and inflation are neglected.

By including taxes and inflation in the cost-benefit analysis, we get a more accurate result. In many instances a standard cost-benefit analysis would suggest that a particular improvement was not justified, while the more complete analysis presented here would show the improvement was justified.

A program incorporating these ideas was written and run using IBM's CALL-OS BASIC. A program listing is shown in Figure 1; the program is self-documented. A typical execution of the program is shown in Figure 2.

I hope readers will find this program useful when they consider improvements designed to reduce their fuel costs.

Figure 1

```
100 REM THIS PROGRAM DOES A COST/BENIFIT ANALYSIS FOR A CAPITAL
110 REM INVESTMENT WHICH REDUCES FUEL EXPENSES
120 REM WRITTEM BY M. L.TROMBETTA APRIL 1978
130 PRINT"ENTER THE AMOUNT OF THE LOAN (IF NONE, ENTER 0)";
140 INPUT L
150 IF L=0 THEN 250
160 PRINT "ENTER LIFE OF THE LOAN (IN YEARS)";
170 INPUT N1
180 PPINT"ENTER THE INTEREST RATE OF THE LOAN (PERCENT)";
190 INPUT R1
200 R1=R1/100
210 PRINT"ENTER  MARGINAL TAX RATE ON THE INTEREST IN THE LOAN PAYMENT";
220 INPUT T1
230 T1=T1/100
240 GO TO 260
250 N1=0
260 PPINT"ENTER THE COST OF FUEL THIS YEAR";
```

```
270 INPUT F
280 PRINT"ENTER PERCENT REDUCTION IN FUEL COST DUE TO IMPROVEMENT";
290 INPUT P
300 P=P/100
310 PRINT"ENTER THE LIFE OF THE IMPROVEMENT (IN YEARS)";
320 INPUT N2
330 PRINT "ENTER THE EXPECTED ANNUAL RATE OF INFLATION OF FUEL COSTS"
340 PRINT "DURING THE LIFE OF THE IMPROVEMENT (PERCENT)";
350 INPUT I
360 I=I/100
370 PRINT "ENTER YOUR RATE OF RETURN ON CAPITAL (PERCENT)";
380 INPUT R2
390 R2=R2/100
400 PRINT "ENTER YOUR MARGINAL TAX RATE ON INCOME FROM CAPITAL";
410 INPUT T2
420 REM S4 & P4 ARE ACCUMMULATOR-S4 FOR SAVINGS & P4 FOR PAYMENTS
430 T2=T2/100
440 S4,P4=0
450 REM P3 IS AFTER TAX RETURN ON CAPITAL
460 R3=(1-T2)*R2
470 IF N1=0 THEN 540
480 REM A IS ANNUAL PAYMENT ON LOAN
490 A=L*R1/(1-(1+R1)↑(-N1))
500 PRINT
510 PRINT USING 530,A
520 PRINT
530: ANNUAL PAYMENT ON LOAN $#####
540 PRINT USING 550
550:                                                    CUMMULATIVE
560 PRINT USING 570
570:       AFTER TAX     PRESENT            PRESENT VALUE  PRESENT VALUE
580 PRINT USING 590
590:         COST OF     VALUE OF   FUEL       OF FUEL       OF SAVINGS
600 PRINT USING 610
610:  YEAR   PAYMENT     PAYMENT  SAVINGS      SAVINGS     LESS PAYMENTS
620 FOR J= 1 TO N2
630 IF J>N1 THEN 750
640 REM WE ARE STILL PAYING BACK LOAN
650 REM P1 IS INTEREST COMPONENT OF LOAN PAYMENT
660 P1=R1*L
670 REM L IS AMOUNT OF LOAN STILL OUTSTANDING
680 L=L-(A-P1)
690 REM P2 IS AFTER TAX COST OF LOAN PAYMENT
700 P2=A-T1*P1
710 REM P3 IS PRESENT VALUE OF P2
720 P3=P2*(1+R3)↑(-J)
730 P4=P4+P3
740 GOTO770
750 P2,P3=0
760 REM I1 IS FUEL INFLATION FACTOR
770 I1=(1+I)↑J
780 REM S IS VALUE OF FUEL SAVINGS
790 S =F*I1*P
800 REM S3 IS PRESENT VALUE OF S
810 S3=S*(1+R3)↑(-J)
820 S4=S4+S3
830 PRINT USING 840,J,P2,P3,S,S3,S4-P4
840:   ##      $#####       $#####   $#####        $#####          $#####
850 NEXT J
860 PPINT
870 PPINT
880 IF N1=0 THEN 910
890 PRINT USING 900,P4
900: PRESENT VALUE OF ALL PAYMENTS $#####
910 PPINT USING 920,S4
920: PRESENT VALUE OF ALL SAVINGS  $#####
930 IF N1=0 THEN 960
940 PRINT USING 950,S4-P4
950: NET SAVINGS  $#####
960 PPINT "COMPLIMENTS OF M.L.TROMBETTA"
970 END
```

Figure 2

```
RUN

          14:39   05/04/78

ENTER THE AMOUNT OF THE LOAN (IF NONE, ENTER 0)? 1000
ENTER LIFE OF THE LOAN (IN YEARS)? 5
ENTER THE INTEREST RATE OF THE LOAN (PERCENT)? 9
ENTER  MARGINAL TAX RATE ON THE INTEREST IN THE LOAN PAYMENT? 39
ENTER THE COST OF FUEL THIS YEAR? 800
ENTER PERCENT REDUCTION IN FUEL COST DUE TO IMPROVEMENT? 10
ENTER THE LIFE OF THE IMPROVEMENT (IN YEARS)? 15
ENTER THE EXPECTED ANNUAL RATE OF INFLATION OF FUEL COSTS
DURING THE LIFE OF THE IMPROVEMENT (PERCENT)? 6
ENTER YOUR RATE OF RETURN ON CAPITAL (PERCENT)? 8
ENTER YOUR MARGINAL TAX RATE ON INCOME FROM CAPITAL? 38

 ANNUAL PAYMENT ON LOAN $ 257

                                                          CUMMULATIVE
       AFTER TAX    PRESENT                PRESENT VALUE  PRESENT VALUE
        COST OF    VALUE OF     FUEL         OF FUEL       OF SAVINGS
 YEAR   PAYMENT     PAYMENT   SAVINGS        SAVINGS     LESS PAYMENTS
   1     $ 222       $ 212     $  84          $  80          $ -131
   2     $ 223       $ 207     $  89          $  81          $ -257
   3     $ 234       $ 203     $  95          $  82          $ -378
   4     $ 241       $ 199     $ 100          $  83          $ -494
   5     $ 249       $ 195     $ 107          $  84          $ -605
   6     $   0       $   0     $ 113          $  84          $ -709
   7     $   0       $   0     $ 120          $  85          $ -804
   8     $   0       $   0     $ 127          $  86          $ -348
   9     $   0       $   0     $ 135          $  87          $ -260
  10     $   0       $   0     $ 143          $  88          $ -172
  11     $   0       $   0     $ 151          $  89          $  -83
  12     $   0       $   0     $ 160          $  90          $    6
  13     $   0       $   0     $ 170          $  90          $   97
  14     $   0       $   0     $ 180          $  91          $  189
  15     $   0       $   0     $ 191          $  92          $  282

 PRESENT VALUE OF ALL PAYMENTS $ 1017
 PRESENT VALUE OF ALL SAVINGS  $ 1299
 NET SAVINGS  $  282
COMPLIMENTS OF M.L.TROMBETTA

TIME 0.0 SECS.
```

Monkeys *Continued from p. 31*

```
4set*    (19)
=====
%ht,[1]/%ea/%ht/


append*    (41)
=======
&md,*,-#ws*:/%ct,*,*,1*/#extract*:


cloop*    (109)
======
%dt,c*,%ed,#p*;,%rn,%hc,&ft,#p*;///,1//%dt,t*,%ed,
p*,1,2/#c*;/`
%tb,#t*;,(&os,#c*;/%ct,p*,t*/)/#cloop*:


create*    (39)
=======
%dt,p*,[1]/#%tb,#p*;,cloop,nope/*:


ext2*    (84)
=====
%dt,#p*:,%#p*:/#c*:/%dt,p*,%ed,p*,2,99/#c*:/`
%dt,c*,%fc,*//#ext2%ii,#c*:,,a/*:


ext2a*    (24)
======
#getlin*:#append*:


extract*    (50)
========
%dt,p*,%fdc,*,#ws*://%dt,c*,%fc,*//#ext2*:


getlin*    (49)
=======
&os,
@>/%dt,1*,&is/ /%ii,&ft,1*/, ,(%ri/)/


learn*    (34)
======
%ca,%xc,0D//%dt,*, /#ext2a*/


nope*    (28)
=====
%os,
No such pattern./


u$    (62)
==
%ef,[1].SAM/%uf,[1].SAM/%bf,[1].SAM/%ef,[1].SAK/%uf,
[1].SAK/%bf,[1].SAK/


upd*    (16)
====
%u$,MONKEYS/


ws*    (4)
===
3
```

```
TOTAL CHARACTERS: 559
WORKSPACE % USED: 2
```

A Primitive Report Generator Implemented in Zilog System Commands

BY W. M. MC KEEMAN
Information Sciences
University of California
Santa Cruz, CA 95064

Abstract

A report generator, implemented as a series of Zilog Development System commands, is reported. It is, of course, generated by the system it reports. Its salient feature is the automatic inclusion of otherwise independent material through access to the file system during report generation.

Introduction

Generating a technical report involves both the preparation of descriptive text as well as the inclusion of other material such as tables and programs that have an independent existence of their own. It is helpful if the generation of the report can automatically include the latest version of such extra materials rather than having them incorporated into the prepared text at some earlier time. It is for such a purpose that the system described here is designed.

The context of this system is the Zilog Development System which includes a command language for file manipulation as well as editors and similar aids for report generation. Information about these systems can be obtained from the appropriate manuals as noted in the bibliography [1,2].

A file in the Zilog system can be either a text file or a command file depending upon the way it is addressed. A command of the form:

DO XXX

accesses the file named XXX and obeys the commands in it. A DO command can have parameters which follow the name of the file; within the command file the parameters are designated by the symbols #1, #2, etc. Finally a command of the form:

COPY XXX $CON;

accesses the file named XXX and causes it to be displayed on the user's console. With this brief introduction the reader should be able to understand the details now to be reported.

This report was in fact generated by the report generator as will become apparent. There are two output devices addressed by the program: $CON and $DIABLO. The former is the CRT console; the latter is a Diablo printer. It is upon the latter that this report is printed.

A Report Generating Report Generated

Below are presented a series of command files that are activated by the DO command. For ease of identification each such file contains an initial COMMENT which gives the name of the file. There are four files for which the information cannot be easily displayed. They are:

NULL This is a file that contains nothing and therefore has no effect if printed out.

COMMENT This file is absolutely empty also. It is used only as a command file and has no effect when executed. NULL could equally well have been used except that the mnemonic name COMMENT was desired. One may later wish to consider the reason that COMMENT does not contain a comment itself. Upon reflection one will realize that while COMMENT does nothing, it could do it forever if it contained a comment.

DOUBLESPACE This file contains two blank lines.

FORMFEED This file contains the single character for formfeed, i.e., control L or Hex CO.

There are in addition some files named P.m.n where m and n are small integers. They correspond to pages or parts of pages of textual information.

To get a copy of this report on a system which has all the relevant files, type

DO REPORT;

and stand back.

The commands that are triggered off, starting with REPORT itself, are:

```
DO COMMENT  This is file REPORT;
DO COMMENT  First suppress extraneous output;
B;
DO COMMENT  Form feed to get clean sheet;
DO PAGE NULL;
DO PAGE TITLE;
DO PAGE ABSTRACT;
DO SECTION1;
DO SECTION2;
DO PAGE BIB;


DO COMMENT  This is file PRINT;
COPY #1 $DIABLO;


DO COMMENT  This is file PAGE;
DO PRINT #1;
DO PRINT FORMFEED;


DO COMMENT  This is file SECTION1;
DO PAGE P.1.1;


DO COMMENT  This is file SECTION2;
DO PRINT P.2.1;
DO PRINT DOUBLESPACE;
DO PRINT REPORT;
DO PRINT DOUBLESPACE;
DO PRINT PRINT;
DO PRINT DOUBLESPACE;
DO PAGE PAGE;

DO PRINT P.2.2;
DO PRINT SECTION1;
DO PRINT DOUBLESPACE;
DO PRINT SECTION2;
DO PRINT DOUBLESPACE;
DO PAGE P.2.3;
```

There is little more to say. The editor can be used to change $DIABLO to $CON in PRINT to allow the user to interact with his report on the CRT. The reverse change will direct the output to the hardcopy device when desired. Note that it is therefore difficult to get a copy of this report with $CON in file PRINT.

Bibliography

1. (anon.) *RIO Operating System User's Manual*, Zilog Inc., 10460 Bubb Rd., Mountain View, CA 95015 (April 1978).
2. (anon.) *RIO Text Editor User's Manual*, Zilog Inc., 10460 Bubb Rd., Mountain View, CA 95015 (January 1978).

Continued from p. 35

```
87 0275  9C 07      JNZ     WSH             ;NOT ZERO MEANS NOT DONE
88 0277  C6 0C      LD      @3*DIGITS(P2)   ;GAME DONE..CLEAR STACK
89 0279  90 85      JMP     START           ;AND START ALL OVER
90 027B         K:  .BLKB   1
91 027C         L:  .BLKB   1
92 027D         B:  .BLKB   1
93 027E  C4 04  WSH:LDI     DIGITS          ;SET OUTER LOOP COUNTER
94 0280  C8 FA      ST      K               ;TO DIGITS
95 0282  C4 04  WOUTLP: LDI DIGITS          ;SET INNER LOOP COUNTER
96 0284  C8 F7      ST      L               ;TO DIGITS
97 0286  02         CCL                     ;CLEAR CARRY
98 0287  32         XPAL    P2              ;GET STACK POINTER
99 0288  01         XAE                     ;TO EXTENSION
100 289  40         LDE                     ;BUT RESTORE IT
101 28A  32         XPAL    P2              ;TO P2
102 28B  40         LDE                     ;LOW HALF OF SP TO AC
103 28C  F0 EE      ADD     K               ;ADD DIGITS LEFT
104 28E  31         XPAL    P1              ;PUT IN P1
105 28F  36         XPAH    P2              ;DO SAME
106 290  01         XAE                     ;THING
107 291  40         LDE                     ;FOR HIGH PART OF
108 292  36         XPAH    P2              ;P2
109 293  40         LDE                     ;MOVE TO P1
110 294  35         XPAH    P1              ;ASSUMING NO CARRY
111 295  C6 01      LD      @1(P2)          ;GET GUESS
112 297  01         XAE                     ;INTO EXTENSION
113 298  C5 01  WINLP: LD     @1(P1)          ;GET ANSWER
114 29A  60         XRE                     ;COMPARE GUESS
115 29B  9C 07      JNZ     NOUU            ;SKIP IF NO MATCH
116 29D  C9 FF      ST      -1(P1)          ;CLEAR ANSWER
117 29F  C4 57      LDI     UU              ;GET A W
118 2A1  3F         XPPC    P3              ;AND PRINT W
119 2A2  90 04      JMP     YESUU           ;GET OUT OF LOOP
120 2A4  B8 D7  NOUU:DLD    L               ;DECREMENT INNER COUNT
121 2A6  9C F0      JNZ     WINLP           ;LOOP UNTIL ZERO
122 2A8  B8 D2  YESUU: DLD  K               ;DECREMENT OUTER COUNT
123 2AA  9C D6      JNZ     WOUTLP          ;LOOP UNTIL ZERO
124 2AC  C4 0D      LDI     CR              ;THEN PRINT
125 2AE  3F         XPPC    P3              ;CARRIAGE RETURN
126 2AF  C4 0A      LDI     LF              ;AND
127 2B1  3F         XPPC    P3              ;LINE FEED
128 2B2  90 82      JMP     NEXT            ;AND TRY AGAIN
129                 ;
130                 ;RANDOM NUMBER ROUTINE
131                 ;
132 2B4  C0 22  RAND:LD     LHR             ;GET LEFT HALF SEED
133 2B6  1F         RRL                     ;AND SHIFT
134 2B7  C8 1F      ST      LHR             ;AND RESTORE
135 2B9  C0 1C      LD      UHR             ;GET UPPER HALF OF SEED
136 2BB  1F         RRL                     ;AND SHIFT
137 2BC  C8 19      ST      UHR             ;LIKEWISE
138 2BE  1C         SR                      ;SHIFT AGAIN
139 2BF  E0 16      XOR     UHR             ;COMBINE BITS
140 2C1  D4 01      ANI     01              ;SELECT SHIFTIN BIT
141 2C3  9C 06      JNZ     WRAN            ;IF NOT ZERO USE ONE
142 2C5  C0 11      LD      LHR             ;OTHERWISE
143 2C7  D4 3F      ANI     03F             ;CLEAR BIT
144 2C9  90 04      JMP     DONRAN          ;AND SKIP OUT
145 2CB  C0 0B  WRAN:LD     LHR             ;SET BIT TO
146 2CD  DC 40      ORI     40              ;ONE
147 2CF  C8 07  DONRAN: ST  LHR             ;SAVE LOW HALF
148 2D1  C0 04      LD      UHR             ;GET MORE RANDOM PART
149 2D3  3D         XPPC    P1              ;AND RETURN
150 2D4  90 DE      JMP     RAND            ;JUMP FOR RECALL
151 2D6  00     UHR:.BYTE   0
152 2D7  02     LHR:.BYTE   2
153      0200       .END    START
```

```
MASTERMIND PROGRAM
SYMBOL TABLE

B       027D        BEE   =  0042        BEELP    025A
CR    = 000D        DIGITS=  0004        DONRAN   02CF
EXPLA = 0021        GECO  =  0186        GENLP    0222
GOON    022C        K        027B        L        027C
LF    = 000A        LHR      02D7        MODLP    0226
NEXT    0236        NOBEE    026D        NOUU     02A4
PUTC  = 01C5        P1    =% 0001        P2    =% 0002
P3    =% 0003       RADIX =  0006        RAND     02B4
RANLP   0212        READLP   0240        SP    =  0020
START   0200        UHR      02D6        UU    =  0057
WINLP   0298        WOUTLP   02B2        WRAN     02CB

ERRORS DETECTED: 0
FREE CORE: 18458. WORDS

,LP:=PROC
$JOB

$CREATE PROG.PAL

$RT11

*
ERRORS DETECTED: 0
FREE CORE: 18458. WORDS

*
$EOJ
```

Memory Map Program for the Z80/8080

BY ROBERT ALKIRE
Hughes Aircraft Co.
Bldg. 377 M/S J-158
P.O. Box 92919
Los Angeles, Calif. 90009

Author's Note: Often I make mistakes but I am always trying to learn from them. This program I am submitting to you is the result of one of my dumbest (but common) mistakes while going through the pains of computering.

Have you ever put two (or more) memory boards into the same address location? If they are RAM, the computer will work fine, except that there will be a lack of memory. This program scans all addresses and displays a map of memory as seen by the processor. This 'lack of memory.' will be caught quickly and hopefully save some time from head-scratching.

After constructing several memory boards for my Z80 computer, I came to the inevitable conclusion that no two manufacturers use the same method of establishing the start address of a board. Since this only added to the confusion of my already confused computer, I decided to write a program that could assure me that all my boards were where they should be, addresswise.

This program displays a representation of the entire address range of my microprocessor, which in this case is 65536 bytes. This representation is accomplished by breaking the 65536 bytes into 256 blocks with 256 bytes to a block and by testing each block for the existance of RAM, PROM or the absence of memory. Then ASCII characters, corresponding to the type of memory in the blocks, are displayed in a 16 by 16 matrix.

These characters are:

'R' = Random Access Memory
'P' = Read Only Memory
'.' = absence of memory

Also to the left of each line, a Hexadecimal value indicating the start address of the first block in the line is displayed.

The program was assembled for the Z80 microprocessor; however, the program should run without modification on any 8080 based system. The program begins execution at 0000H and is 107 bytes long. In this listing, colons denote Hexadecimal values. Care should be taken if the program is to be relocated since the first location of each block will be modified by the program.

The only external references made by the program are 'OTA' and 'EXIT'. 'OTA' is a subroutine that will output an ASCII character with the parity bit off to a list device. 'EXIT' is a location in memory that the program can go after completing execution. 'OTA' and 'EXIT' are currently set up to operate under the North Star Disc Operating System.

Theory Of Operation:

Beginning testing at location 0000H, the first location is modified and then tested for the presence of RAM. If RAM is detected then an 'R' is printed. If the first location cannot be modified then the next 255 locations are inspected for OFFH (absence of memory). If all locations in the block are OFFH, then the block is non-memory and a period is printed; otherwise, the block is declared PROM and a 'P' is printed. After the block is tested and the appropriate character printed, the routine continues to the next block until all of the 256 blocks are tested.

Conclusion:

Many times software such as this can give a better insight to the inner workings of a computer. I am hoping to see more diagnostic software come about, since it's often painstaking to keep my computer up on all four pads.

```
2                       *
3                       ************************************
4                       *                                  *
5                       *       MEMORY MAP PROGRAM         *
6                       *                                  *
7                       ************************************
8                       *
9                       * PROGRAM IS CURRENTLY SET UP TO RUN
10                      * WITH NORTH STAR OPERATING SYSTEM
11                      * IN ORDER TO PERSONALIZE, CHANGE
12                      * THE OTA ROUTINE AND THE EXIT VECTOR.
13                      *
14 0000 20 28           EXIT    EQU  :2028      NORTH STAR WARM BOOT
15 0000 20 0D           OUTCH   EQU  :200D      NORTH STAR OUTPUT ROUTINE
16                      *
17 0000                         ORG  0
18 0000 00 00           MAP     EQU  $
19 0000 31 BE 00                LD   SP,STACK   SET STACK POINTER
20 0003 21 00 00                LD   HL,0       POINT TO BLOCK ZERO
21 0006 06 01                   LD   B,1        'ITEMS TO A LINE' COUNTER
22 0008 1E 52           MAP1    LD   E,:52      INDICATE 'R' FOR RAM
23 000A 7E                      LD   A,*HL      GET FIRST BYTE OF BLOCK
24 000B 2F                      CPL             INVERT IT
25 000C 77                      LD   *HL,A      TRY TO MODIFY MEMORY
26 000D BE                      CP   *HL        TEST IF MEMORY WAS MODIFIED
27 000E 2F                      CPL             RESTORE A
28 000F 77                      LD   *HL,A      RESTORE MEMORY
29 0010 C2 17 00                JP   NZ,MAP2    SKIP IF NOT RAM
30 0013 BE                      CP   *HL        TEST IF RAM FOR SURE
31 0014 CA 27 00                JP   Z,PRINT    YUP, IT'S RAM
32 0017 1E 50           MAP2    LD   E,:50      INDICATE 'P' FOR PROM
33 0019 3E FF           MAP3    LD   A,:FF      ABSENCE OF MEM IS :FF
34 001B BE                      CP   *HL        TEST FOR NO MEMORY
35 001C C2 27 00                JP   NZ,PRINT   EXIT IF PROM
36 001F 2C                      INC  L          NEXT BYTE
37 0020 AF                      XOR  A          CLEAR A FOR 256 BYTE CTR
38 0021 BD                      CP   L          TEST IF BLOCK CHECK DONE
39 0022 C2 19 00                JP   NZ,MAP3    LOOP 256 TIMES
40 0025 1E 2E                   LD   E,:2E      INDICATE '.' FOR NO MEMORY
41 0027 2E 00           PRINT   LD   L,0        CLEAR THE LSBYTE OF MEM PTR
42 0029 05                      DEC  B          COUNT ITEMS TO A LINE
43 002A C2 35 00                JP   NZ,NLINE   SKIP IF NOT A NEW LINE
44 002D 06 10                   LD   B,16       16 ITEMS TO A LINE
45 002F CD 66 00                CALL  CRLF      OUTPUT CARRAIGE RETURN
46 0032 CD 45 00                CALL  HXOT4     OUTPUT BLOCK ADDRESS
47 0035 3E 20           NLINE   LD   A,:20      A SPACE
48 0037 CD 5E 00                CALL  OTA       OUTPUT SPACE
49 003A 7B                      LD   A,E        GET MEMORY TYPE CHARACTER
50 003B CD 5E 00                CALL  OTA       OUTPUT MEMORY TYPE CHAR
51 003E 24                      INC  H          NEXT BLOCK
52 003F C2 08 00                JP   NZ,MAP1    CONTINUE UNTIL LAST BLOCK
53 0042 C3 28 20                JP   EXIT       RETURN TO OPERATING SYSTEM
54                      * 16 BIT HEXADECIMAL OUTPUT ROUTINE
55 0045 4C              HXOT4   LD   C,H        GET MSBYTE
56 0046 CD 4A 00                CALL  HXO2      OUTPUT MSBYTE
57 0049 4D                      LD   C,L        GET LSBYTE
58 004A 79              HXO2    LD   A,C        GET BYTE TO BE CONVERTED
```

```
59 004B 1F                RRA             ROTATE UPPER NYBBLE
60 004C 1F                RRA
61 004D 1F                RRA
62 004E 1F                RRA
63 004F CD 53 00          CALL  HXO3      OUTPUT NYBBLE
64 0052 79                LD    A,C
65 0053 E6 0F       HXO3  AND   :F        MASK UPPER PART
66 0055 FE 0A             CP    10        TEST IF >10
67 0057 DA 5C 00          JP    CY,HADJ   SKIP IF <10
68 005A C6 07             ADD   7         ADJUST ASCII
69 005C C6 30       HADJ  ADD   :30       ASCII BIAS
                    * DROP THROUGH TO OTA
70                  *
71                  * CHARACTER OUTPUT ROUTINE
72                  * OUTPUTS CHARACTER IN ACC.
73                  * PARITY IS OFF.
74                  * H,L,B,C & ACC. MUST NOT BE MODIFIED
75                  *
76 005E 00 5E       OTA   EQU   $
77 005E C5                PUSH  BC
78 005F 47                LD    B,A       MOVE CHARACTER TO B
79 0060 AF                XOR   A         NORTH STAR DEVICE #0
80 0061 CD 0D 20          CALL  OUTCH     CALL NORTH STAR OUTPUT CHAR
81 0064 C1                POP   BC        RESTORE REGISTERS
82 0065 C9                RET
83 0066 3E 0D       CRLF  LD    A,:D      CARRAIGE RETURN CHAR
84 0068 C3 5E 00          JP    OTA       OUTPUT CHAR & RETURN
85 006B 00                DS    5         IF YOU NEED LINE FEED
86 0070 00                DS    30        STACK AREA
87 008E 00 8E       STACK EQU   $
88 008E                   END

ERRORS=    0


                    ---SYMBOL TABLE---

EXIT  =2028         OUTCH =200D         MAP   =0000
MAP1  =0008         MAP2  =0017         MAP3  =0019
PRINT =0027         NLINE =0035         HXOT4 =0045
HXO2  =004A         HXO3  =0053         HADJ  =005C
OTA   =005E         CRLF  =0066         STACK =008E


                --- INTEL BINARY OBJECT ---

:19000000318E0021000006011E527E2F77BE2F77C21700BECA27001E5012
:190019003EFFBEC227002CAFBDC219001E2E2E0005C235000610CD6600B8
:19003200CD45003E20CD5E007BCD5E0024C20800C328204CCD4A004D7952
:19004B001F1F1F1FCD530079E60FFE0ADA5C00C607C630C547AFCD0D20DC
:19006400C1C93E0DC35E0000000000000000000000000000000000000008D
:11007D0000000000000000000000000000000000000072
:00
```

Continued from p. 33

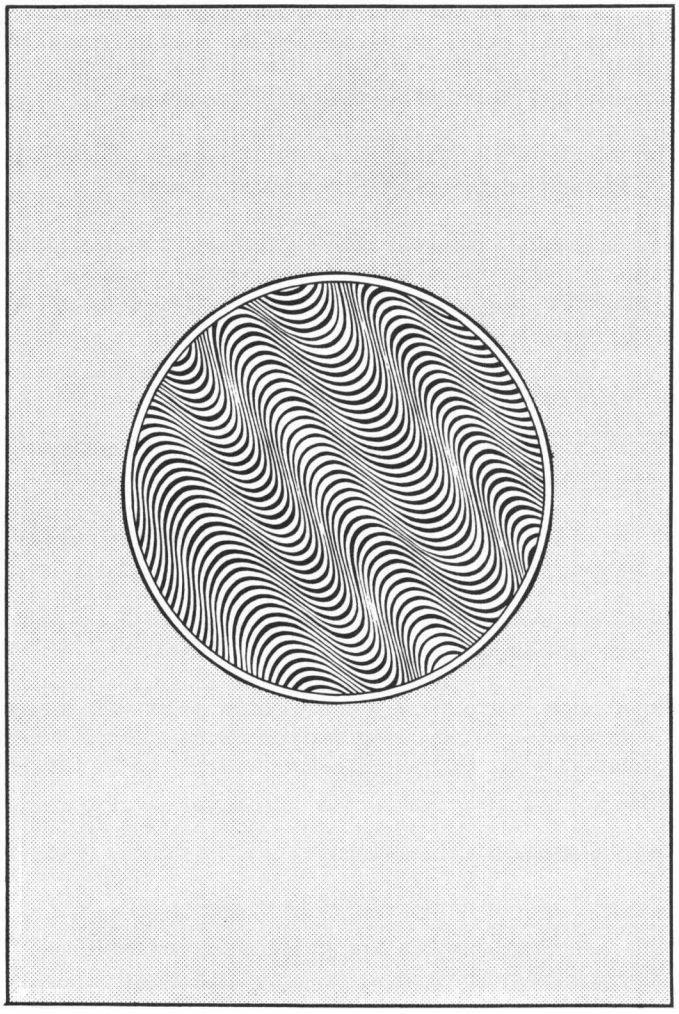
This is a 7 byte instruction which forms a source effective address by adding the 16 bit address Slabel to the contents of the 16 bit index Is. The 16 bit contents at this effective source address are moved to the address similarily formed by adding Dlabel to the index Id. Note, if one of the index locations were in the autoincrementing area, the contents of the index would be incremented *before* the formation of the effective address. Conversely, if the index was in the autodecementing area, the contents of the index would be decremented *after* the formation of the effective address. This is much like the operation of the 6502's stack. If the index was specified at location 0000, this would indicate that no indexing is to take place. Any other location in zero page can be used as an index but no autoincrementing or autodecrementing will take place.

This instruction set extension has very powerful functional characteristics but in no way will it be equivalent to the speed of the 6502's hardware implemented instruction set. It should be apparent that usage of one of the extended instructions when a standard instruction will do is wasteful of processing time. In many applications which require equivalent 16 bit operations to be performed, this instruction set should result in easier programming and may be no slower or use more memory than an application programmed without this extension.

As mentioned at the start, the ideas in this article are conceptual and have not yet been implemented. The purpose of this preliminary description is to obtain reader feedback on any preferences or ideas. When implemented, I plan to have my assembler/text editor support it.

Continued from p. 43

```
100 INPUT X
110 Z = X
200 GOSUB 500
220 PRINT Z,Y
230 GOTO 100
500 REM Y = 10^X
510 X1 = 0
520 IF X>= 0 THEN 530
522 X1 = 1
530 IF X>= 0 THEN 540
532 X = -X
540 X2 = INT (X)
550 X = X - X2 -.5
560 Y = X * X
570 X3 = 69.55697 + Y*8.751758
580 X4 = 60.41641 + Y*(34.29514+Y)
590 Y = (2*X*X3)/(X4-X*X3)
595 Y = Y + 1
596 Y = Y * SQRT(10)
600 IF X2 = 0 THEN 640
610 Y = Y * 10
620 X2 = X2 -1
630 GOTO 600
640 IF X1 = 0 THEN 650
642 Y = 1/Y
650 RETURN
999 END
```



Letters

PREDICTIONS FOR 1979 AND BEYOND

Dear Dr. Dobb's *Received: 78 November 8*

I have always enjoyed reading New Year predictions; this year I decided to make a few myself. Over the next two years:

- the Intel 8086 and its support chips will be the most prevalent 16-bit micro system.

- the Z8000 and MC68000 will suffer and take a long time to gain popularity mostly because of their later arrival and the lack of a comprehensive and immediately available family of support devices. Typically, only applications requiring large amounts of memory will decide against the 8086.

- Pascal will be the most-often-seen high level language on micros (the UCSD version will be the basis of most implementations, but the compiler and editor will generally be extracted, and the remaining software will go unused.

- the lack of stack overflow detection in hardware will be the main disadvantage for many chips running Pascal.

- the hobbyist software market and its major problems (high cost of quality software and documentation; profusion of incompatible hardware, software, and distribution media) will remain largely unchanged.

Over the next five to seven years:

- more and more processors using a true stack architecture will appear.

- more machines will be designed AFTER the major software (particularly the compiler) has been written.

- C and similarly designed translators will be the prevalent systems languages.

- physical micro/mini/midi/maxi distinctions will become more vague until completely lost; processors will only be differentiated by their architectures, speed, and addressing ranges.

- minifloppies will be obsolete.

- voice control of the household (lights, appliances, and so on) will be widely available in the form of reliable, cheap, commercial products.

Over the next fifteen to twenty years:

- paper mail and newspapers will be obsolete as a result of improvements in electronic media, particularly graphics.

- most education will take place in the home and be run by competitive educational corporations, or appear in the form of a public utility. There will be heavy use of teaching machines.

Sincerely,
Mike Gabrielson

Box 2692
Stanford, CA 94305

WE BLEW IT!

Dear Dr. Dobb:

Some minor errata for *DDJ* 3:9.

Page 3. The code for the "neat trick" has a typo! In the 8080 case it should read

```
ADI 090H
DAA
ACI 040H
DAA
```

Also, both Ken Burbett and Bill Byerley are no longer with the Intel software group. Bill is at Digital Research and Ken has his own company, Dharma Systems.

Page 14. The code for the power function is *very* incorrect. The algorithm specified, however, is correct. Specifically, in the basic code n should be replaced by x2, the polynomials need to use x rather than y, and the final result needs to be adjusted by adding one and multiplying by $\sqrt{10}$ before multiplying by 10^n. I've attached a DEC system 20 version which does run correctly. I've made no attempt to optimize; notice the differences between the different dialects of BASIC.

Page 17. Jim Warren attributes to me the aphorism about standing upon each other's shoulders rather than each other's toes. Would that I had said it, but it was said by Richard Hamming in his Turing Lecture. *Listing on p. 41*

Respectfully,
Dennis Allison

169 Spruce
Menlo Park, CA 94025

Dr. Dobbs:

I have received a correction to my 8080 disassembler article published in *DDJ* 27.

Tom Thornton of Sacramento pointed out that line number 1500 in figure 3 seems to have gotten lost. It should be a

CPI ' ' ; compare to blank. The hex dump provided in the article will cover it for those who notice the omission, but I hope you'll print a warning.

Also, ALS-8, which assembled figure 3, has messed up the addresses relating to OBUF and SYMTB. It shows them at DF3A and DF7A in lines 5835 and 5840, but references to them are slightly different.

This seems to be an undetected phase error between pass 1 and pass 2 of assembly caused by the formatting of the line 5825 changing the number of blanks in the ASC constant on the second pass, presumably before the line was assembled! Since these lines are at the end all references are correct and only the lines mentioned are printed incorrectly. This does not cause any bugs, but is very confusing to someone trying to verify that they haven't left out any instructions by checking the length of the program. The ALS-8 manual warns of this, so it's my bug. Sorry about that.

Some users of CUTER have forgotten to make the required changes as specified in the article. At least for one version of CUTER the changes are:

```
> EN DAFA
78 C3 C3 A5 C3/
```

Note that PSCN is really PSCAN in CUTER.

Sincerely,
Richard Greenlaw

251 Colony Ct.
Gahanna, OH 43230

ERRATA

Frits Van Der Wateren's article in the Number 28 issue, LISP for the M6800, contained two extra parentheses in the example given to define the function N!

The correct listing should look like this:

```
(DEC FAC (N) (COND  ((EQ N 0) 1)

                    (T (TIMES N (FAC (MINUS N 1))))

          ) )
```

BEDAZZLED

Dear Dr. Dobb's;

Received: 78 Sept 13

Modification is also sincere flattery and here is a mod for the *Journal.* Phil Mork has a neat idea in his *Vocal Memory Dump* (*DDJ* Vol 3, No.8, p.42). I tried it and got beautiful results with a ten-second sentence that I recorded earlier. (My wife was not so happy – it reminded her too much of *Demon Seed*). Anyway, it occurred to me that combining Phil's way of putting voice in RAM with my dazzler might produce an interesting display. A sort of digital 'color organ' resulted. Here is how it works: Phil's circuit in figure 2 is a one bit analog-to-digital converter which I use to square-up the music from by hi-fi forming a train of pulses as he described. Each eight bits are placed as a byte randomly into memory. The dazzler then displays the resulting pattern. Quiet passages will produce an empty dark red screen. A swell of music or sudden voice will cause a pulsating burst of colors. A constant passage results in a kaleidoscope-like display and if you look for long enough . . .

My dazzler uses ports 1E and 1F for address initialization and mode command respectively. To change to your dazzler ports, change DAZADR and DAZCOM in lines 1100, 1360. To relocate the dazzler RAM change DAZRAM and DAZLOC (lines 1080, 1260). Remember, DAZLOC is the most significant byte of the dazzler RAM start address modified as per the instructions from Cromemco.

Additional comments and mods:

- For a black background, put a 'NOP' in place of the MOV B,A in line 3178. The most common byte has one or two nybbles equal to 15_{10} (i.e., all 'ones'). This tends to make lots of white squares unless one is added to each nybble as I have done.
- You may use switches connected to a spare port to select the dazzler mode in line 1340.
- You may use the 2K RAM mode by changing ADMSK (line 1220) to a hex 7 at the cost of a much slower display.
- If the background is not dark, try adjusting one of the two 10K bias resistors on the LM3900. No input signal should produce the LOW output at the output, pin 10.

Keep up the good work *DDJ*.

```
0000            0010 ******************
0000            0020 * AUDIO DAZZLER *
0000            0030 * VER 0.2       *
0000            0040 * 9/9/78        *
0000            0042 *               *
0000            0044 * ROGER COLEMAN *
0000            0046 * (WITH THANX   *
0000            0047 * TO PHIL MORK) *
0000            0050 ******************
0000            0060 *
0000            0810 *
0000            0930 * GENERAL EQUATES:
0000            0940 *
001E            0945 DAZADR EQU 1EH
001F            0950 DAZCOM EQU 1FH
0008            0955 PORT EQU 8
0000            0958 * * CHANGE NEXT TWO TO RELOCATE DAZZLER RAM * *
2800            0960 DAZRAM EQU 2800H
0094            0965 DAZLOC EQU 94H
0001            0970 ADMSK EQU 1  MASK FOR 512 BYTES
0010            0978 MODE EQU 10H  COLOR IN 512 BYTES
0000            0997 *
0000            0998 * MAIN:
0000            0999 *
0000 315F00     1000 STRT LXI S,STACK
0003            1020 * SELECT DAZZLER RAM AREA AND TELL DAZZ
0003 3E94       1080  MVI A,DAZLOC
0005 D31E       1100  OUT DAZADR
0007            1120 * NOW GET 'AUDIO BYTE'
0007 CD3800     1140 LOOP CALL AUDIN
000A C5         1160  PUSH B  SAVE AUDIO BYTE
000B            1170 * AND RANDOM ADDRESS
000H CD2500     1180  CALL RANDO  GET 16 BIT #
000E            1190 * TEST FOR RAM SIZE
000E 7C         1200  MOV A,H  GET MS BYTE
000F E601       1220  ANI ADMSK  MASKKE FIT DAZ RAM
0011 67         1240  MOV H,A
0012 110028     1260  LXI D,DAZRAM
0015 19         1280  DAD D
0016 F1         1300  POP P  GET AUDIO BYTE
0017 FEF0       1305  CPI 0F0H  TOO WHITE?
0019 DA1D00     1310  JC NXT  NO-SKIP
001C B5         1315  ORA L  ELSE-RANDOMIZE BYTE
001D            1319 * PUT IN RAM
001D 77         1320 NXT MOV M,A
001E 3E10       1340  MVI A,MODE
0020 D31F       1360  OUT DAZCOM  SET MODE
0022            1370 * KEEP ON TRUCKIN'
0022 C30700     1380  JMP LOOP
0025            1400 *
0025            1500 * SUBROUTINES:
0025            1600 *
0025            1700 *  MAKE RANDOM NUMBER BY X=13*X+1  (WELL, ALMOST
0025            1800 *                                   RANDOM)
0025 2A4900     2000 RANDO LHLD RND  GET RNDO #
0028 E5         2020  PUSH H
0029 C1         2040  POP B  TIMES**
002A 29         2060  DAD H  13***
002B 29         2080  DAD H
002C E5         2100  PUSH H
002D 09         2120  DAD B
002E E5         2142  PUSH H
002F C1         2160  POP B
0030 E1         2180  POP H
0031 29         2200  DAD H
0032 09         2220  DAD B
0033 23         2240  INX H  PLUS ONE
0034 224900     2260  SHLD RND
0037 C9         2280  RET
0038            2300 *
0038            2400 *  THIS IS FROM PHIL'S PROGRAM
```

```
0038            2500 *
0038 0E08       3000 AUDIN MVI C,8  GET COUNT
003A DB08       3020 ALOOP IN PORT
003C 87         3040  ADD A  PUT B7 INTO CY
003D 78         3060  MOV A,B  GET OLD BYTE
003E 1F         3080  RAR  SHIFT IN NEW
003F 47         3100  MOV B,A  SAVE IT
0040 0D         3140  DCR C FINISHED BYTE?
0041 C23A00     3160  JNZ ALOOP NO
0044 3E11       3170  MVI A,11H
0046 82         3175  ADD B  THIS MAKES DARK RED BACKGROUND
0047            3176 *        PUT NOP IN 3178 FOR BLACK
0047 47         3178  MOV B,A          3178
0048 C9         3180  RET YES-FINISHED
0049            3200 *
0049            8090 *
0049            9000 * RAM AREA
0049            9010 *
0049            9020 RND DS 2
004B            9030 *
005F            9040 STACK EQU $+20
004B            9999 END DS 0
```

Roger Coleman
890 S.E. Gemini Ave.
Palm Bay, Fla. 32905

DAZZLED AGAIN . . .

Dear *DDJ*:

You have recently published articles dealing with Cromemco's TV Dazzler. Gordon French and Gregory Yob wrote "Absolutely Dazzling Dazzleware" and Phillip Simpson submitted "Dazzler Driver." These articles presented very useful software for the TV Dazzler owner. Each article presented similar routines for drawing on color 'dot' on the TV Dazzler screen given X and Y coordinates (DAZC by French and Yob, and PLTBY by Simpson).

Since this routine is frequently used by most TV Dazzler programs, it should be efficient in order to produce smooth 'animation.' For example, in moving an 8 by 8 graphic across the screen, this routine is called 2048 (2 k) times. Therefore, any savings in processor time is greatly magnified, particularly in 'animation.' The maximum number of cycles for DAZZ is 487, 115 for PLTBY, and 106 for DAZC. My routine takes only 97 cycles at maximum, saves all registers, and is completely PROMable. And this can be reduced 92 cycles if the routine is called with the contents of HL pointing at the start of the Dazzler memory location (i.e., steps 3,4,5,6, and 12 could be deleted in such a modification).

The following is the listing of my routine. In calling this routine, the contents of registers HL point to any Dazzler memory location, register D contains the 'X' value, register E contains the 'Y' value, and the lower four bits of register A are the nyble (or color) to be written onto the Dazzler screen.

As you may have guessed from this letter, I am now working on using the TV Dazzler for 'animation' in the 2k color mode. I will soon be experimenting with using the Dazzler port output for a timing signal in this 'animation.' My project requires a standard video output which the Dazzler does not quite provide (Note: It works fine on a television set, but it cannot be accurately videotaped due to slight frequency offset). If any of your readers are working on this problem, or if they are interested in a Dazzler's Users Group, please ask them to contact me.

```
0     325 PUSH D          SAVE REGISTER
1     345 PUSH H            CONTENTS
2     365 PUSH SW
3     174 MOV A,H
4     346 ANI
5     360  -
6     147 MOV H,A
7     021 LXI DE          SETS HL  TO  Note: FIRST DAZZLER
                          POINT AT           MEMORY LOCATION
                          START OF           MUST START AT 2K
                          DAZZLER            BOUNDARY
                          MEMORY
10    000                 SET DE
11    002                 512    (½k)
12    153 MOV L,E            10
13    170 MOV A,B         SEE 3-6          IF X IS GREATER
14    346 ANI                              THAN OR EQUAL 32
15    040                                  (I.E., POINT ON
16    312 JZ                               RIGHT HALF OF SCREEN)
17     22 -                                THEN ADD 512    (½k)
20        -                                                 10
21    031 DAD D                            TO DAZZLER MEMORY
22    171 MOV A,C                          POINTER (HL).
23    346 ANI
24    040 -                                IF Y IS GREATER
25    312 JZ                               THAN OR EQUAL 32
26     32 -                                                 10
27        -                                (I.E.,POINT ON BOTTOM
30    031 DAD D                            HALF OF SCREEN)
31    031 DAD D                            THEN ADD 1024    (1k)
32    171 MOV A,C         STRIP OFF        TO DAZZLER MEMORY
33    346 ANI             EXCESS BITS      POINTER (HL)
34    037 -               FROM Y
35    123 MOV D,E         (I.E., IF        CLEAR DE
36    267 ORA, A              ≥ 32)        CLEAR CARRY BIT
37    007 RLC
40    007 RLC
41    007 RLC
42    037 RAL           PUT 'Y' COMPONENT
43    137 MOV E,A       OF DAZZLER MEMORY
44    322 JNC           OFFSET IN DE
45     50 -                                                 10
46        -
47    024 INR D
50    170 MOV A,B
51    017 RRC           ADD 'X' COMPONENT
52    346 ANI           (AFTER STRIPPING ANY
53    017               EXCESS BITS (I.E.,IF ≥ 32 ))
54    263 ORA,E         OF DAZZLER MEMORY
55    137 MOV E,A       OFFSET TO DE
56    031 DAD D         ADD MEMORY OFFSET(DE)TO DAZZLER POINTER
57    361 POP SW        TAKE NYBLE OFF                    (HL)
60    073 DCX SP        STACK AND RESET       NOTE: LESS CYCLES
61    073 DCX SP        STACK POINTER         THAN A POP-PUSH
62    346 ANI           STRIP ANY EXCESS
63    017  -            BITS FROM NYBLE
64    127 MOV D,A       AND STORE IN 'D'
65    170 MOV A,B
66    037 RAR           DETERMINE FROM X
67    332 JZ            WHETHER NYBLE IS
70    +100 -            TO BE STORED IN THE
71        -             RIGHT OR LEFT SIDE
72    176 MOV A,M       - LEFT SIDE
73    346 ANI           READ PRESENT DAZZLER
74    360 -             MEMORY AND SAVE PART
75    303 JMP           OF BYTE NOT TO BE
76    +111              WRITTEN INTO
77
100   172 MOV A,D        ' RIGHT SIDE
101   007 RLC            ROTATE NYBLE TO
102   007 RLC            LEFT SIDE OF BYTE
103   007 RLC            (MOST SIGNIFICANT PORTION)
104   007 RLC            AND READ PRESENT DAZZLER
105   127 MOV D,A        MEMORY LOCATION AND SAVE
106   176 MOV A,M        PART OF BYTE NOT TO
107   346 ANI            BE WRITTEN INTO
110   017 -
111   262 ORA D          'ADD' NYBLE TO DAZZLER BYTE
112   167 MOV M,A        AND WRITE ON DAZZLER SCREEN
113   361 POP SW         RESTORE REGISTER
114   341 POP HL          CONTENTS AND
115   321 POP DE          RETURN TO
116   311 RET             CALLING PROGRAM
```

Sincerely,
Ray Daly

2617 42nd St. NW #2
Washington DC 20007

FORTH VERSUS ASSEMBLY

Dear Doc:

Received 78 Oct 21

Here are some facts regarding Forth object size and execution speeds versus Assembly coding.

Forth, Inc., some programmers (including me) and others have made incredible statements about Forth code resulting in less memory than, and execution speeds as fast as, Assembly

written code. To clear the air I'll try to explain those claims.

First, Forth code can run as fast as, but not faster than, Assembly, using a constructional statement called "code", followed by a sort of mneumonic machine code string and a jump back to the Forth inner interpreter. It isn't reasonable to just have one big code statement for the whole program, so this gets us into another Forth constructional statement called a "colon definition". Colon statements are slower but save program memory compared to Assembly; they constitute the "high level" aspect of Forth.

An example of a "code" statement in Forth to handle the character input from a CRT to an Intel SBC 80/20 would be:

```
Code key begin ED INP RRC RRC CS END
EC INP A L MOV O H MVI HPUSH JMP
```

NOTE: Forth code statements allow begin-end and if-else-then constructs within the assembly. Forth also requires source-destination-operand organization of each assembly statement (A L MOV instead of MOV L,A).

This routine in Assembly language would be:

```
        ORG $               ;place in next avail
KEY:                                space
BEGIN:  IN EDH              ;input CRT status
        RRC                 ;rotate receiver ready
        RRC                 ;into carry bit
END:    JNC BEGIN
        IN ECH              ;input CRT data
        MOV L,A             ;push data on
        MVI H,Ø             ;stack in 16-bit
        JMP HPUSH           ;format
```

By entering 'KEY 0D DUMP' on the Forth system, you'll get the object code displayed as:

```
4ØØØ DB ØF ØF D2 ØØ 4Ø DB EC 6F
      26 ØØ C3 41 ØØ
```

This is what the Assembly code would produce if ORG'ed at 4000H and the label HPUSH was at 41H.

Reviewing the Forth code statement: "BEGIN" produced no object but acted as a label for "END" and provided the JNC address for END. "CS" provided the JMP type for END, in this case JNC. "CS NOT END" would have complemented the jump type and produced JC.

The above examples, while not especially exciting on the surface, are quite interesting when you're actually writing these programs on a system installed with Forth or one that isn't. Using a standard disk-based Assembler system, you'd probably have to open an edit file, write the program, close the edit file, call the assembler, and load the object file so you could use the debug program to execute: maybe 10-30 minutes, depending on the problems you have along the way. In Forth, you'd enter the code statement on the command line, carriage return, type "KEY", (CR), and it's executing. 30 seconds maximum! If you liked the way "KEY" executed, you'd save it off on the disk using the Forth Editor (another 20 seconds).

The colon statement in Forth was said to save room in memory over Assembly and provide high level language ability. A code statement that would read the CRT keyboard command messages and then execute the desired action could look like:

```
:KEYBOARD 64 Ø DO KEY 7F AND DUP ØD
  = IF LEAVE THEN LOOP EXECUTE;
```

Keyboard is the label of this routine. Every other word (DO, KEY, AND, =, LEAVE, THEN, LOOP, and EXECUTE) requires two bytes of memory. 8-bit numbers require 3 bytes, 1 for the number and 2 for the routine that differentiates numbers from words and provides these numbers on the stack for use by succeeding operations, e.g., 64 and 0 for 'DO'.

The memory saving can be visualized by thinking of the routine "keyboard" as a routine that looks like:

```
KEYBOARD:  CALL 64          ;a program
           CALL Ø           ;a program
           CALL DO          ;to start a 64 loop
           CALL KEY         ;to input data
           CALL 7F          ;# for AND
           CALL AND         ;to AND it
           CALL DUP         ;to DUP data
           CALL ØD          ;for (CR) test
           CALL IF          ;for (CR) test
           CALL LEAVE       ;if (CR) leave loop
           CALL THEN        ;to complete IF
           CALL LOOP        ;to loop 64 times
           CALL EXECUTE     ;to DO command
           CALL ";"         ;to DO next one
```

Looking at it this way, each CALL takes a byte. Fourteen bytes could be saved if the CALL OPCODE could be eliminated. The result would be the two byte addresses of everything to CALL. The innermost Forth interpreter uses these addresses in sequence and is about 12 bytes of memory code with the label "NEXT".

Thus, for just this example, 14 bytes were waved at the cost of 12 bytes for "NEXT". Every colon and code statement uses "NEXT", so memory savings build since "NEXT" is frequently executed. The justification in using sub-routine calls in Assembly code versus in-line code is based on how many times it is called. Forth, Inc., has stated "NEXT" would be an excellent micro-code to be included in a CPU OPCODE set, and I'd have to agree. Before a "NEXT" OPCODE would be implemented in MOS processor-like 8085, 6800, or the like, Forth is going to have to become quite dear to the industry. So I don't see it happening except in some 2900 bit-slice implementations.

"NEXT" is executed between each word in a colon statement, and between each word of a word that is itself the name of a colon statement. Therefore, "NEXT" slows things down during execution, but is redeeming since it saves space and allows the high-level nature of Forth.

To keep things moving quickly in the execution of Forth programs, colon statements should contain few words defining the action of the defined colon statements, and each word should be closely connected to a code statement, since code statements run at full machine speed. Each word in a colon statement should also be powerful; if the word is the label of a code statement, this could mean large code statements.

Large code statements can quickly get out of hand with more than two lines (line in the example of "KEY"), because of the lesser ability to comment each OPCODE as in Assembly. So Forth, Inc., states that code statements should be kept short and sweet. It's really up to the user to trade off readability for speed. The naming of colon and code statement labels can improve readability if you put thought into the naming. As was said earlier, the Forth program statement can be executed by entering it on the command line, then typing the name for execution. Colon statements are included in this ability, and extremely fast coding and debugging is the result.

I really object to paying $2,500 for any software, but Forth is worth it. Besides the high price, there seem to be a few other impediments to Forth gaining popularity: 1) it does take some getting used to; 2) there are not many Forth systems and programmers around; 3) people are quick to condemn it.

There is a group here in Northern California that's doing good things with Forth and they can be reached at Forth Interest Group, 787 Old County Road, San Carlos, CA 94070.

Sincerely,
Richard B. Main

Neptune Ues
7070 Commerce Circle
Pleasanton, CA 94566

MERGING AND PRETTY PRINTING

Dear Sirs,

In your issue #27, you published a Basic renumbering program. It is 'basically' the same program I wrote over a year ago. However, Mr. Trenholme left off the two best parts, merging and pretty printing. Enclosed is a RUN of this program written in Basic. Except for the disk calls the program can be run on all the Microsoft Basics I've tried (OSI 65-D, 65-U, Altair, etc.) by changing the program starting pointer.

```
10 ? "This is a test
11 GOTO 13
12 RETURN
13 ON X GOTO 15,16,17
15 GOSUB 12
16 ? "THE END"
17 GOTO 100
LOAD
01T
A*P50
A*RB
OK
NEW
OK
100 REM  Test two
101 !
102 !   ? "Pretty print.
103!
104 ? "Test for errors"
105 GOTO 200
106 GOSUB
107 IF Y=2 THEN 300
108 END
LOAD
01T
A*P51
A*RB
OK
LOAD
01T
A*L49
A*RB
OK
RUN
MERGE OR NUMBER? M
RELOCATED TO? 24576
MERGE WHAT TRACK? 50
NEXT MERGE AT 24674
MERGE OR NUMBER? N
FILE STARTS AT 24576
START NUMBERING WITH? 10
ORIG. PROGRAM STARTED AT LINE 10
NEXT AVAILABLE LINE NUMBER IS 80
LINE  100 NOT FOUND ( 70 )
 101 Bytes used.
Merge at 24674
STORE (Y OR N)? N
MERGE OR NUMBER? M
MERGE WHAT TRACK? 51
NEXT MERGE AT 24794
MERGE OR NUMBER? N
FILE STARTS AT 24576
START NUMBERING WITH? 10
ORIG. PROGRAM STARTED AT LINE 10
NEXT AVAILABLE LINE NUMBER IS 180
LINE  200 NOT FOUND ( 140 )
ERROR IN LINE 150
LINE  300 NOT FOUND ( 160 )
 221 Bytes used.
Merge at 24794
STORE (Y OR N)? Y
STORE ON TRACK? 50
MERGE OR NUMBER?
OK
LOAD
01T
A*L50
A*RB
OK
LIST
10 PRINT "This is a test
20 GOTO40
30 RETURN
40 ON X GOTO50 ,60,70
50 GOSUB30
60 PRINT "THE END"
70 GOTO90
80
90 REM Test two
100
110     PRINT "Pretty print.
120
130 PRINT "Test for errors"
140 GOTO 200
150 GOSUB
160 IF Y=2 THEN 300
170 END
OK
```

Yours truly,
L. Barker

Chicago, Ill. 60660

MAKING AN ALS-8 WORK FOR A Z-80

Doc:

The Processor Technology memory-resident operating system ALS-8 is a compact, extensible editor, assembler, file manager and, optionally, a screen editor and 8080 interpretive simulator/debugger. It was designed for 8080 processor systems but has been used for Z-80 based systems. Regretably those few 8080/Z-80 differences cause one annoying bug, which in turn causes one of the most significant ALS-8 capabilities to be unusable on Z-80 systems.

The END pseudo-op is used by the ALS-8 assembler to detect the end of the PASS 1 and PASS 2 scans through the assembler source code input. The normal ALS-8 assembly is from memory-resident files and the assembler also recognizes an end-of-file to terminate the scans. The decoding of the END pseudo-op utilizes a JPE (jump on parity even) instruction in two places. Since the Z-80 and 8080 utilize the parity flag differently, the consequence is that *a Z-80 running ALS-8 cannot recognize the END pseudo-op!*

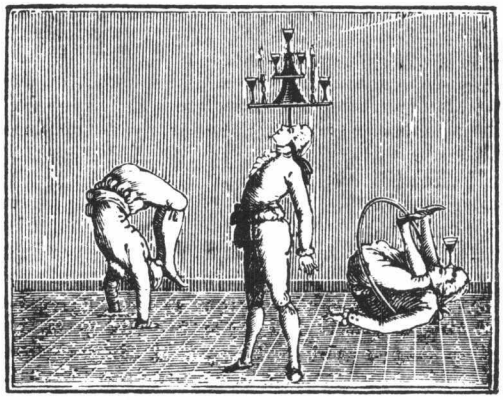
For normal memory-resident files the "fix" is simply not to use the END statement and let the end-of-file marker serve the same function.

But ALS-8 also allows assembly from files on external storage, such as cassettes, by using the ASSI command. But this capability *requires* the END statement because ALS-8 has no way of determining end-of-file on external storage devices. Therefore, a *Z-80 running ALS-8 cannot assemble from external files using the ASSI command.*

Fortunately there is a simple object code "fix" that solves both problems. Essentially the pseudo-op decode logic in two places must be changed to not utilize the offending JPE instruction yet still decode all pseudo-op correctly. The two changes defined below achieve this capability without jumps to "patch" locations; the new code occupies the same locations as the old.

LOCATION	OLD		NEW	
EBDA	FE	CPI 5	FE	CPI 4
EBDB	05		04	
EBDC	D0	RNC	CA	JZ OEE39H
EBDD	EA	JPE OEE39H	39	
EBDE	39		EE	
EBDF	EE		D0	RNC
EC86	CA	JZ OECD3H	28	*JR Z, OECD3H
EC87	D3		4B	
EC88	EC		30	*JR NC, OECCDH
EC89	D2	JNC OECCDH	43	
EC8A	CD		FE	CPI 4
EC8B	EC		04	
EC8C	EA	JPE	CA	JZ

Note that in order for the fixes to be implemented in-place, using no "patch" memory locations, the Z-80 two-byte relative jump instruction was used instead of the 8080 three-byte equivalent in two instances. The two instances are marked with an asterisk. If 8080 and Z-80 compatibility is an important consideration these could be replaced by a normal 8080 jump to a "patch" location and using conventional 8080 conditional jumps at the patch to perform the same function. Since the patch is only required if one is using the Z-80, the use of Z-80 unique instructions seemed justified to me.



Sincerely,
Jon E. Avery

1070 Wayfield Dr.
Norristown, PA 19403

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 32 February 1979 Volume 4, Issue 2

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

A Voyage to Antares *Judith Wasserman* **4**
Robert Burt Ramon Zamora

GENIE (An Immodest Proposal) *David Chapman* **20**
An Electronic Information Network

Let Your Computer Speak ASCII *Gary Gaugler* **18**
It's time for your computer to talk back

PASCAL Bibliography *Mike Gabrielson* **29**

An Unusual Pseudorandom Number Generator Program *H.T. Gordon* **14**

Features for TDL Editor *R.J. Long* **31**
Making the assembler and TOP read from memory

A Simple 2708 PROM Programmer for the 6800 *E. Insam* **34**

A VOYAGE to ANTARES

BY ROBERT BURT

JUDITH WASSERMAN

RAMON ZAMORA

Many years ago, the British savant C. P. Snow, in his book The Two Cultures, *warned of the danger of separating art from science. Snow, a scientist turned writer, decried this tendency toward polarization. Only through the constant mingling of ideas and perspective, he felt, could a brilliant creativity survive in the world.*

It's probably true that there are those in both science and the arts who deny viability of expression to the opposite culture. We've all known artists who dismiss technology as boring and life-annihilating; or the technology buff who's never read a work of fiction or studied a painting. They miss a lot: at the very least a greater appreciation for the world around them and a heightened perspective. I am reminded of my undergraduate days at Stanford when the physicist Wolfgang Panovsky addressed a class of mine. He said if he didn't play his violin at least an hour a day, his creativity would dry up.

What's particularly remarkable about this article is that all three contributors combine technological expertise and artistic knowhow.

The artist, Judith Wasserman, has a B.A. in Physics from Cornell University. For the last several years, she has devoted her time to artwork, particualrly etching. She is President of the Board of Directors of Palo Alto's successful art cooperative, Gallery House. Judy does layout work for PCC; in fact, she has done most of the layout for this article.

The pilot, Robert Burt, has an M.S. in Physics from NYU and an M.A. in Music from Columbia. Bob is a Research Specialist at Lockheed's Satellite Systems Division. He has a grand piano in his living room and is addicted to classical music. He is also an imbiber of fine California wines.

The critic, Ramon Zamora, has a B.A. from the Florida Institute of Technology. He has a total of fifteen years experience with computers. Currently, Ramon is one of three editors of Recreational Computing *magazine. Ramon is also an artist, with particular emphasis on silkcreening and sculpture.*

One note of caution before you proceed with the article. Bob Burt developed his ByByBaybee Program on an HP 9830, but he gives enough information to get the small computer user underway. Unlike most DDJ *articles, then, this one does not give you all the answers: it is meant to excite your creative palate, to get you up and running. Have fun with this one, and good luck! –SMR*

The Critic

At times, computers are said to generate art (graphic forms, textile design patterns). At times, artists (engineering artists, design artists, software artists—yes, there is art involved) generate computers. Once in a while a quiet synthesis occurs when a person working in an area of the fine arts and information from a computer program are joined together.

A show called "Visions of Flight," installed from the end of October to mid-November in the Gallery House of Palo Alto, California, contained such a point of synthesis. Judith Wasserman (*RC*'s own Judith Wasserman) in her section of the show, subtitled "Voyage to Antares," displayed a series of relief etchings derived from the output of a computer program. The program, written by Robert Burt, generated the star configurations that would be observed if one were to travel along a line from our sun to Antares.

For the show, points along this space path at 10, 50, 200 and 500 light years were selected. Haunting views, as seen from a spacecraft's window, fill each print. The cold blackness of space; the bright points of light that represent the stars; the faint yellows, oranges and reds of distant suns. And then, embossed in white across the surface of the prints, renderings of mythical creatures formed of star points and dark voids. *The Bearded Bull of Canopus*; *The Great Octopus of Antares* hang in space 500 light years from here. *The Horned Cats of Hadar*; *The Serpent of Spica* prowl the stars 50 light years away. In all there are fifteen creature/myths portrayed. In the words of the artist they represent "an expansion of earthbound mythology into a future time and space . . ."

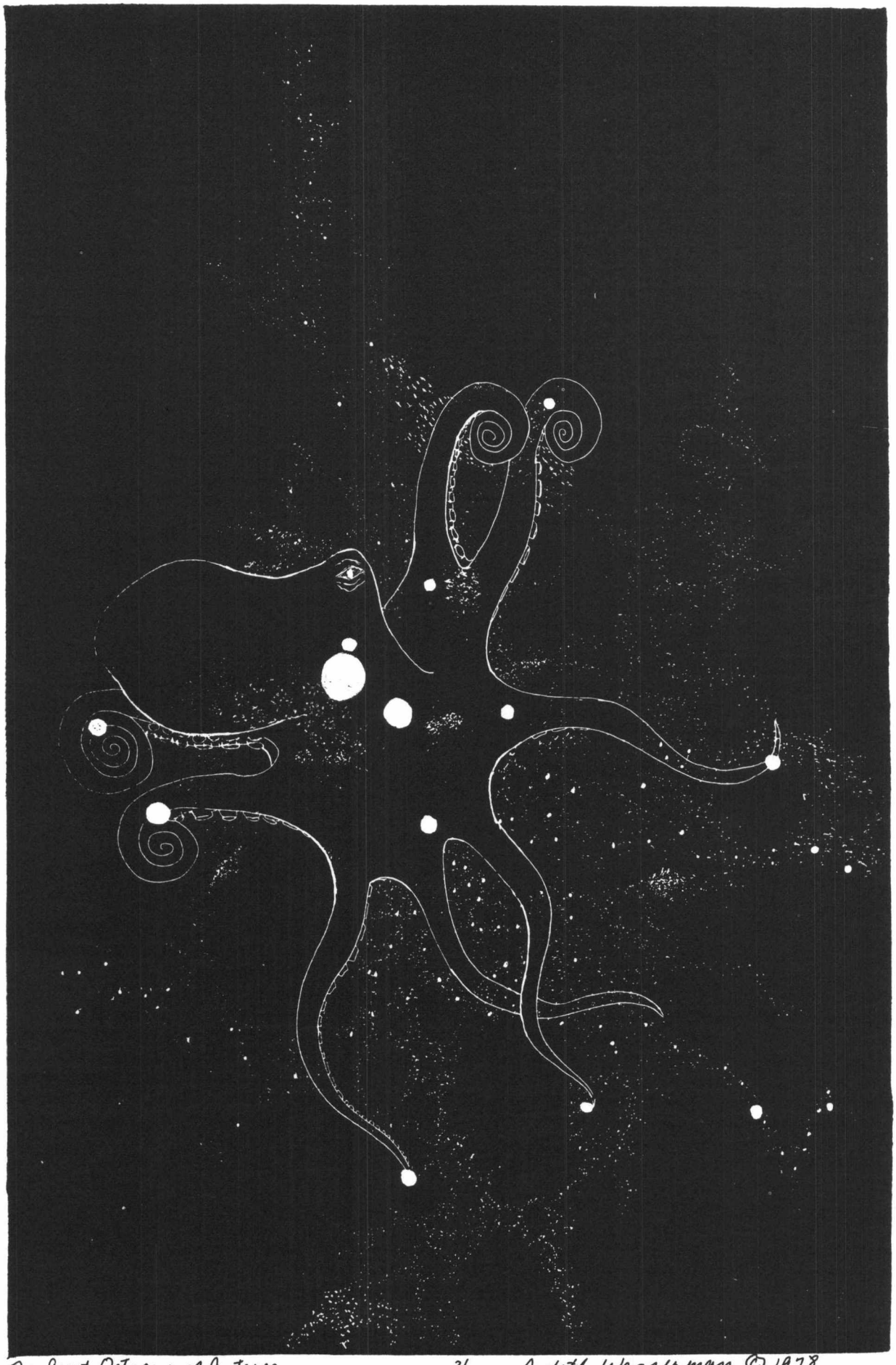
It is joyful to see and experience this creative marriage of technology and spirit; of data and insight. The question arises as to who else and where else is this being done.

(Reprinted from *Recreational Computing*, Vol. 7, No. 4).

The Artist

I first became interested in a program like ByByBaybee when I thought I'd like to go to the center of the Galaxy on my next vacation. Then I met Bob Burt at a party on New Year's Day and discovered he was a celestial mechanic. I had found my pilot. He knew programming, he knew computers, he knew astronomy, he liked art and he was very interested.

My idea was to design a program that would print out a chart of stars as seen from any point in the Galaxy, then to pick a destination, make several stops and draw the constellations we could see along the way.



The Great Octopus of Antares 3/20 Judith Wasserman © 1978

We picked Antares because it was toward the center of the Galaxy and was particularly beautiful from Earth, being red and sparkly, and interesting from close up, being a red giant and very luminous.

Bob spent the spring and summer building the ship and we decided how many stops we would make and at what distances. Since each stop meant a complete sky plot, it entailed a fair amount of computer time, so we limited the number of stops to four. By this time I had a date lined up on which to exhibit these constellations (October 21), and I thought I could find at least three images at each distance, which, along with a few other related pieces, would be enough to both show and exhaust my concept.

On August 20 the print-out was delivered to me: an 11 x 15" piece of graph paper with 286 x 4 diamond shapes in four colors and assorted sizes. (Actually, some of the stars were too dim to show up at 200 and 500 light-years). The chart in itself was beautiful in four colors, one color for each distance, but in order to find constellations I first had to trace off the stars at each distance separately. Then I could look at them up, down and sideways; sometimes I found it necessary to project something not quite there.

One thing missing from this trip to Antares was the sighting of new stars, since the computer couldn't print out stars we didn't put in. Consequently, by the time we got to 500 light-years, the sky was almost empty. I compensated for this by adding fields of imaginary stars. I made a visual distinction between "real" or computer-generated stars and imaginary ones: I made the real ones larger and varied their size according to their computer-determined relative luminosities, whereas the imaginary stars were just pinpricks and swarmed in arbitrary or artist-generated patterns over the image.

In the end I had twelve constellations, three at each distance (10, 50, 200 and 500 light-years), named after the brightest star in each group. They were hand printed as relief etchings with white lines and stars on dark fields and had names like *Serpent of Spica* and *Sphinx of Arcturus*.

The Pilot

Overview. The computer program ByByBaybee calculates and plots the altered patterns and visual magnitudes of the stars as seen by a space traveler at a specified distance from the sun on the path to a specified star. The program, written in BASIC for an HP 9830 and associated plotter, is in three parts. The first part swallows, processes and stores star data in six memory files. The second part retrieves the data and plots a conventional star map for checkout purposes. The third part retrieves the data (one file at a time to prevent core overload) and calculates and plots the new orientations and visual magnitudes (star brightness) at the given spacecraft location.

Change in Star Patterns. The mathematics of the change in star patterns is the mathematics of three-dimensional perspective. The stars diverge from the point where the extended line of spacecraft motion pierces the heavens (pole of approach) and converge toward the opposite pole, the vanishing point of the artist. If the spacecraft moves along a straight line, the apparent position of the stars in the celestial sphere moves along half great circles which arise at the pole of approach and terminate at the vanishing pole. The rate of travel along these half circles is slower for the further stars and varies for each star, being fastest when the spacecraft passes the point of closest approach to the star.

Two-Dimensional Analysis. The two-dimensional analysis of the altered direction of a star is illustrated in Figure 1. Points S, L, Y and X are the solar system (at these interstellar distances, the solar system is a mere point), spacecraft location, destination star and observed star respectively. Angle x is the direction of the observed star at S with respect to the direction of the destination star (taken for convenience as reference direction), and d is the distance from S to X. The problem is to find angle y, the direction of the observed star at various distances, s, from the solar system on the path to Y, and to find l, the distance of X from L—needed, together with the given absolute magnitude, to compute the visual magnitude of the star as seen from L. The solution is simple. We are given three consecutive parts of triangle XSL: distance d, angle x and distance s. Angle y′, the supplement of desired angle y is the fourth consecutive part of the triangle; it is obtained by means of the four-parts law: $y' = \tan^{-1}[d \sin x/(s - d \cos x)]$. Distance l is obtained from the given data by the law of cosines: $l = (d^2 + s^2 - 2ds \cos x)^{1/2}$.

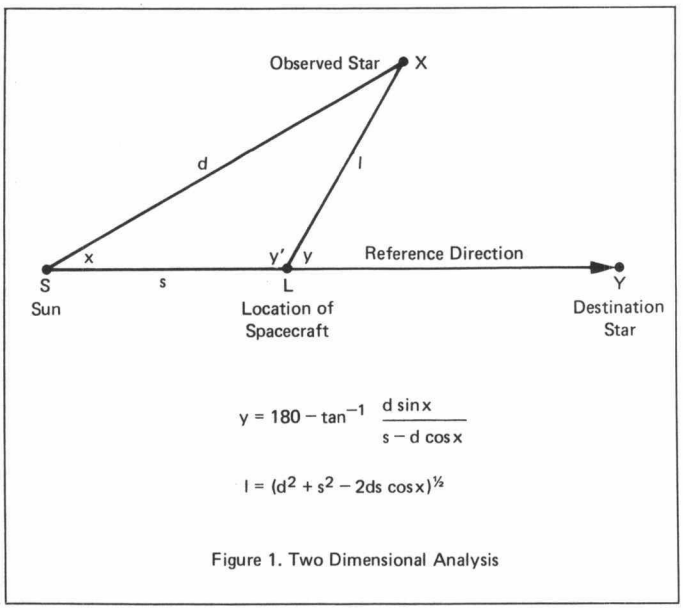
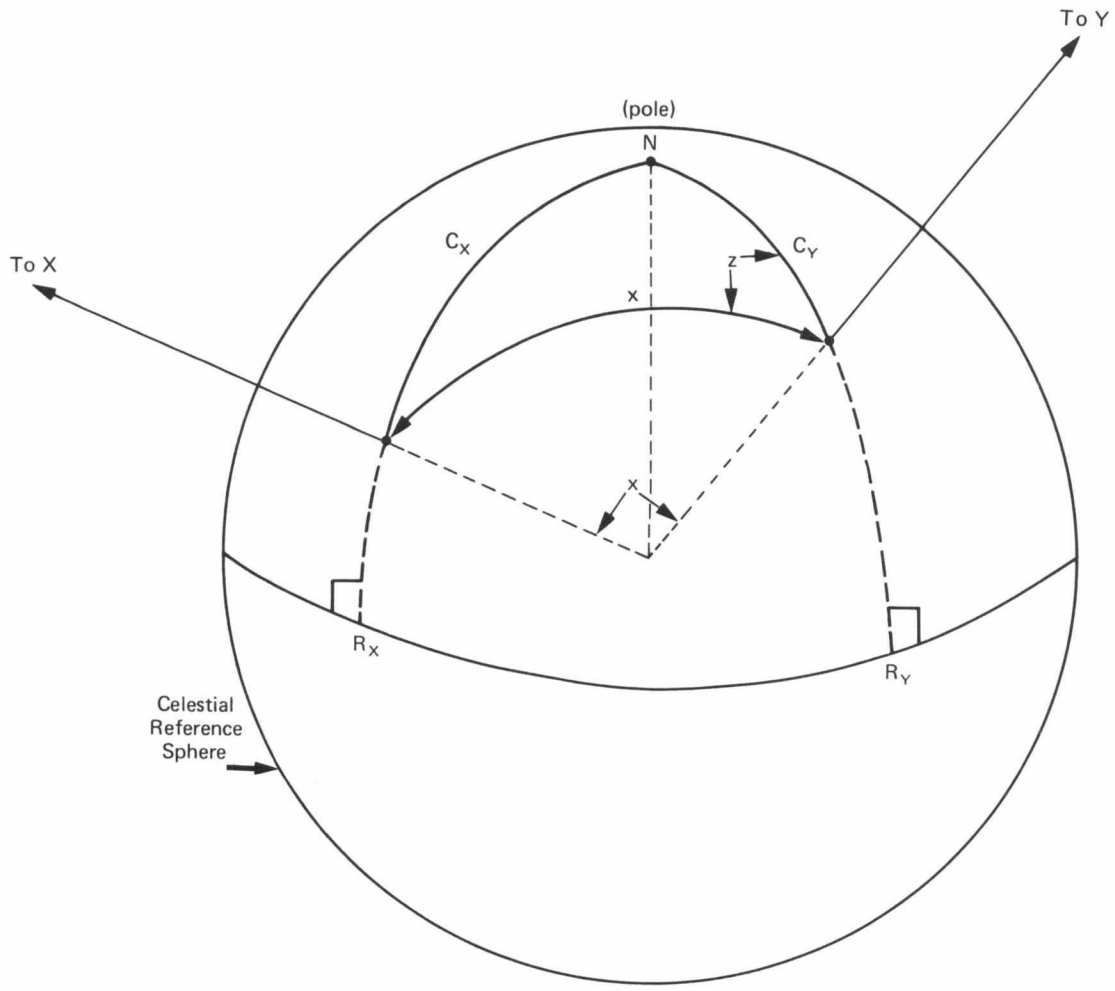


Figure 1. Two Dimensional Analysis

Three-Dimensional Analysis. The two-dimensional analysis is inherent in the three-dimensional analysis: each observed star defines a plane passing through the star and the line of spacecraft motion. Angle x is not given directly but must be determined from the given right ascensions and codeclinations (codeclination, the complement of declination, is the angle of the star from the north celestial pole) of stars X and Y by the law of cosines of spherical trigonometry (Figure 2). Angle z (needed later) is determined by the four-parts law of spherical trigonometry.

The three-dimensional problem is concluded by finding the right ascension R'_X and codeclination C'_X of star X in the celestial reference sphere now centered at L (see Figure 3). Angle z, formed by the plane passing through the NS axis of the reference spheres at S and L and the plane of star X, is the same angle in both spheres. Again we have three consecutive parts: angles y, z and C_Y (the supplement of codeclination of star Y). Again we use the law of cosines and the four-parts formula of spherical trigonometry, this time to find the code-



$$\cos X = \cos C_X \cos C_Y + \sin C_Y \cos (R_X - R_Y)$$

where C_X, C_Y, R_1 and R_Y are the codeclinations and right ascensions of X and Y respectively.

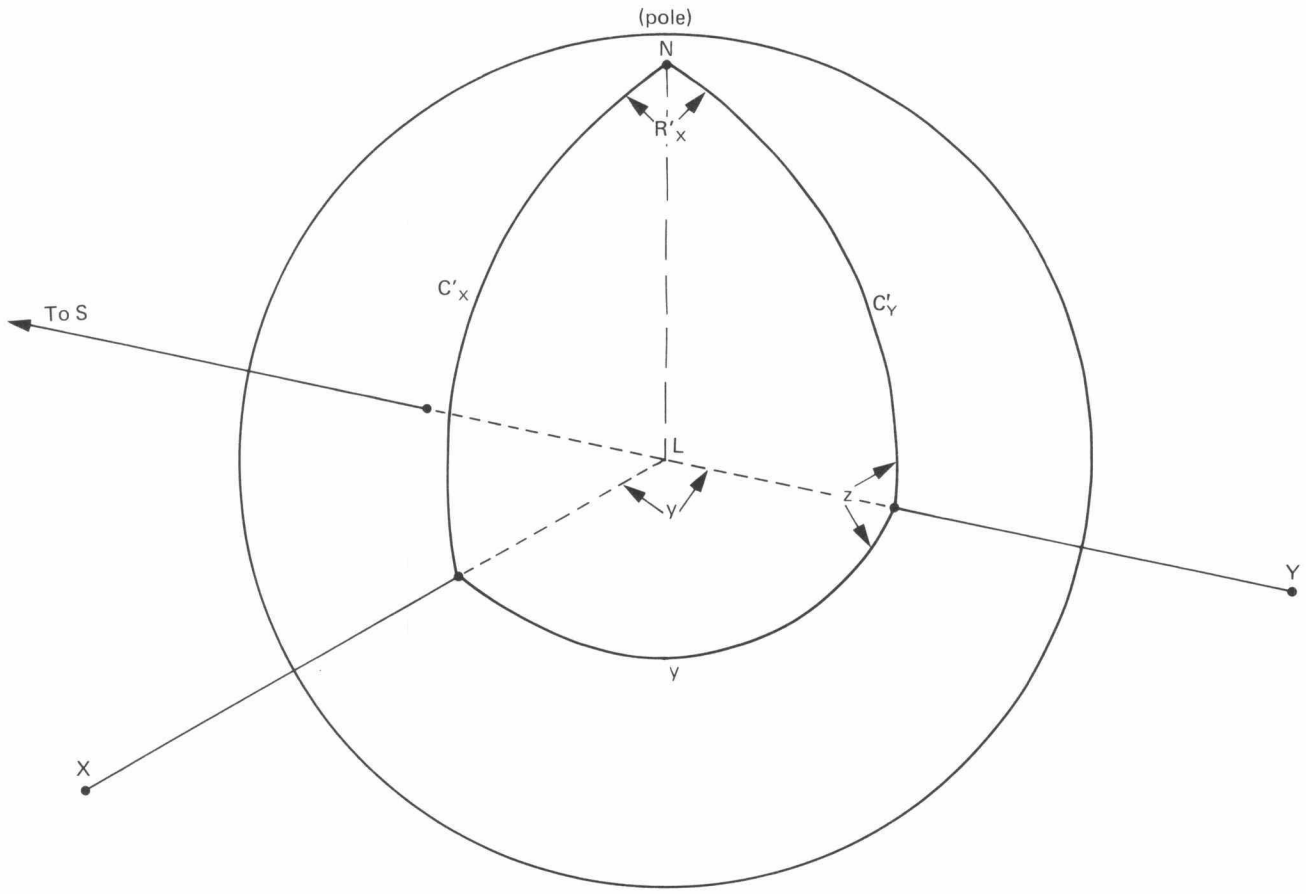
$$\tan z = \frac{\sin (R_Y - R_X)}{\frac{\sin C_Y}{\tan C_X} - \cos C_Y \cos (R_Y - R_X)}$$

Figure 2. Determination of Angle X

clination and right ascension (with respect to the reference direction) of star X observed at the spacecraft location.

Working on a Small Computer. A small computer can perform a more limited but interesting determination of the manner in which a familiar constellation changes shape as the spacecraft heads toward an arbitrary point in that constellation. Since the stars all diverge from the pole of approach, a simpler and less distorted polar plot is possible. Angle z is the polar angle and angle y is the radial coordinate, the angular "distance" from the pole. For convenience, relevant data and a sample calculation are given. The constellation is the Big Dipper and the destination point is the center star of the seven stars.

Sample Calculation. The calculation below is for the destination star and the observed end star of the handle at spacecraft distances of 0, 20, 40 and 60 light years from the solar system (63, 43, 23, and 3 light years respectively from the destination star). Visual magnitude is given by the formula: visual magnitude = absolute magnitude − 7.566 − $5 \times \log_{10}$ (distance). The visual magnitude of the destination star as seen from the solar system is 3.3 (dim star). At 20, 40 and 60 light years the visual magnitudes are 2.5, 1.14 and −3.28. Brightness



$$\tan R'_X = \frac{\sin z}{\frac{\sin C'_Y}{\tan y} - \cos C'_Y \cos z}$$

$$\cos C'_X = \cos C'_Y \cos y + \sin C'_Y \sin y \cos z$$

Figure 3. Determination of New Orientation Coordinates of Observed Star

increases with decreasing numerical value of visual magnitude. The system is a carryover from the practice of listing and numbering stars in order of brightness. The last visual magnitude is approximately that of Venus at greatest splendor.

To find the orientation of the end star of the handle from the destination pole, we have

$$\cos x = 0.8819 \times 0.8401 + 0.4714 \times 0.5424 \times \cos (2.891 - 3.204)$$
$$= 0.984$$
$$x = 10.22$$
$$\sin x = 0.1773$$

$$z = \tan^{-1}\left[\frac{\sin (2.891 - 3.204)}{0.5424 \times \frac{0.8819}{0.4714} - 0.8401 \times \cos (2.891 - 3.204)}\right]$$

$$z = -55.02^\circ$$

Thus, the end star is -55.02° (counterclockwise) from the north direction through our destination star and 10.2° away. As we travel toward the star, polar angle -55.02° remains fixed but the observed star moves radially outbound from the destination pole. At the first jump of 20 light years we have

$$y = 180^\circ - \tan^{-1}\left[\frac{\sin 10.22}{\frac{20}{105} - \cos 10.22}\right] = 12.60^\circ$$

At jumps of 40 and 60 light years from the sun, angle y is 16.38° and 23.25°. At 0, 20, 40 and 60 light years from the sun, the distances and visual magnitudes of the observed star are 105 and 1.84, 85.39 and 1.39, 66.02 and 0.83, and 47.17 and 0.10 respectively.

Performing the calculations for the five other observed stars of the world-renowned constellation and polar plotting the results will yield the Big Dipper as we see it, and in three altered patterns.

Background to the Stars. The annual voyage of the earth around the sun is miniscule in size compared to the enormous distances from our solar system to the nearest stars. Consequently, the differences in the geometric patterns of the stars as seen from opposite sides of the sun are too minute to be discernable to unaided earthling eyes. These differences were also too small for detection by the instruments, however sophisticated for their day, of the gifted astronomers from ancient Greece through the Renaissance, to the first practitioners of the telescope. These scientists carefully measured the angular separations of the stars and plotted these light points on a celestial globe, but found no semi-annual change in the patterns of the constellations (half the angular shift of a star is called the stellar parallax).

At the time of Copernicus and many years thereafter, the absence of detectable shift in the patterns of the constellations as seen from opposite sides of the sun was the strongest scientific argument against his heliocentric hypothesis. It was difficult to believe that stellar distances were so great that semi-annual star shifts were below detectable measurement. Furthermore, if these freakish distances were correct, then incredible strengths of outpouring luminous flux would be required of these "fires" of heaven if they were to be seen as far as earth. Thus, two unbelievable concepts accompanied the Copernican challenge to the old belief in a fixed earth at the center of the universe: incredible star distances and gargantuan light sources (both true).

Acceptance of these enormities required much time and perhaps new generations of earthlings.

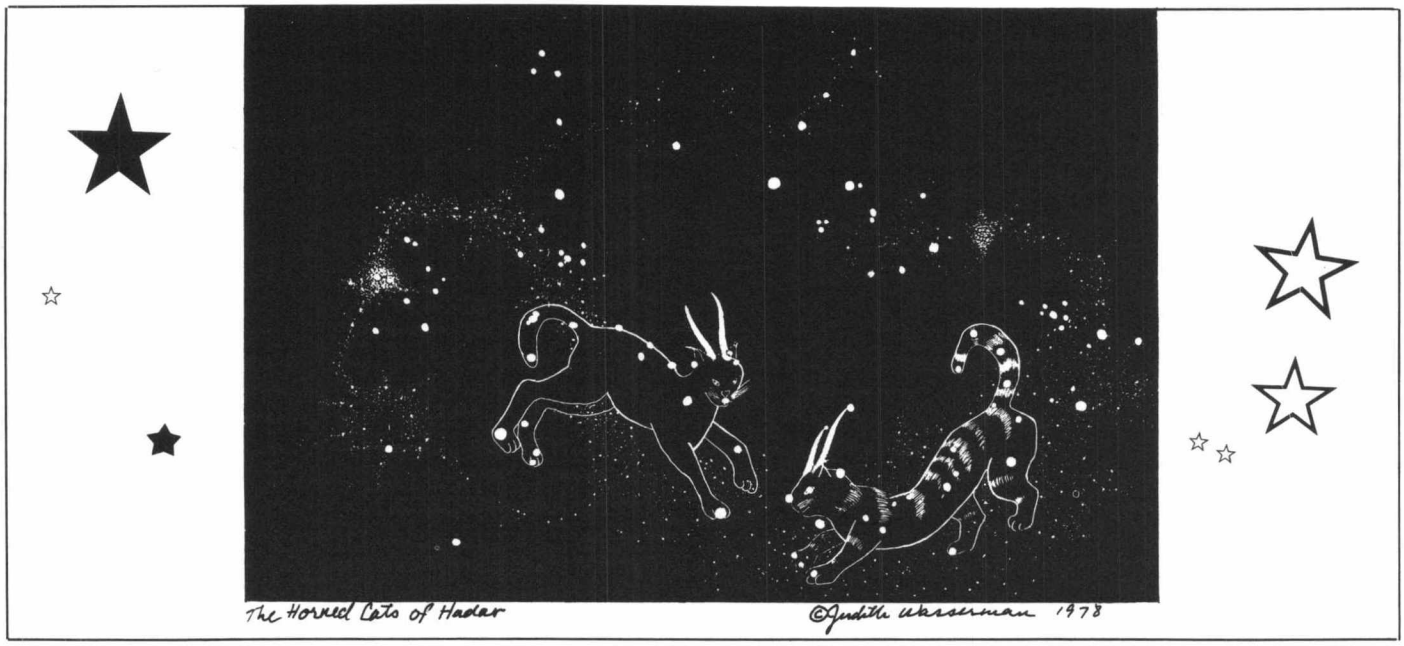
Ultimately, increasing evidence, more sophisticated planetary theory and mathematical simplicity forced the adoption of the Copernican view and carried the implications of stellar hugeness along with it.

The search for semi-annual star shifts continued. These shifts would tell how far away the stars were. Toward the middle of the Nineteenth Century, three centuries after Copernicus, precision telescopic instrumentation was sufficiently developed. In South Africa, Thomas Henderson measured the greatest possible shift: that of the nearest star, α Centauri. The shift was 1½ seconds of arc or about 1/1200 the diameter of the moon! The next nearest star, Barnard's Star, has a shift of just over one second of arc; all other star shifts are below one second of arc!

Semi-annual star shifts would be much more discernable to a freezing and dedicated astronomer on the planet Pluto. Pluto is the outermost planet of the solar system and is, on the average, forty times further from the sun than the earth. Star shifts are therefore forty times as great and the largest shift of α Centauri would be one minute of arc, about 1/30 the diameter of the moon and coincidentally the average angular diameter of the planet Venus. These shifts would take half a Plutonian year (124 earth years) and would probably be undetectable to the naked Plutonian eye.

So you see, dear reader (are you still there?) that to observe a substantial change in the patterns of the constellations, sufficient to interest an artist and to cause this artist to imagine new shapes and beings that inhabit the transformed heavens, we must leave the solar system far behind with the sun ultimately at our rear no matter the direction we travel. We must take a gargantuan voyage through our galaxy (the Milky Way) —gargantuan to us, but quite small compared to the size of the galaxy. In our first excursion, we observe fewer than 200 stars in a small suburb of a galaxy that has an estimated two billion stars.

And finally, a confession to two omissions: We have included only the brightest stars of our heavenly experience on earth and not many dimmer stars that may well become prominent in a galactic voyage. And we have assumed that in our travels, perhaps over many generations, the stars will remain motionless—which is not quite true.



Big Dipper Star Data

STAR (From end of Bowl)	RIGHT ASCENSION Radians	CODECLINATION Sine	Cosine	DISTANCE Light years	ABSOLUTE MAGNITUDE (Intrinsic brightness)
1.	2.891	0.4714	0.8819	105	-0.7
2.	2.882	0.5519	0.8339	78	0.5
3.	3.110	0.5904	0.8071	90	0.2
4.	3.204	0.5424	0.8401	63	1.9
5.	3.374	0.5582	0.8297	68	0.2
6.	3.504	0.5731	0.8195	88	0.1
7.	3.608	0.6505	0.7595	210	-2.1

Adding Star Data to Files

```
10 REM RFD3 - BYPYDAYBEE - ADD STAR DATA TO OLD/NEW FILES -
20 REM DAT/TIM: 10AUG/12:48PM
30 DIM A$[100],B$[100],C$[100],E$[100],F$[100]
40 DISP "NEW DATA FILE";
50 INPUT A
60 IF A=0 THEN 80
70 OPEN "SD6",3
80 X=180/PI
90 P=PI/2
100 REM NEW STAR ENTRIES
110 DISP "FIRST NUM OF NEW STAR ENTRIES";
120 INPUT I
130 I1=I
140 DISP "RA (HRS)";
150 INPUT U
160 DISP "RA (MIN)";
170 INPUT V
180 E1=(U+V/60)*15
190 C[I]=E1/X
200 DISP "DEC (DEG)";
210 INPUT J
220 DISP "DEC (MIN)";
230 INPUT K
240 C1=(J+K/60)
250 B[I]=COS(P-C1/X)
260 DISP "ABS MAG";
270 INPUT E[I]
280 DISP "DIST (LY)";
290 INPUT D[I]
300 PRINT I;U;V,J;K,E[I],D[I]
310 F[I]=D[I]^2
340 REM PRINT I;U;K;B;U;V;E1
350 I=I+1
360 GOTO 140
370 J=I-1
380 REM
390 REM FILE DATA
400 REM
410 FILES SD6
430 FOR I=I1 TO J
440 PRINT #1;B[I],C[I],D[I],E[I],F[I]
450 PRINT I;E[I];C[I];D[I];E[I];F[I]
460 NEXT I
470 PRINT
480 PRINT
490 REM CHECK: RETRIEVE DATA/CONVERT TO DEG/MIN/PRINT
500 READ #1,I1
510 FOR I=I1 TO J
520 READ #1;B[I],C[I],D[I],E[I],F[I]
530 PRINT I;B[I];C[I];D[I];E[I];F[I]
540 NEXT I
550 PRINT
560 PRINT
570 FOR I=I1 TO J
580 A[I]=SQR(1-B[I]^2)
590 B=90-ATN(A[I]/B[I])*X
600 IF B <= 90 THEN 620
610 B=B-180
620 A=INTB
630 B=(B-A)*60
640 C=C[I]*X/15
650 E=INTC
660 C=(C-E)*60
670 PRINT I;E;C;A;B;E[I];D[I]
680 NEXT I
690 END
```

Normal Star Map

```
10 REM RFD5 - BYBYBAYBEE:DATA CHECKOUT: RETRIEVE/CONVERT/PLOT NORMAL STAR MAP-
20 REM DAT/TIM: 18AUG/ 7:36AM
30 REM  DATA CHECKOUT: ASN FILES; GET DATA; CNV TO DEC/RA; CNV TO REL MAG; PLOT
40 DIM A$[100],B$[100],C$[100],D$[100],E$[100],F$[100]
50 X=180/PI
60 K=0
70 REM
80 REM ASSIGN FILES
90 REM
100 FOR N=1 TO 6
110 GOTO N OF 120,150,180,210,240,270
120 J=39
130 FILES SD1
140 GOTO 320
150 J=57
160 FILES SD2
170 GOTO 320
180 J=40
190 FILES SD3
200 GOTO 320
210 J=60
220 FILES SD4
230 GOTO 320
240 J=50
250 FILES SD5
260 GOTO 320
270 J=40
280 FILES SD6
290 REM
300 REM RETRIEVE AND PRINT DATA
310 REM
320 FOR I=1 TO J
330 READ #1;B[I],C[I],D[I],E[I],F[I]
340 PRINT I;B[I];C[I];D[I];E[I];F[I]
350 NEXT I
360 PRINT
370 REM
380 REM PLOT GRID
390 REM
400 K=K+1
410 IF K>1 THEN 480
420 READ X1,X2,X3,X4,X5,Y1,Y2,Y3,Y4,Y5
430 DATA 0,360,45,90,90,-90,90,45,90,90
440 GOSUB 740
450 REM
460 REM CONVERT/PLOT/PRINT
470 REM
480 FOR I=1 TO J
490 A=SQR(1-B[I]^2)
500 B=90-ATN(A/B[I])*X
510 IF B <= 90 THEN 530
520 B=B-180
530 A=INTB
540 B1=INT((B-A)*60)
550 C=C[I]*X
560 F=INT(C/15)
570 C1=INT((C/15-F)*60)
580 PRINT I;F;C1;A;B1;D[I];E[I]
590 S=D[I]
600 W2=E[I]-7.566+5*LGTS
610 IF W2>6 THEN 700
620 W1=(7-W2)^1.5/10
630 REM PRINT S,E[I],W2,W1
640 PLOT C,B+W1
650 PLOT C+W1,B
660 PLOT C,B-W1
670 PLOT C-W1,B
680 PLOT C,B+W1
690 PEN
700 NEXT I
710 PEN
720 NEXT N
730 END
740 SCALL X1,X2,Y1,Y2
750 XAXIS Y1,X3,X1,X2
760 YAXIS X2,Y3,Y1,Y2
770 XAXIS Y2,-X3,X2,X1
780 YAXIS X1,-Y3,Y2,Y1
790 FOR I=(Y1+Y4) TO (Y2-Y4) STEP Y4
800 PLOT X1,I
810 PLOT X2,I
820 PEN
830 NEXT I
840 FOR I=(X1+X4) TO (X2-X4) STEP X4
850 PLOT I,Y1
860 PLOT I,Y2
870 PEN
880 NEXT I
890 RETURN
```

Main Program
Sky at Four Spacecraft Distances

```
10 REM RFD4 - BYBYBAYBEE - MAIN PROGRAM: SKY AT FOUR S/C DISTANCES
20 REM DAT/TIM: 15AUG78/ 8:54PM
30 DIM GS[60],HS[60],US[60],VS[60],YS[60],ZS[60],TS[3,4]
40 DIM A$[60],B$[60],C$[60],D$[60],E$[60],F$[60],L[4]
50 X=180/PI
60 P=PI/2
70 K=0
80 MAT READ L
85 DATA 10,50,200,500
90 MAT PRINT L
100 PRINT
120 REM
130 REM SELECT DESTINATION
140 REM N=0: NEW DESTINATION
150 REM
160 DISP "NEW DESTINATION INPUT";
170 INPUT N
180 IF N=0 THEN 380
190 REM
200 REM NEW DESTINATION
210 REM
220 DISP "DEC (DEG)";
230 INPUT J
240 DISP "DEC (MIN)";
250 INPUT K
260 C2=J+K/60
270 C=P-C2/X
280 DISP "RA (HRS)";
290 INPUT U
300 DISP "RA (MIN)";
310 INPUT V
320 B1=(U+V/60)*15
330 R=B1/X
```

```
340 GOTO 690
350 REM
360 REM OLD DESTINATION
370 REM
380 MAT READ T
390 DATA 16,28.2,-26,-23,0,46,40,40,1,43.2,-16,-3
400 MAT PRINT T
410 PRINT
420 PRINT "DESTINATIONS"
430 PRINT "1. ANTARES"
440 PRINT "2. ANDROMEDA"
450 PRINT "3. TAU CETI"
460 PRINT "4. VEGA"
470 PRINT "5. SPICA"
480 PRINT "6. ARCTURUS"
490 PRINT "7. BETELGEUSE"
500 PRINT "8. ALPHA CENTAURI"
510 PRINT "9. ALPHA URSA MAJOR"
520 PRINT
530 PRINT
540 PRINT
550 PRINT
560 DISP "DESTINATION NUMBER";
570 INPUT N
580 REM
590 REM CONVERT DESTINATION COORDINATES
600 REM
610 J=T[N,3]
620 K=T[N,4]
630 U=T[N,1]
640 V=T[N,2]
650 C2=J+K/60
660 C=P-C2/X
670 E1=(U+V/60)*15
680 R=E1/X
690 S1=SINC
700 C1=COSC
710 PRINT N;U;V;J;K
720 PRINT E1;C1;C*X;S1;C1
730 PRINT
740 PRINT
750 REM
760 REM PLOT GRID
770 REM
780 READ X1,X2,X3,X4,X5,Y1,Y2,Y3,Y4,Y5
790 DATA 0,360,45,90,90,-90,90,45,90,90
800 GOSUB 1960
810 REM
820 REM ASSIGN FILES
830 REM
840 FOR N=5 TO 6
850 GOTO N OF 860,890,920,950,980,1010
860 FILES SD1
870 J=39
880 GOTO 1060
890 FILES SD2
900 J=57
910 GOTO 1060
920 FILES SD3
930 J=40
940 GOTO 1060
950 FILES SD4
960 J=60
970 GOTO 1060
980 FILES SD5
990 J=50
1000 GOTO 1060
1010 FILES SD6
1020 J=40
1030 REM
1040 REM RETRIEVE STAR DATA
1050 REM
1060 FOR I=1 TO J
1070 READ #1;B[I],C[I],D[I],E[I],F[I]
1080 PRINT B[I];C[I];D[I];E[I];F[I]
1090 NEXT I
1100 REM DETERMINATION OF TRIG FUNCTIONS OF DIHEDRAL AND CENTRAL
1110 PRINT "DRA","O","DIHED ANG","CEN ANG"                    ANGLES
1120 FOR I=1 TO J
1130 A[I]=SQR(1-B[I]^2)
1140 S=C[I]-R
1150 IF S>-PI THEN 1170
1160 S=S+2*PI
1170 IF S<PI THEN 1190
1180 S=S-2*PI
1190 O=SGNS
1200 U[I]=SIN(ABSS)
1210 V[I]=COSS
1220 REM DIHEDRAL ANGLE
1230 C3=U[I]/(S1*B[I]/A[I]-V[I]*C1)
1240 C4=ATNC3
1250 IF C4>0 THEN 1270
1260 C4=C4+PI
1270 G[I]=SINC4
1280 H[I]=COSC4
1290 REM CENTRAL ANGLE
1300 Y[I]=C1*B[I]+S1*A[I]*V[I]
1310 Z[I]=SQR(1-Y[I]^2)
1320 W=ATN(Z[I]/Y[I])
1330 IF W>0 THEN 1350
1340 W=W+PI
1350 IF I>10 THEN 1380
1360 PRINT S*X;O;C4*X;W*X;A[I]
1370 PRINT U[I];V[I];G[I];H[I];Z[I];Y[I]
1380 NEXT I
1390 REM
1400 REM DISTANCE LOOP
1410 REM
1420 FOR L=1 TO 4
1430 E=E[L]
1440 K=E^2
1450 PRINT
1460 PRINT "S/C DISTANCE NOW";E;"LIGHTYEARS"
1470 PRINT
1480 PRINT
1490 REM PRINT "S","D(I)","W1","W2","A*X","D","U1","V1"
1500 FOR I=1 TO J
1510 REM DETERMINATION OF DISTANCE FROM S/C TO OBSERVED STAR
1520 S=SQR(K+F[I]-2*E*D[I]*Y[I])
1530 W2=E[I]-7.566+5*LGTS
1540 IF W2>6 THEN 1570
1550 W1=(7-W2)^1.5/10
1560 REM DETERMINATION OF ANGLE FROM DESTINATION STAR TO OBSERVED
1570 A=ATN(Z[I]/(E/E[I]-Y[I]))                                STAR
1580 IF A>0 THEN 1600
1590 A=A+PI
1600 S2=SINA
1610 C2=COSA
1620 REM DETERMINATION OF DECLINATION OF OBSERVED STAR AT S/C
1630 C5=-C2*C1+S2*S1*H[I]
1640 S5=SQR(1-C5^2)
1650 U1=ATN(S5/C5)
```

Continued from pg. 33

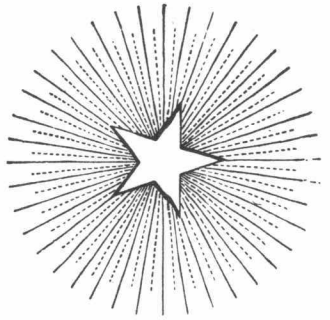
```
F8DD    ED4B F92A           LBCD    SAVEBC  ; Retrieve saved registers
F8E1    ED5B F92C           LDED    SAVEDE
F8E5    2A F92E             LHLD    SAVEHL
F8E8    DD2A F930           LIXD    SAVEIX
F8EC    FD2A F932           LIYD    SAVEIY
F8F0    ED7B F934           LSPD    SAVESP
F8F4    CD F830             CALL    LEAD    ; Punch leader (only useful for
                                            ;   Assm. PASS 3  but we
                                            ;   do it anyway).
F8F7    C3 F8A4             JMP     MEMRD
                    ;
                    ; Resets pointer to front of Editor buffer
                    ;
F8FA    C5          RESET:  PUSH    B
F8FB    01 0C01             LXI     B,EDBUFF; Load PTR with buffer
F8FE    ED43 F928           SBCD    PTR     ;  location
F902    C1                  POP     B
F903    C9                  RET
                    ;
                    ; This routine is called from a patch in
                    ; the Assembler when 'PASS=' is printed.
                    ;
F904    CD F8FA     RESETA: CALL    RESET
F907    CD 6AEC             CALL    PASS    ; This is in Assembler
F90A    C9                  RET
                    ;
                    ; Similar to above but from TOP when it requests
                    ; a character to start processing.
                    ;
F90B    CD F8FA     RESETW: CALL    RESET
F90E    CD 8006             CALL    CONCHR  ; In TOP
F911    C9                  RET
                    ;
F912    0000074C4F41 MSG:   .ASCII [0][0][07]'LOAD NEXT SEGMENT'[07]
0015                MSGL    = .-MSG
                    ;
F927    00          FLAG:   .BYTE   00
F928    0000        PTR:    .WORD   0000

F92A    0000        SAVEBC: .WORD   0000
F92C    0000        SAVEDE: .WORD   0000
F92E    0000        SAVEHL: .WORD   0000
F930    0000        SAVEIX: .WORD   0000
F932    0000        SAVEIY: .WORD   0000
F934    0000        SAVESP: .WORD   0000
                            .END
```

+++++ SYMBOL TABLE +++++

```
APPEND F874     ASSMB  5000     BEGIN  F842     CLRED  F89B
CO     F827     CONCHR 8006     CONT   F8D7     CRLF   F821
EDBUFF 0C01     EDITOR 0100     EDPRMT 01D7     FLAG   F927
FRONT  F886     LEAD   F830     MCO    F48A     MCRLF  F512
MEMRD  F8A4     MLEAD  F5A3     MRIFF  F474     MSG    F912
MSGL   0015     MTI    F736     MTOM   F452     OLDED  00B2
PASS   6AEC     PTR    F928     RESET  F8FA     RESETA F904
RESETW F90B     RESTRT F86E     RIFF   F82A     RSTED  0103
SAVE   F8B4     SAVEBC F92A     SAVEDE F92C     SAVEHL F92E
SAVEIX F930     SAVEIY F932     SAVESP F934     TI     F824
TOM    F82D     TOP    8000     ZSTART F053
```

```
1660 IF U1>0 THEN 1680
1670 U1=U1+P1
1680 U1=(P-U1)*X
1690 REM DETERMINATION OF RIGHT ASCENSION OF OBSERVED STAR AT S/C
1700 D=G[1]/(S1*C2/S2+C1*H[1])
1710 D=ATN(D
1720 IF D>0 THEN 1740
1730 D=D+P1
1740 V1=(R-P1-D*U)*X
1750 IF V1<360 THEN 1770
1760 V1=V1-360
1770 IF V1>0 THEN 1800
1780 V1=V1+360
1790 IF 1>10 THEN 1810
1800 PRINT S;D[1];W1;W2;A*X;D*X;V1;U1
1810 PLOT V1,U1+W1
1820 PLOT V1+W1,U1
1830 PLOT V1,U1-W1
1840 PLOT V1-W1,U1
1850 PLOT V1,U1+W1
1860 PEN
1870 NEXT I
1880 PEN
1890 REM NEW COLOR
1900 STOP
1910 NEXT L
1920 PEN
1930 NEXT N
1940 GOTO 160
1950 END
1960 SCALE X1,X2,Y1,Y2
1970 XAXIS Y1,X3,X1,X2
1980 YAXIS X2,Y3,Y1,Y2
1990 XAXIS Y2,-X3,X2,X1
2000 YAXIS X1,-Y3,Y2,Y1
2010 FOR I=(Y1+Y4) TO (Y2-Y4) STEP Y4
2020 PLOT X1,I
2030 PLOT X2,I
2040 PEN
2050 NEXT I
2060 FOR I=(X1+X4) TO (X2-X4) STEP X4
2070 PLOT I,Y1
2080 PLOT I,Y2
2090 PEN
2100 NEXT I
2110 RETURN
```

The End



An Unusual Pseudorandom Number Generator Program

H.T. GORDON
College of Natural Resources
University of California
Berkeley 94720

Many computer programs can generate a sequence of pseudorandom 8-bit binary numbers. Ideally, such a sequence requires that any number in the sequence have an equal probability of being followed by an other number (including itself). The ideal of perfect unpredictability of the next number may be unattainable, since this can always be predicted with complete certainty if both the generator algorithm and the numerical "keys" it will use are known. However, some algorithms can produce long sequences that satisfy statistical tests of randomness.

The goal of my work was to devise logic that would create the illusion of randomness, using a minimum of code and execution time. The starting point was my reading of the (13N + 1) algorithm by Daniel Greiser in the Nov 1977 *BYTE*, my first encounter with simple, fast algorithms of this kind. Since then, my conception of the problem has undergone many sea-changes. I have come to the point where I can no longer see how more could be done with less. It is time for fresher, keener minds to look at the problem!

Linear Core Algorithms

One requirement is that (in a long sequence) each of the 256 binary numbers occur at equal frequency. This is met by many simple algorithms of the form: $N_{i+1}=(4a+1)\ N+c$, where c must be an odd number. If $a=0$, the sequences are obviously non-random, and can easily be "deciphered" from a very short sequence. If $a=1$, the sequence has a much more random look. This (5N + 1) algorithm is the basis of my new subroutine SIMRND (650X-coded in the listings). My original thinking-out of this type of algorithm (in a MS sent to *BYTE* a year ago but not yet published) was in error, and caused me to use the unnecessarily complex (5N + $2B) algorithm, which works but codes less efficiently.

The SIMRND output is a sequence of the 256 different binary numbers which endlessly repeat. As with all linear algorithms, the "randomness" is in the high-order bits. If one writes the sequence as a 16 x 16 matrix, the low nibbles show a characteristic pattern: the first row has a sequence of all 16 possible low nibbles (with alternating odds and evens which are repeated exactly in each succeeding row (so each column has 16 identical low nibbles).

Evaluation of the Degree of Disorder in Short Numerical Sequences

If any number N_1 has an equal probability (1/256) of being followed by *any* other number N_2, all values of the *difference* $(N_2 - N_1)$ have a probability of 1/256. One way of analyzing a short sequence uses the concept of *heterogeneity*. The *zero-order* heterogeneity, H_0, is that of the sequence itself. It can be defined as the number of *different numbers* occurring in a sequence of 256. The maximum H_0 is of course 256. The *first-order* hereogeneity, H_1, requires a sequence of 257 numbers, from which the first-order sequence of 256 differences between successive numbers is calculated. The maximum H_1 is again 256. The concept and operation can be extended to H_2, H_3, etc. An interesting theoretical problem is to what extent a large value of H_0 is compatible with large values of H_1, H_2, etc. Another problem which I lack mathematical skill to solve is what values of H have the highest probability; although there exist an incredibly large number of different sequences with $H_0 = 256$, there are even more sequences with values of 255 or less, so it seems unlikely that a truly random sequence will have a maximum H_0.

To test the output of various algorithms, I wrote a KIM-1 program that generates sequences of increasing orders of heterogenity and calculates the approximate H values. I do not present the listing here because it is too specialized to be of general interest, but anyone who sends me an SASE can have a photocopy of it. The test results for 5 different linear algorithms are given in Table 1. (This also includes the effect of various "jumbling" routines, to be described in the next section of this note). All have an $H_0 = 256$. The simple (N + $7F) algorithm, of course, has $H_1 = 1$, the minimal value (it must remain so for all higher orders). The more complex algorithms are much more heterogeneous. The (5N + 1) and (13N + 1) have identical values, declining from 64 at H_1 to 1 at H_4, showing a fairly complex kind of order has been generated. The (9N + 1) is less heterongeneous, and the (17N + 1) still less. It is clear that the most useful algorithms are in fact of the type $(8n+5)N + 1$, where n is any number from 0 on up. Nevertheless, there is an underlying orderliness, quickly unraveled by the test program.

The Creation of Disorder by Jumbling Algorithms

In my (rapidly obsolescing!) BYTE MS, I introduced a second algorithm to "jumble" the output of the core algorithm. The underlying principle is that one can define *subsets* of the set of binary numbers, and *operations* that uniquely transform each number in a subset into another number in the same subset. The effect is to *relocate* these numbers in an order different from that in the original sequence.

My first try used the concept of *parity* to define 2 equal subsets: one in which bit 7 = bit 6, and one in which bit 7 ≠ bit 6. Every subset in which 2 (or any even number of) bits have the same (odd or even) parity has the useful property that the *complement* of every number is the number that, when itself complemented, will be the original number. An algorithm which recognizes numbers in which bit 7 ≠ bit 6, and complements all of them, will *exchange* pairs of complementary numbers within the original sequence. My original coding of this algorithm was very inefficient, using the 650X BIT instruction and the status of the N and V flags to identify numbers with bit 7 ≠ bit 6. In my new coding (the COMJUM module in the listings) this is done by a single CMP #$C0 instruction, that sets the N flag if the number in the accumulator has bit 7 ≠ bit 6. The significance of the "sign" flag in COMPARE instructions is seldom well explained in programming manuals, so it is used less often (and sometimes incorrectly) than the carry and zero flags in decision-making. For any given comparand, the N status subdivides the 256 possible bit-patterns in the accumulator into 2 equal sets. The comparand $C0 sets the N flag only if the accumulator contains a number in the range from $40 to $BF, which is the set of all numbers in which bit 7 ≠ bit 6.

When I first used the "jumbling" concept, I had not thought it out properly, and so did not realize that *the core algorithm itself* defines 2 equal subsets, since the N flag is set only by the 128 numbers with bit 7 = 1. If one uses an EOR #$7F instruction, to avoid altering bit 7, one can "exchange" number-pairs in one of these 2 subsets in which the 7 low-order bits are complementary. This is the simple logic of the SIMJUM module in the listings. Its effect on the heterogeneity of the output of various core algorithms is shown in Table 1, and is *identical* with the effect of the COMJUM module. Used with the ultrasimple (N+$7F) algorithm, it gives a remarkably high value of H_1. However, the output of the jumbled algorithm is not all that disorderly, as shown by the rapid decline in higher-order H values. Most of the high H_1 signifies a more complex order. Even if one goes to H_{32} the value does not decline to 1 (and even shows a very gradual, erratic rise).

With the more complex algorithms, SIMJUM creates a similar large increase in disorder (insofar as H values define it), most of it persistent in higher-order H analysis. This is obviously not *true* disorder, since a simple "dejumbling" of the output would allow my H test program to resolve it. However, it may be a good *simulation* of true disorder.

Since jumbling based solely on the status of bit 7 had such an interesting effect, I also explored bit 6 by the JUMSIX algorithm (cf. listings). As shown in Table 1, jumbling on the bit 6 status has a very small effect on the (N+$7F) algorithm (however, it has a much larger one on the (N+$3F) algorithm, with an H_1 of 65). With the more complex algorithms, JUMSIX creates less heterogeneity than SIMJUM, especially on the ones on which SIMJUM is most effective.

Having at last recognized that each bit defines a pair of equal subsets which are (in principle) independent, I was able to see that the CMP #$C0 of the CONJUM logic actually defines *four* subsets. One way of defining these and causing some exchange of complementary pairs in all four of them is shown as module TETJUM in the listings. The operands of the 3 EOR instructions can be altered, but the effects are minor. However, the effect on heterogeneity is not markedly different from that of SIMJUM.

A quite different way of defining and altering the original sequence is module ROLJUM (cf. listings). Again there is no greater heterogeneity than with SIMJUM. The jumbling information in bits 7 and 6 seems not to be independent.

The H_1 value of 187 differences (in Table 1 for the (5N+1) algorithm with SIMJUM) is not the possible maximum. The RISJUM module (cf. listings) gives an H_1 of 205 differences with (5N+1), but successive values decline to 64, 63, 16, 16, 4, 4, and 2 (the stable value), again proving that a high H_1 may simply indicate very complex order. If the SIMJUM module is used first, to introduce "stable disorder", then followed by RISJUM, there is usually a small enhancement of the H values. Note that the SIMJUM EOR operand, however, must be changed from $7F to $7E, so as not to alter either bit 7 or bit 0; the reason is that RISJUM is jumbling on the status of bit 0 (moved into the carry flag by the LSR A instruction). It is vital (to retain an H_0 of 256) that the bit or bits on which jumbling is based *not be changed* by the operation. The reason why RISJUM alone cannot introduce significant stable disorder is that bit 0 is the least disorderly bit in the zero-order sequence, alternating between 0 and 1. However, I now doubt that *any* multiple-jumbling will be worth the cost.

A friend pointed out to me that the sequence: N, N+1, N-1, N+2, N-2 . . . . . N+$80 will contain all 256 different numbers and 255 of the possible differences (except 0); since the 257th number will be equal to the 256th, H_1 must be 256. I generated and tested this sequence and found successive H values of 256, 64, 64, 16, 16, 4, 4, and 1. The extreme orderliness of this sequence is revealed by adding SIMJUM to the generator. H_0 is 256, but H_1 crashes into 3 (and eventually to a stable value of 2). A very high H_1 value is compatible with (and may well *indicate*) extreme orderliness! So the *higher* orders of H seem to be a better criterion.

I have emphasized the concept of orders of heterogeneity because it can (and will) be used to resolve the underlying order in pseudorandom sequences. Dejumbling tools will be needed as well. Whatever has been raveled by rigid logic can be unraveled (though it may take a vastly more complex program to do so). I have dealt only with the simplest logic which can be unlocked with keys of very few bits. With more elaborate logic and keys, an infinite variety of pseudorandom sequences can be generated. While a deciphering program could soon recognize the pseudorandomness, recreating the generator logic might be extremely difficult!

Creating Sequences Longer than 256 Numbers

In my original design, I used 2 more zero-page "working registers" (in addition to the RND seed location): one worked as a "same-sequence-of-256" counter, incremented at every call to the subroutine, while the other was an "addend", incremented only when the counter became zero (i.e., at the start of each new sequence). The content of the "addend" location

was added to the output of the core algorithm before jumbling of the accumulator content, to give the final output number. This added 9 program bytes and 15 microseconds execution time per number generated. Since the primitive version was both code-efficient and time-inefficient, this seemed trivial. The new SIMRND-SIMJUM coding needs only 14 program bytes and executes in 31 microseconds, so that the simpler INCRND module (cf. listings) seems preferable since it adds only 6 program bytes and 8 microseconds. It also lengthens the non-repeating sequence to 65K. Both the slower "addend" operation and the INCRND operation would be easily decipherable. The former has the desirable feature that every one of the 190-odd "differences" follows every possible binary number (a good "randomy" touch), but a good deciphering program would soon recognize output (N+256) is generally output (N)+1. While the INCRND operation does not have this flaw, the decipherer would soon see every output number is usually (not always, because of jumbling) followed by a completely predictable number. In both, the fact that many first-order differences *never* occur, while many others occur at twice the expected frequency, would let a lot of cats out of the bag!

Both of the surface flaws (but not the deeper one of low heterogeneity) are corrected fairly simply in subroutine DUBRND (cf. listings), at the cost of 5 program bytes and 5 microseconds execution time. This can be viewed as a sequence of 4 modules: DUBRND, a minor complication of INCRND which decrements a new zero-page "addend" working register (in addition to incrementing the RND seed) at the start of each new sequence; SIMRND; ADDRND, which starts the "variation" of the output by adding the sequence addend; and SIMJUM. While there are still only 256 different sequences before exact repetition occurs, every one has a different initial seed and a different addend. The decipherer might need a bit more time to figure out what was going on, since the addend operation produces (when the addend is an odd number) sequences that no linear core algorithm could yield.

Nothing would be easier than to produce greater complexity and much longer non-repeating sequences. But *why?* Unless one could correct the heterogeneity flaw, this would not create a puzzle of genuine interest to cryptologists. This is a good point to argue the value of speed and aperiodicity in pseudorandom generators (code efficiency, always nice if you can get it, seems less important here). If used with a fast D/A converter to simulate "white noise", SIMRND alone would cause output fluctuations at frequencies near 60,000 Hz, but its short non-repeating sequence means periodicity near 400 Hz. The INCRND-SIMRND combination would lower the peak frequency to near 40,000 Hz, but would have an exact-repetition frequency of less than 1 Hz. (Some added coding might be needed to equalize the timing of each operation; this would lower these values slightly.) A repeating sequence of 256 may be too short, while one greater than 65K may be too long.

Improved heterogeneity, not length of the non-repeating sequence, is the challenge to theoretical minds of a higher order than my own!). Given enough complex logic, I have no doubt that a much closer approximation to true randomness than that achieved by my simple logic is attainable, especially if the keys are initialized (something not required by my logic). But if this involves a large increase in execution time, it will be useful only to the few who possess superfast machines. It seems to me that mere statistical tests are not searching enough to reveal the orderliness in short sequences; many routines that (with long sequences) pass such tests with flying colors would fail the heterogeneity test.

Decimalizing the Pseudorandom Sequence

This can be done simply by a module like SELDEC (cf. listings), which allows only the 100 "natural" decimal numbers (00 to 99) to be output, while rejecting the 156 numbers that have a hex value in either high or low nibble. Although SELDEC needs only 13 program bytes, the mean execution time per decimal number output is (2.56 T+32.6) microseconds, where T is the mean execution time of the binary generator being used. Even if the binary generator is SIMRND alone, the execution time per decimal number output is over 70 microseconds, and the timing penalty rises quickly as the binary generator becomes more complex. Also, the length of the non-repeating sequence is only 25K.

The more complex (25 program byte) module DECRND is much faster, with a mean execution time per decimal number output of (1.28 T+24.4) microseconds. Even with SIMRND alone, where its advantage is minimal, the mean time is less than 45 microseconds, and increases only to 65 microseconds with the complex DUBRND/SIMRND/ADDRND/SIMJUM generator logic (that with SELDEC requires over 110 microseconds per decimal output).

The operation of DECRND outputs 200 decimal numbers (each repeated once, but in a non-repeating sequence of 200) for every 256 binary numbers output by the basic generator. It first detects the 160 binary numbers that have a decimal value in the high nibble and passes these through a decimal-adjust module. All the "natural" decimals emerge unaltered. The "unnaturals", 60 numbers of the type (0-9)(A-F) emerge "adjusted" to duplicates of type (0-9)(0-5). The task of the SPECOP module, to which the 96 numbers of type (A-F)(0-F) were sent, is to reject 56 and convert the other 40 into the "missing" duplicates of type (0-9)(6-9).

SPECOP clears the carry and does 2 left-rotates through the carry. It can now reject the 32 numbers of the original type (A,B)(0-F), that have cleared the carry. With 2 more left-rotates, the previously low nibble (0-F) is now the high, and the low nibble is (6,7) with carry either set or clear. One of the (6,7) sets is accepted, the other is converted to (8,9) by an EOR #$0E. Finally it uses a CMP #$A0 to reject the 24 numbers with a hex high nibble and allows the 40 of type (0-9)(6-9) to exit.

DECRND does not output all the 100 different decimals in any sequence of 100 consecutive numbers (H_0 is near 50). Some numbers occur once, others are duplicated, and many are absent; this is corrected in the next 100 numbers. It is not only much faster than SELDEC, but (with a 65K binary generator) yields a non-repeating sequence of 51K decimals. One can concatenate successive outputs to form large numbers of $2n$ decimal digits; if n is odd, wraparound allows generation of 51K different large numbers before exact repetition. The degree of "stable disorder" is probably higher than that with SELDEC, and perhaps even higher (as a percentage of the maximum) than that of the binary generator since decimalizing is a kind of jumbling. I have not tested this because I am not much

interested in decimals. For the many who are, DECRND may be useful. For me, the interest lies in the unusual (less easily decipherable) conversion of hex to decimal numbers, a process far more complex than my hex-jumbling modules, which hints to really elaborate jumbling operations being able to create higher levels of heterogeneity.

Modules, Superinstructions, and Macros

All are blocks of code, goal-oriented to produce a desired effect, and *insertable* into programs that require the effect. Insertion saves the time lost by a subroutine call and return. The word "module" is quite general. Macros tend to be rather big modules. I view my simpler modules as "superinstructions". In fact, the idea for the RISJUM module came from an earlier exploration of the emulation in 650X code of useful instructions missing in the set. The ancestor of RISJUM used an EOR #$80, and emulated a rotate-right-without-carry operation. I presume that emulator programs use this kind of thing to convert code from one kind of instruction set to another.

The creation and recognition of modular structure in programs allows many variants of an operation, each fitting the particular need for speed, code-efficiency, or complexity in the unending game of trade-offs. Many programs contain well-written (and well-hidden) modules to do various tasks, whose value as optimized superinstructions usable in other programs is given no emphasis. Similar tasks in other programs are often badly coded (unless the programmer re-invents the optimal coding). A rich module/subroutine library would make programming both easier and more efficient.

Copyrights

I am copyrighting the listed modules (and their combinations) with the same "free-diffusion" clause used for my program EDITHA (*DDJ* #25). Note that the f-d-c allows totally unrestricted use in association with all f-d-c software, but does not *forbid* its use in programs protected by the usual copyright. However, in such programs, f-d-c software is entitled to protection *equal* to that of the associated software (use only by prior permission). I am aware that a lot of unacknowledged use of software is going on. As an advocate of diffusion, I do not disapprove of this. I *would* object to a "double-standard" in which "borrowers" incorporated f-d-c software into their programs without permission, then made waves if someone else "borrowed" *their* work. This raises the question: what about coding of these algorithms for other micros than the 650X? You cannot copyright an algorithm. Thousands of people could easily code them for the 8080 etc.—who, if anyone, can then claim an exclusive copyright?

Table 1. Heterogeneity Numbers (H_1-H_4) for Linear Core (+ Jumbling) Algorithms

CORE	alone	SIMJUM	RISJUM	SIMJUM RISJUM	JUMSIX	TETJUM	ROLJUM
(N + $7F)	1	129	129	5	3	132	65
	1	66	64	4	5	70	38
	1	36	64	7	4	43	22
	1	20	16	10	5	28	22
(5N + 1)	64	187	205	187	119	148	161
	16	158	64	183	100	157	143
	4	143	63	158	90	163	144
	1	146	16	145	96	166	147
(9N + 1)	32	137	199	163	105	138	135
	4	118	64	161	86	136	124
	1	137	64	150	79	118	140
	1	141	16	162	93	142	125
(13N + 1)	64	175	198	179	127	158	162
	16	160	64	182	108	158	146
	4	162	63	158	70	159	150
	1	160	16	171	98	171	147
(17N + 1)	16	107	193	151	89	124	121
	1	130	64	140	90	122	121
	1	123	64	132	84	131	134
	1	133	16	140	95	161	130

```
A5 C1     SIMRND LDA RND (load seed)
ØA               ASL A   (X 2)
ØA               ASL A   (X 4)
38               SEC     (to add 1)
65 C1            ADC RND (X 5 + 1)
85 C1            STA RND (next seed)

C9 CØ     COMJUM CMP #$CØ (bit 7 = bit 6?)
1Ø Ø2            BPL NEXT (yes, retain #)
49 FF            EOR #$FF (no, complement)
            NEXT

1Ø Ø2     SIMJUM BPL NEXT (bit 7 = Ø, retain #)
49 7F            EOR #$7F (complement all but 7)
            NEXT

4A        RISJUM LSR A (bit Ø to carry, zero b7)
9Ø Ø2            BCC NEXT (bit Ø = Ø, retain #)
49 FF            EOR #$FF (bit Ø = 1, complement)
            NEXT

1Ø Ø2     SIMJUM BPL RISJUM (for 2 jumblings)
49 7E            EOR #$7E

24 C1     JUMSIX BIT RND (bit 6 to V flag)
5Ø Ø2            BVC NEXT (if = Ø, retain #)
49 BF            EOR #$BF (complement all but 6)
            NEXT

C9 CØ     TETJUM CMP #$CØ (quad jumbling)
1Ø Ø2            BPL OMIT
49 FE            EOR #$FE
49 Ø1       OMIT EOR #$Ø1
1Ø Ø2            BPL NEXT
49 21            EOR #$21
            NEXT

C9 CØ     ROLJUM CMP #$CØ  (quad jumbling)
1Ø Ø5            BPL ROLOP (half have set carry)
49 FF            EOR #$FF (sets N for half)
1Ø Ø1            BPL ROLOP (N and carry clear)
38               SEC     (N and carry both set)
2A         ROLOP ROL A  (all  numbers new)
            NEXT

E6 C2     INCRND INC COUNT
DØ Ø2            BNE SIMRND
E6 C1            INC RND (next sequence of 256)
           SIMRND

E6 C2     DUBRND INC COUNT
DØ Ø4            BNE SIMRND
E6 C1            INC RND
C6 C3            DEC ADDEND
```

Continued on pg. 19

Let Your Computer Speak ASCII

BY G. GAUGLER
2276 Beaver Valley Rd.
Fairborn, OH 45324

Having wrestled with several unyielding programs and having talked to my computer at great length, I decided it was time for it to talk back. Telesensory Systems, Inc. has produced a speech synthesis board which, when properly interfaced, can give your computer an ASCII voice of its own. Interfacing this board to a SWTPC 6800 is not a particularly difficult task. The hardware and software necessary to perform several different functions will be described.

The Telesensory Systems, Inc. (TSI) S2C speech synthesis board uses a PMOS dedicated microprocessor and two large ROMs to generate digitized speech consisting of a 64 character ASCII vocabulary. The speech data consisting of pronunciation and duration are contained in the two ROMs. A six bit binary code representing the character to be spoken is presented to the speech synthesizer board (SSB). A positive strobe pulse is sent to the SSB telling it to start the speech generation. While the speech is being synthesized, the SSB generates a BUSY signal which will be low while the synthesis is in progress. When the synthesis is complete, the BUSY line returns high. The synthesized speech consists of digitized data and must be filtered to remove the impressed clock and to make the audio sound as natural as possible.

The requirements placed on the software and hardware interface consist of two main tasks: 1) obtaining the proper binary code for the desired character, and 2) handshaking between the processor and the SSB from time of speech start to stop.

Fig. 1 is a schematic of the hardware needed to interface the SSB. A simple audio amplifier is included which could be replaced by an IC audio amplifier or an external audio system. The interface can be constructed on a wire-wrap board or a suitable perf-board. Referring to the schematic, the PIA (MC6821) is used to transfer the chosen character to the SSB and to handshake between the processor and the SSB. The software must initialize the PIA before any transfer can begin. The software provided in Listing 1 will perform the following functions: initialize the PIA, cause the ASCII character in the A-register to be converted to speech, fetch a character from the keyboard and convert it to speech, and a routine to speak an ASCII character string (equivalent to a vocal PDATA1 in MIKBUG).

The SSB is available for $180 from Telesensory Systems, Inc., 3408 Hillview Ave., Palo Alto, CA 94304.

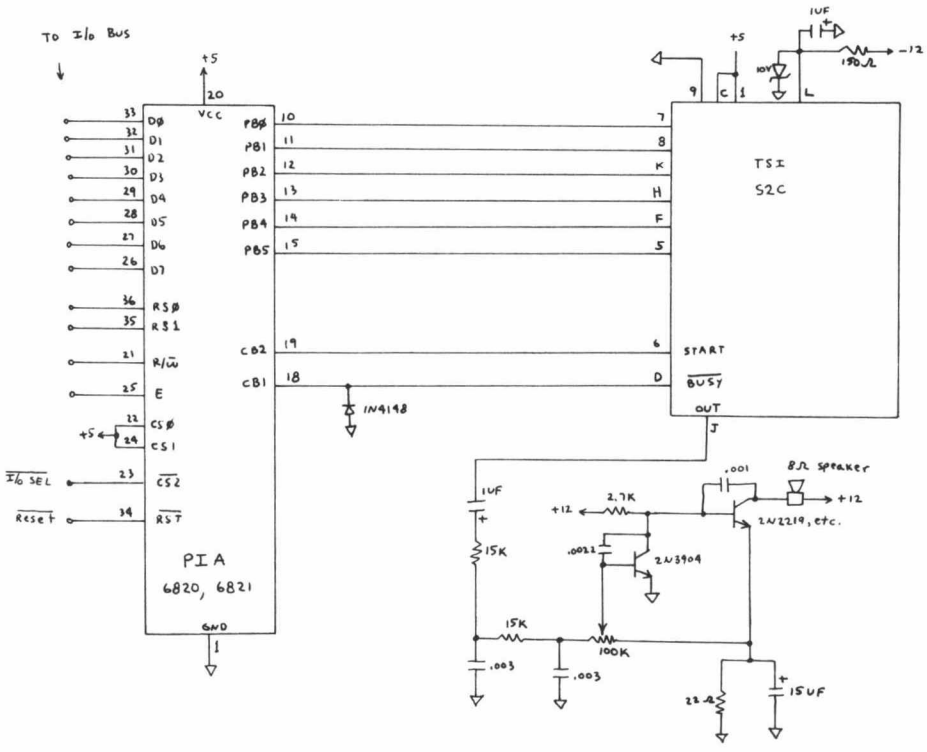


Figure 1

```
TTTTTTTT    AAAA    LL        KK    KK
TTTTTTTT   AAAAAA   LL        KK   KK
   TT      AA  AA   LL        KK  KK
   TT      AA  AA   LL        KK KK
   TT      AAAAAAAA LL        KKKK
   TT      AAAAAAAA LL        KKKK
   TT      AA  AA   LL        KK KK
   TT      AA  AA   LLLLLLLL  KK  KK
   TT      AA  AA   LLLLLLLL  KK   KK


MM    MM    AAAA      CCCCC
MMM  MMM   AAAAAA    CCCCCCC
MMM  MMM   AA  AA   CCC  CCC
MM MM MM   AA  AA   CC
MM MM MM   AAAAAAAA CC
MM    MM   AAAAAAAA CC
MM    MM   AA  AA   CCC  CCC
MM    MM   AA  AA    CCCCCCC
MM    MM   AA  AA     CCCCC
```

```
 1                      NAM    TALK

 3                * DRIVERS FOR THE TSI MODEL S2C SPEECH SYNTHESIZER.
 4                * INCLUDED ARE A KEYBOARD TO-SPEECH ROUTINE AND A
 5                * SPEECH STRING (VOCALIZED PRINT STRING) ROUTINE.

 7                * WRITTEN BY GARY D. GAUGLER

 9                * EXTERNALS

11 E1AC          INCH   EQU    $E1AC       GET CHAR FROM KEYBOARD
12 8002          PIABD  EQU    $8002       B-SIDE DATA PORT
13 8003          PIABC  EQU    $8003       B-SIDE CONTROL/STATUS

15                * CONSTANTS

17 0037          LOW    EQU    $37         STROBE OUTPUT=0
18 003F          HIGH   EQU    $3F         STROBE OUTPUT=1


20                      OPT    PAG*NOG

22                * THIS IS THE INITIALIZATION DRIVER FOR THE
23                * PIA. THIS ROUTINE ASSUMES THAT THE PIA IS
24                * LOCATED AT PORT 0. CHANGE AS NEEDED.

26 0000 4F       INZ    CLR A
27 0001 B7 8003         STA A  PIABC       GET AT DATA DIR REG
28 0004 43              COM A
29 0005 B7 8002         STA A  PIABD       B-SIDE=OUTPUTS
30 0008 86 37           LDA A  #LOW        STROBE IS LOW
31 000A B7 8003         STA A  PIABC
32 000D 39              RTS


34                * THIS ROUTINE STORES THE VALUE OF THE CHARACTER
35                * TO BE SPOKEN INTO THE B-SIDE OF THE PIA.  A
36                * POSITIVE STROBE IS THEN GENERATED TO START THE
37                * SPEECH.  A RETURN IS PERMITTED AFTER THE SPEECH
38                * BOARD HANDSHAKES A COMPLETION.

40 000E 80 20    TALK   SUB A  #$20        STRIP BIAS
41 0010 B7 8002         STA A  PIABD       STORE CHAR AT OUTPUT PORT
42 0013 86 3F           LDA A  #HIGH       TAKE STROBE HIGH
43 0015 B7 8003         STA A  PIABC
44 0018 86 37           LDA A  #LOW        NOW TAKE IT LOW
45 001A B7 8003         STA A  PIABC
46 001D 7D 8003 WAIT    TST    PIABC       WAIT TILL SPEECH DONE
47 0020 2A FB           BPL    WAIT
48 0022 B6 8002         LDA A  PIABD       CLEAR IRQ FLAG
49 0025 39              RTS
51                * ROUTINE TO GENERATE SPEECH FROM THE KEYBOARD
52                * KEYS AS THEY ARE PRESSED.

54 0026 BD 0000 SAYKBD JSR     INZ         SET-UP PIA
55 0029 BD E1AC SAYKB0 JSR     INCH        GET CHARACTER
56 002C BD 000E        JSR     TALK        OUTPUT SPEECH
57 002F 20 F8          BRA     SAYKB0      GET NEXT FOREVER


59                * SPEECH STRING ROUTINE. (X)<-- ADDRESS OF START
60                * OF TEXT. TERMINATES ON EOT (04).

62 0031 A6 00    SAYLST LDA A  0,X          GET CHAR
63 0033 81 04           CMP A  #4           EOT?
64 0035 27 12           BEQ    DONE
65 0037 BD 000E         JSR    TALK         SEND OUT SPEECH
66 003A 08              INX
67 003B FF 004A         STX    TEMX         SAVE X-REG
68 003E FE CFFF         LDX    $CFFF        NEED A DELAY BETWEEN..
69 0041 09       DELAY  DEX                 ..SPOKEN CHARACTERS
70 0042 26 FD           BNE    DELAY
71 0044 FE 004A         LDX    TEMX         RESTORE X-REG
72 0047 20 E8           BRA    SAYLST       GET NEXT CHAR
73 0049 39       DONE   RTS
74 004A          TEMX   RMB    2            TEMPORARY X-REG STORAGE


76                      END

   SYMBOL TABLE:
DELAY  0041   DONE   0049   HIGH   003F   INCH   E1AC   INZ    0000
LOW    0037   PIABC  8003   PIABD  8002   SAYKB0 0029   SAYKBD 0026
SAYLST 0031   TALK   000E   TEMX   004A   WAIT   001D
```

Continued from pg. 17

```
         SIMRND
18       ADDRND CLC
65 C3           ADC ADDEND
         SIMJUM

C9 AØ    SELDEC CMP #$AØ (hi nibble hex?)
BØ XX           BCS REJECT (yes, get next #)
AA              TAX   (save # in X reg.)
29 ØF           AND #$ØF (retain lo nibble)
C9 ØA           CMP #$ØA (lo nibble hex?)
BØ XX           BCS REJECT
8A              TXA  (restore decimal #)
6Ø              RTS

C9 AØ    DECRND CMP #$AØ
BØ Ø5           BCS SPECOP (hi nibble hex)
F8              SED     (set dec. mode)
69 ØØ           ADC #Ø  (add zero)
D8              CLD    (clear dec. mode)
6Ø              RTS  (exit; 160 decimals)
18       SPECOP CLC
2A              ROL A
2A              ROL A  (bit 6 in carry)
9Ø XX           BCC REJECT (32 numbers out)
2A              ROL A
2A              ROL A
9Ø Ø2           BCC ACCEPT (keep some 6,7)
49 ØE           EOR #$ØE  (6,7 to 8,9)
C9 AØ    ACCEPT CMP #$AØ
BØ XX           BCS REJECT (24 numbers out)
6Ø              RTS   (40 decimals exit)
```

coding for SIMRND (versions 1 or Ø)
(no addresses, fully relocatable)

```
A5 C1           LDA RND (load core seed)
18              CLC
69 2B           ADC #$2B (add constant)
85 C1           STA RND (store next seed)
E6 C3           INC COUNT (new sequence?)
DØ Ø5           BNE NORMAL (no, same)
E6 C2           INC ADDEND (new addend)
A5 C4          *LDA PREOUT (previous output)
6Ø             *RTS   (repeat   "     "   )
18       NORMAL CLC
65 C2           ADC ADDEND (add the addend)
C9 CØ           CMP #$CØ  (bit 7 = bit 6?)
1Ø Ø2           BPL SAME (yes, do not change)
49 FF           EOR #$FF (complement output)
85 C4    SAME  *STA PREOUT (save output)
6Ø              RTS       (normal output)
```

coding for SIMRND (version 2)

```
A5 C1           LDA RND    (load core seed)
C5 C5           CMP REPNUM (= repeat # ?)
FØ 1B           BEQ REPEAT (yes, repeat exit)
85 C6           STA PRERND (save seed)
18              CLC
69 2B           ADC #$2B  (add constant)
85 C1           STA RND  (store next seed)
18              CLC
65 C2           ADC ADDEND (add the addend)
C9 CØ           CMP #$CØ  (bit 7 = bit 6?)
1Ø Ø2           BPL SAME  (yes, no change)
49 FF           EOR #$FF (complement output)
E6 C3    SAME   INC COUNT (new sequence?)
DØ Ø4           BNE NORMEX (no, normal exit)
E6 C2           INC ADDEND (new addend)
C6 C5           DEC REPNUM (decrement repeat#)
85 C4    NORMEX STA PREOUT (save output)
6Ø              RTS       (normal output)
A5 C6    REPEAT LDA PRERND (load previous seed)
85 C5           STA REPNUM (store in repeat #)
A5 C4           LDA PREOUT (previous output)
6Ø              RTS   (repeat   "     "   )
```

GENIE

AN IMMODEST PROPOSAL

BY DAVID CHAPMAN
34 The Fenway
Boston, MA 02215

At least ten years ago, Marshall McCluhan declared the printed media dead; print, he said, would be replaced by electronic information forms. The printed media is still with us—your perusal of this note is indication enough of that—but more and more voices join McCluhan's in predicting an end to print. There has been much talk lately of electronic networks available on small computers, and GENIE (An Immodest Proposal) is one of the more interesting hypotheses on this topic to cross our desk. David Chapman covers many points ranging from copyrights to government intervention to the possibility of a renewed form of Athenian Democracy via information networks. Can a video screen, though, replace curling up with the Sunday Paper and a hot cup of coffee on a rainy winter day? —SMR

Introduction

A lot of the ideas in this article are largely speculative, although I have generally used the present tense in describing GENIE. Nevertheless, none of the things I have mentioned (with a few possible exceptions, like animated holographics) are technically impossible in 1978. All that's needed is a little development to make the technology reliable and inexpensive. How long might this take?

One of the most important components of the system, the pocket computer, will probably be available very soon. In fact, at the time I am writing this, several pocket computers for dedicated applications like teletype emulation and navigation are available; I expect that general purpose units will be available by the time you read this or soon thereafter.

The main development problem is in building the network—a sort of bootstrap operation insofar as it won't be much use to the first few subscribers and demand will probably depend on usefulness. My guess is that if there is sufficient interest (shown by articles of this sort and the readers' response to them) there will be a GENIE-like system available by 1982 or thereabouts. This system probably won't have some of the fancy extras I discuss, like an EFTS or voice input; while these are technically feasible, they are difficult enough to program so they probably will be delayed until 1985 or thereabouts. If it catches on as I think it will, there will probably be a GENIE in every pocket by 1990 or so; the complete transition to computer screens will take longer, but my guess is that by 1995 or the millenium, paper will be of interest to historians only.

Overview

As anyone who has written for publication knows, paper is bulky, has low information density, and forces extraneous structure on the writer. It requires long seek cycles because it uses sequential access; cannot be easily indexed; and is largely, like some PROMs, once-only writable. Because of this, the writer spends much time wrestling with the medium's mechanics. Likewise, the researcher wastes time reading bibliographies and catalogs and devises complex systems involving file folders, index cards, notebooks and pigeon holes.

As recently as Shakespeare's time, one might know almost "everything" there was to know; but since the Scientific Revolution, the growth rate of human knowledge has increased to the point where it is estimated the amount of information in the world doubles every few years. This is a problem for even the most specialized researchers.

GLOSSARY

Words italicized in the text are defined in this Glossary.

Document. any "lump" of information; a single, unified work. A document could be a "book," an article, a piece of music, a picture, a game program, or almost anything else that has informational content. (See the section on Local Software.)

Network. this word is used in three apparently different senses in this article: in "computer network," "telephone network," and "network-structured text." All three uses have the same underlying idea: a set of "points" ("nodes" in graph theory jargon—but not to be confused with the "central nodes" introduced later) connected together in an arbitrary (in the sense of not neatly ordered) way. In a computer network, the points are computers; in a telephone network, they are telephones; and in a network-structured document they are "chunks" of information. (The word "net" is sometimes used as an abbreviated form of "network.")

View. this word has rather arbitrarily been redefined herein to a general purpose word for making a document do its thing. Viewing a textual document would be reading it; viewing a musical document would be listening to it; viewing a game would consist of playing it; and so on.

Publish. is used rather specially in this article to mean "making a document public so other users of GENIE can view it."

Unit. "unit" is used to mean any computer that is a part of the GENIE network that has an "active" function. Other computers might have "passive" functions such as switching messages between units.

Multiprocessing. multiprocessing systems are those that can apparently do more than one thing at once by splitting their time between several tasks. The difference between multiprocessing and **time-sharing** (q.v.) is that multiprocessing generally allows one user to do several things at once, while time-sharing allows several users each to do one thing at a time.

"Bought" documents. are those for which the user has paid the "buying fee" and thus has unlimited free access to. A good analogy is that of buying a book rather than borrowing it from a library.

EFTS. Electronic Funds Transfer System, an extension of credit cards that eliminates paper and metal money.

Front-end program. a front-end program is one which deals with the user, sending him prompts, getting his input, checking to see that it makes sense, and formatting data output to him.

Time-sharing. is a means whereby several users can share the use of a computer, each apparently having his own complete unit. Compare **multiprocessing**, above.

Many people in the computer field believe the solution is an **information storage and retrieval system**, a sort of computerized card catalog conveniently located in the Library of Congress or some similar place. I believe this is the wrong approach. In the last few years, several trends in the development of small computers have emerged that will make possible a different approach. Among these are increased use of graphics and text-handling software; increased concern about and better techniques to ensure privacy of information; increasing power and decreasing cost of the computers; and the democratization of computer power; that is, the widespread usage of computers by non-programmers.

All of these trends and others are likely to contribute to the production, in the 1980s, of a nation-wide computer *network* capable of supporting, among other facilities, an **electronic library system** for programmers and non-programmers alike, designed to display not only books, but also magazines, pictures, music, movies, thus making paper and other such primitive media obsolete. This article is a proposal for such a system. I call my version GENIE or the Grand Electronic Network for Information creation and Exploration.

GENIE is radically different from most other actual or proposed computer networks. The "classic" computer network—ARPANet, for example—is composed of a small number of large installations tied together to allow electronic mail sending, program exchange, and one installation's computer to do the work of another's when the latter gets ill.

Recently, two home-computer networks have been proposed: PCNET and CIENET. PCNET is very similar to ARPANET, so far as I can see, except it is based on micros instead of maxis. CIENET is more similar to GENIE, because it has non-computer related storage functions.[1]

The closest thing to GENIE is Theodore Nelson's Xanadu system. (Xanadu is a registered trademark of the Nelson Organization, Inc.) Like GENIE, Xanadu is an electronic library system, and the two have many similar features. However, they have slightly different design ideologies, which has led to some major differences in details. There is no room to even summarize these differences here; instead, I urge you to read Nelson's fascinating book *Dream Machines*[2], which gives at least a glimpse of Xanadu.

The major difference between GENIE and the others I've mentioned (with the exception of Xanadu) is that the emphasis is not on the computers themselves, but rather on what the computers do (namely act as display controllers for an electronic library system). GENIE gives the user instant access to any sort of information available on the net. Not only does this include a wide variety of media (text, graphics, animation, music, etc.) but it also lets the user *interact* with the information: the user can control "what happens next." Standard GENIE hardware uses extremely high-resolution graphics, voice and written input, and powerful multi-processor architecture; GENIE software allows the user to *view*, create, and *publish* documents.

Hardware

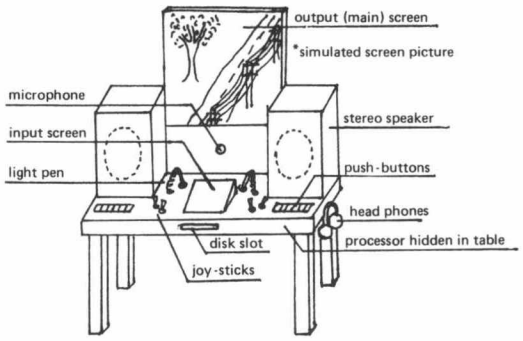
The basic hardware component of the GENIE system is the **pocket computer**. I have written about the pocket computer elsewhere[3], and a similar idea was independently proposed by Richard A. Ahern[4], so I will give only a brief description here. The pocket computer is a box about the same size and shape as a pocket calculator, differing in appearance mainly in that a lightpen is used to draw on the flat screen which covers the entire front surface of the box;

inside is a sophisticated computer equal in power to a present day midi. (fig. 1.) I estimate that by 1982 the pocket computer will cost as little as a sophisticated scientific calculator does today.

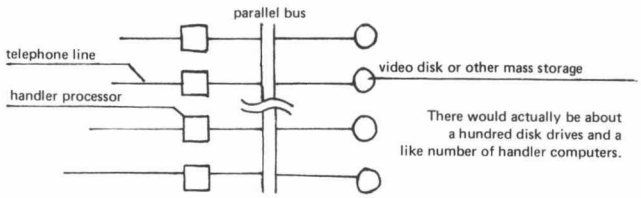
The next most important *unit* in the GENIE system is the home computer. This is a much more elaborate system, with such features as *multiprocessing*, voice input, musical output, a large (four foot square) flat screen graphics display, and perhaps holographic output. Its main function, however, is to act as a mass storage handler for all the family's pocket computers. I imagine that many people are capable of utilizing literally billions of bits of storage, especially when one considers the amount of data required to generate animated graphics and other *"bought" documents.* There is no current technology that can fit that kind of information into the approximately cubic inch that might be available in a pocket computer; indeed, there is none, except video disk, that can store that much data at all. Presumably, video disk-based mass storage devices will be available in time for GENIE; presumably, I say, because they have been promised so many times and are still not available. In any case, we will assume that some form of information storage device capable of dealing with billion-bit documents and requiring too much space to fit into a pocket-sized box will be available by 1981 or so. Then your pocket computer will be linked, by radio or other means, to your home computer, which will serve as an intermediary between it and the mass storage device: fetching, sending, receiving and storing on command. (fig. 2.) Of course, it may well be that charge-coupled or bubble memories will be sufficiently miniaturized to fit in a pocket computer, in which case our assumption is false and this rather messy linkage scheme will prove unnecessary. Nevertheless, the home computer will still be useful, if only because of its greatly superior screen area.

One of the most important decisions in designing GENIE will be whether or not to include **central nodes** for published information-storage. These central nodes, located in every town, would each contain a copy of every public document available on the GENIE system. The computer of anyone wishing to view such a document would get it for him from the nearest central node automatically. As each such node would store as much or more information than is currently in the Library of Congress, all on-line, they would consist essentially of huge mass storage units connected to telephone lines. (fig.1.) The alternative to this would be to store one or several copies of each public document on private units chosen automatically at random in each community without the owner's knowledge and in such a way that they could not be modified by them. Again, the computer of someone wanting to view a document would get it from the nearest unit that had a copy. This approach is far less efficient, as it would require phenomenally complex directory and scheduling schemes, and would waste part of the processor time and storage space of some or all individuals. However, it has the advantage, possibly crucial at some future time, that the inherent decentralization would make it much more difficult for the government or some other group to modify or destroy public information, thus helping prevent *1984*-like re-writing of history or other informational abuses. Most previous proposals for information networks have taken the existence of central repositories for granted and neglected the other possibility. Ideally, both systems might operate at the same time, with the central units being used for normal retrieval

1A. Pocket Computer



1B. Home Computer*



1C. Central Node

Figure 1. Unit-types of the GENIE Network.

and the users' computers that have public documents stored on them periodically checking to make sure that their versions match those of the nearest few central nodes (in a manner invisible to the owners, of course) and raising Cain if there is a discrepancy.

Hardware: Linking Mechanisms

The impossibility of direct physical connections between pocket computers makes the networking mechanism and not the computers themselves the main hardware problem of the GENIE system. The obvious alternative to a physical networking medium is direct radio connection. Unfortunately, this is impossible. Quite apart from miniaturization problems and power requirements, there is the question of channels.[5] I imagine there will be several hundred thousand pocket computers operating by the late 1980s and a large percentage of them in use at once. This many connections would require many thousands of high-bandwidth, long-range channels, and, even allowing for repeaters to decrease the range problem, there simply aren't that many channels available.

The solution to this problem is to use two or more levels or networking media. Nonphysical media connect pocket computers to **pickups** or simple transceivers which connect to the physical medium (fig.2.) Every home computer has its own indoor pickups so that you can use your pocket unit while wandering around the house; and in addition, there should

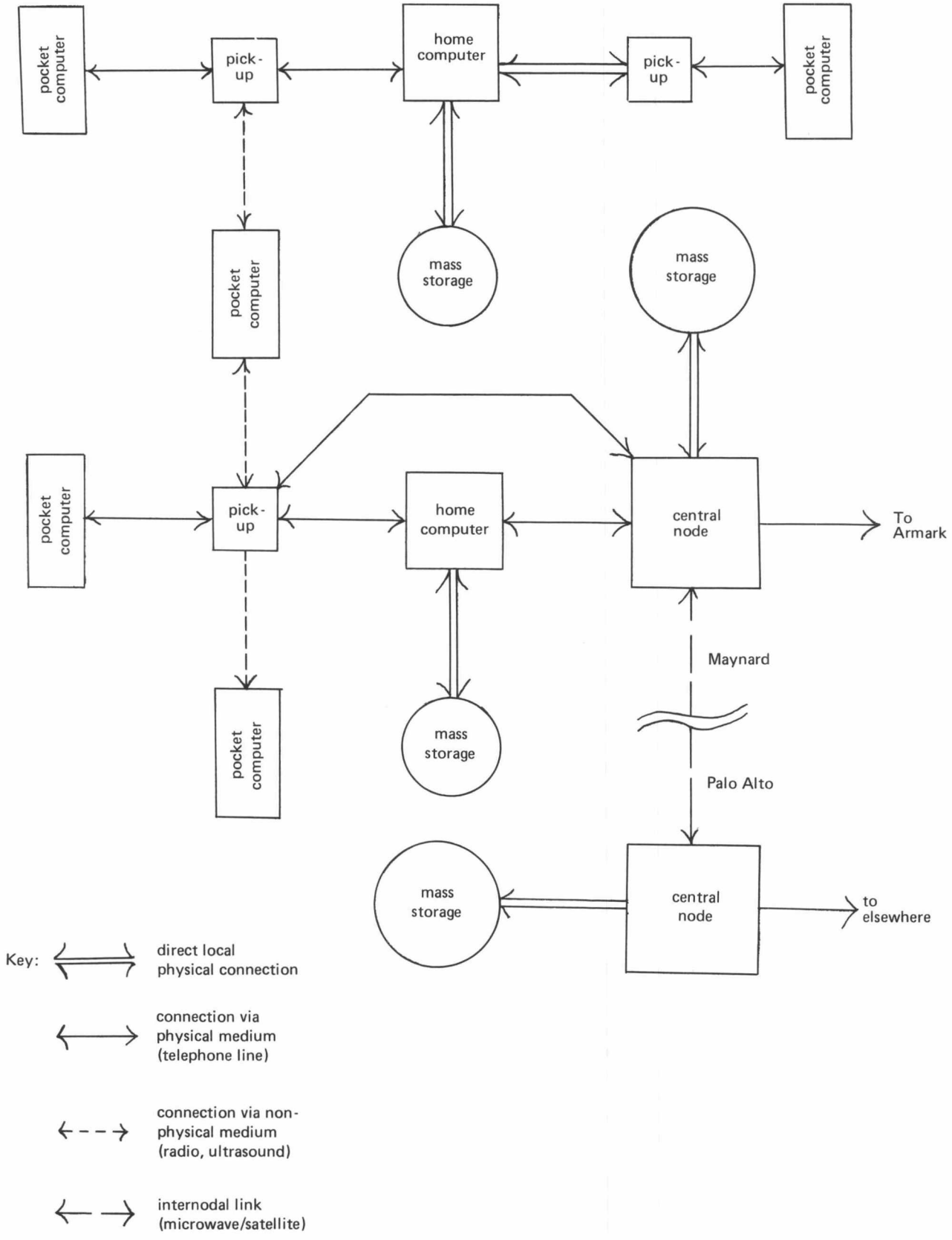


Figure 2. A very small portion of the GENIE Network at some specific microsecond.

be enough pickups, operated by local shopkeepers and costing a small amount, paid automatically to the owner via the *EFTS*, to use, that you could always link into the network. Of course, you'll probably never be able to use GENIE while on top of Mount Everest without special equipment, but the general feasibility of this scheme is demonstrated by a pilot project of the telephone system to become operational next year in Washington, DC, involving micro-processor controlled pocket telephones radio-linked to the telephone network and usable anywhere within the city.[6]

In fact, the telephone system is probably the inevitable choice for the physical medium: it already exists, is already used extensively for computer-computer and computer-terminal connections, and extends to every part of the country. However, it does have some technical and political problems. The technical problems are high cost and low bandwidth. Current telephone lines can transfer enough information to run a ten c.p.s. teletype, but could not begin to handle the megahertz transfer rates needed to send animated high resolution graphics in real time. Both these problems will soon be killed with one stone: optical fibers as substitutes for wires in telephone cables. Optical fibers, which are already being used in some cities, have a practically unlimited bandwidth. The political problem is that the telephone system is monopolized. I will discuss the issue of monopolization later in the section on Applications and Implications.

There are two possibilities for the non-physical medium: radio and ultra-sound. Radio is actually somewhat problematic technically: can a multichannel radio of sufficient power and bandwidth be miniaturized, with its power supply, into a few cubic inches? While, on the other hand, ultrasound has a shorter range and thus requires fewer channels, radio's longer range would make fewer pickups necessary. Ideally, one might use either both radio and ultrasound in one unit, or a radio transceiver capable of producing a weak signal when a pickup is nearby, or a strong one if the pickup is further away. The former would probably be preferable, as it would free all the radio channels for long-range use.

If the GENIE system has central nodes for public-document storage, they will probably be linked by microwave and/or satellite transmissions, thereby forming a third level of interconnection.

Local Software

It is the software features of GENIE that are most important, and it is in the software that it differs most from previously proposed systems. The primary function of GENIE is to handle **text**, although it also permits the creation and viewing of other types of documents – music or animated graphics, for example. GENIE is designed to deal with some unusual types of text for which new words are needed:

Straightext is what most computer text-editors handle. It is a linear sequence of punctuation symbols and letters, often upper-case only.

Extext is a datatype that is used to some extent in most current publications, particularly periodicals, which are less subject to formal format constraints than books. Extext is largely linear like straightext, but differs in that a number of structuring mechanisms may be present: subtitles and section headings; colophons and copyright notices; indices and tables of contents; introductions, prologues, and forewords; epilogues and afterwords; chapters and paragraphs; the division of magazine and newspaper pages into columns; advertising embedded in straightext; abstracts and the partial plot summaries found on the back of paperbacks and the inside front covers of hardbacks; and many more. Extext is not used nearly as much as it could and should be, and it is the lack of extext-handling features that makes most computer text-editors next to useless for writing English. Straightext is fine for computers, but not well enough structured for humans. It could well be argued, however, that the entire "structured programming" movement is an attempt to add extext devices to computer languages.

Hypertext (a term stolen from Theodore Nelson) is non-linear and can really only be implemented on a computer, though such non-linear constructions as footnotes approximate it. Although Nelson lists and at least partly explains many hypermedia, including discrete hypertext, collateration, Strechtext (TM The Nelson Organization) hypergrams, hypercomics, and Th3, two basic forms underlie most of these: plex-or-network-structured text, and collateration.[2]

The concept of **plex-structured text** seems to be largely original to Nelson (although he claims that his thought is based on a much earlier article[7]); it may be the most important innovation in printing technology since Gutenberg. Essentially the idea is that the linear structure imposed by straightext, and, to a lesser extent, by extext, is arbitrary, unnecessary, and often harmful. The linearity of most writing is due to the apparent linearity of speech[8], of which writing was originally a direct transcription. Later, the hierarchical or tree organization of chapters, sections, and so on was imposed on the language to give it structure; the tree form was used because it can be mapped directly onto a linear list. Unfortunately, until the development of interactive computer based storage systems, it was not possible to make the transition to the more free plex representation. (fig. 3.)

The other type of hypertext that is important is **collateration**. This seems to have been developed independently by Nelson and others, notably Alan Kay's Dynabook project at the Xerox Palo Alto Research Center and Doug Englebart's NLS project at the Stanford Research Institute. The basic idea is that the computer screen is divided up into several text "windows" in which there appear the documents to be collaterated. Text can be copied between windows, different parts or verisons of one work viewed simultaneously, and, in Nelson's version, links generated between them. Text within the windows can be edited and made to scroll, preferably smoothly, and not jumpily as with present video displays. (fig. 4.)

While most people might use the GENIE mainly to read, watch, or listen to documents that other people have written, almost everyone occasionally writes something. Remember that we're eliminating paper—and that includes shopping lists, order forms, tickets, envelopes, and personal letters. For this reason, GENIE must have easily understandably extext and hypertext editors. These must be based on the use of a light pen and a "zappable" menu of editing operations, rather than a command language.[9]

GENIE differs from all text-handling systems I know of in its text storage technique. A document actually consists of a plex of files, each made up of three sections: a program, a set of data and a group of links. Each "chunk" of text in a plex-structured text or between links in a collateration is stored as a separate file; but this is invisible to the user. In

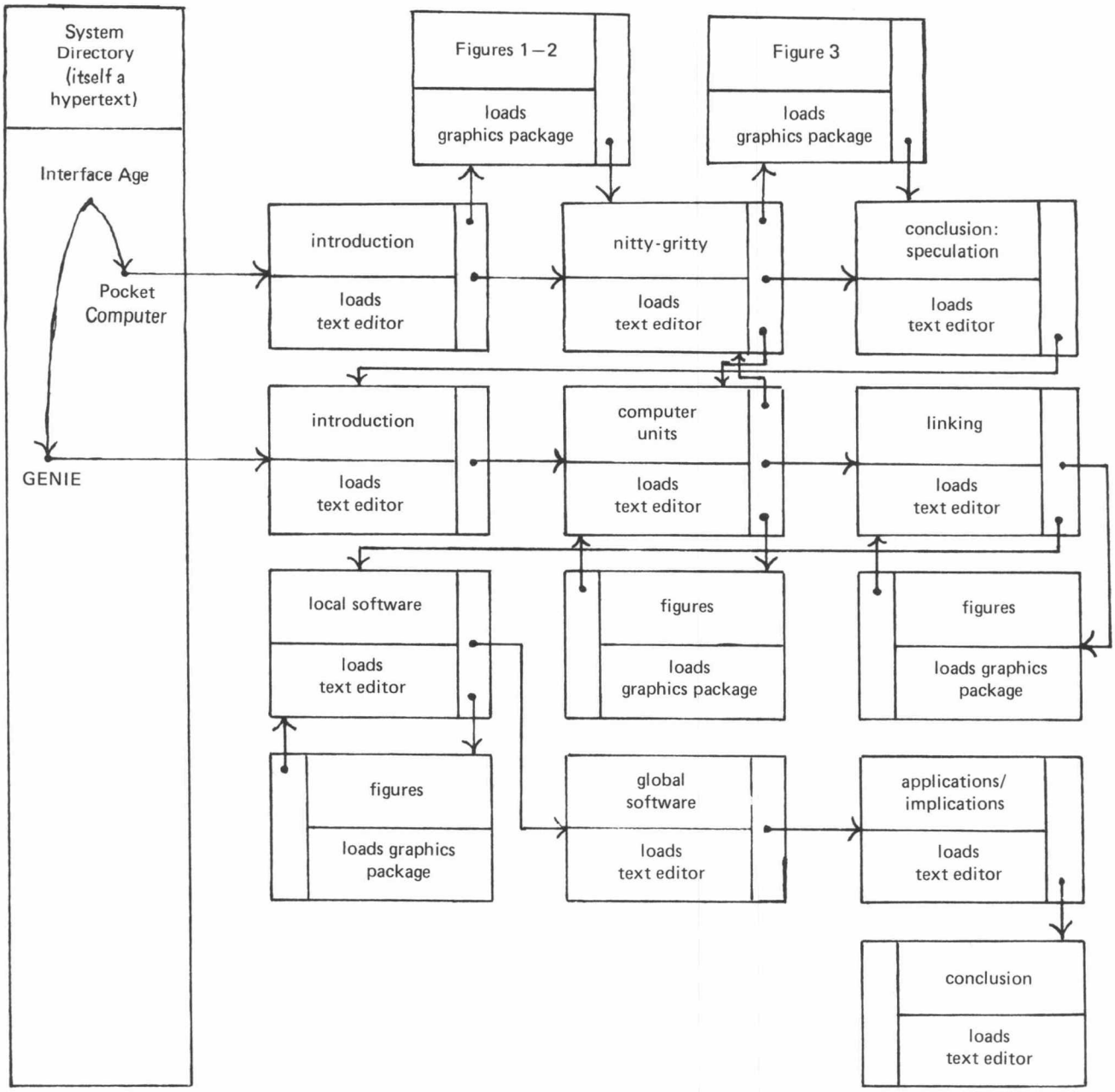


Figure 3. A very incomplete hypertext for this article.

fact, most files do not even appear in the system directory; only the first file in a document does. Viewing a document actually consists of running the program section of the first file in the chain; using the links, this will automatically jump to the second section, and so on. Each program is normally generated automatically by the editor used in writing the chunk, and consists of a simple loader that pulls a display program (which may in fact be the editor itself) into core and has it display the data section, which contains the actual text of the document. The program could, however, be user-written, in which case it would function just like a program on a current computer system; this allows the user to expand the capabilities of his sytem. The "links" section consists of a list of all the files to which the user could branch.

This storage technique has several advantages. First, the user need not keep track of which documents are in which formats or which display programs should be used with them. Second, parts of a document can be in one format while others are in another, so that one could intersperse text with graphics (illustrations), have musical backgrounds to textual documents, or mix one type of text (say Strechtext) with another (say extest.) This approach may make it possible to apply LISP-like list processing techniques to hypermedia. Since a document is itself only an entry in the system directory with a pointer to the first file in a sequence, it is very easy for one document to include part or all of another without redundantly storing the contained data; copying is merely a matter of rearranging links.

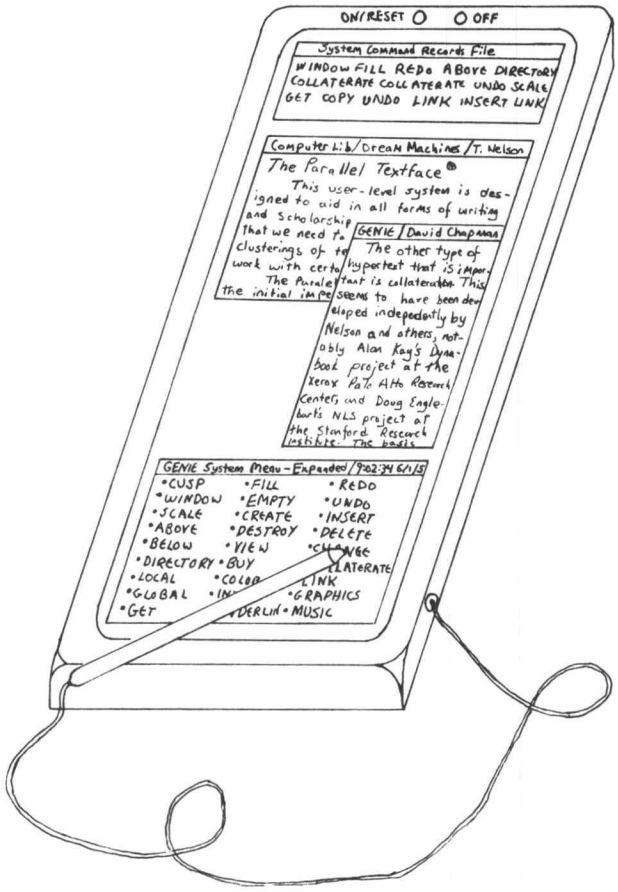


Figure 4. Collateration on a pocket computer.

Graphics

One important feature of GENIE is its ability to handle graphics, and particularly animated graphics. It is said that a picture is worth a thousand words, and certainly it is often easier to understand something if you can "see" it. The combination of animated graphics with text and perhaps sound will probably be one of the most-used media of the GENIE system.

The pocket computer is far from ideal from graphics, due to its small screen size and probable low resolution. Most graphics will probably be done on home computers or perhaps even special artists' computers. With a four-foot square screen, or, better yet, true 3D in color, remarkable things might be done; including ultimately the production of images indistinguishable from reality.

The main problem with contemporary graphics systems, however, is not the hardware but the software. Even the best systems currently available permit only the quality of good TV animation. This is because the systems force the user to try to build a picture out of lines, or, at best, polynomial surfaces; and as the world is composed of neither lines nor polynomial surfaces, extensive experimentation and "fudging" are required to produce even the simplest scene; producing a reasonable picture of a human being is virtually impossible.

What I suggest is a system that would let one "build" a scene out of parts. For example, to create a picture of a cat, you would first write a program to generate fur that would take as input such factors as length, color, fineness, and position; then a program describing the articulation of the bones and how the muscles interact with them to produce movement; then you would apply the fur program to the body program, filling in minor details like the eyes "by hand." The overall cat program would take as input position in a scene, color, size, and motion (perhaps also sound – this would require a "meow" program) and could thus be used as a small building block in a large scene. Of course, these graphic programs could be stored as documents, and public programs would be available for such common things as people, buildings, trees, sky with clouds, mirrors, and cars. This would free the user from reinventing the wheel every time a cat is needed for a picture or animated movie.

One of the crucial features that all of the *front-end programs* of the GENIE operating system must have is **reactivity**. It has been pointed out that in current interactive systems the computer spends much of its time waiting for the user to type something. This is the basis of *time-sharing*: the computer excutes one job while others are waiting for input. However, low-cost personal computers have made time-sharing obsolete. Therefore, it has been suggested[10] that systems be devised to spend the time otherwise wasted by trying to anticipate what the user is going to say. Such systems are termed "reactive" and are undoubtedly the wave of the future.

You will have noticed that although this section is titled "Local Software" I have not mentioned many of the programs you might expect to find on a computer: compilers, interpreters, debuggers, and the like. While such programs should certainly be available (and I differ on this point from Ted Nelson, who would make it difficult or impossible for users to access these features) as systems programs (that is, portions of the operating system supplied free with each unit) or as public documents, most users will probably never want to use them. This is because GENIE is to be aimed at **people** of all sorts, most of whom can't and don't want to program. However, programmer-oriented programs should be available, because programmers are people too, and should be allowed to do their own thing.

Global Software

So far, we have considered only local software; that is, programs concerned with what goes on inside one unit. But GENIE is a network; and much of the system's software will inevitably deal with communication between units. The vast number of documents that will become available when the GENIE system supplants paper, film, records, and the like will make three things necessary: huge mass storage devices, some method of limiting the number of public documents, and a way of finding the document that you want. The first requirement was dealt with in the hardware section of this article; we'll now look at the other two.

Ideally, anyone could make any document they had created public so that others could view it. This is the way many computer networks and timesharing systems work now; but this freedom is impossible with GENIE because it generates too much garbage. As all public information must be stored redundantly around the country, a lot of public rubbish could be very expensive. Another problem is that the hypertextual nature of many of GENIE's documents would tend to "lead" users into rubbish, so that one might "get a potlock-7" and be "fooled into [someone's] dopey collateration of Edgar Cayce with St. Thomas Aquinas."[11]

One solution to this problem would be to establish a national censorship bureau that would examine submitted documents to see if they are worthy of being made public. This is both impractical, due to the thousands of employees such an agency would require, and undesirable, because it would almost certainly increase rather than decrease publication delays, and because censorship bureaus tend to become politically active and oppressive.

A far better way to limit public information is to make use of the existing "censorship" structure of the publishing companies who must decide which documents to publish on the basis of usefulness (i.e. salability) in order to make a profit. What I suggest is that a fairly large fee be charged to make a document public; this is reasonable insofar as the information must be stored permanently on central nodes or other users' systems. This process of making something public could be called **publication**; the cost of publication will presumably have to be determined by trial and error, but should be high enough to strongly discourage private publication; the cost of publication will presumably have to be determined by trial and error, but should be high enough to strongly discourage private publication; presumably the actual fee would depend on the length, in bits or some other measure, of the document. The publication process normally works as follows: the author writes his document on his GENIE unit using a text-editor, and then sends it, possibly in several versions, to one or more publishers. If they think it salable, they pay the publishing fee (analogous to the cost of printing a book or periodical) and then take a cut of the royalties from viewing or purchasing the document.

Which brings us to the royalties system. It is obvious that users must pay authors and their publishers as an incentive for publication. When a user wants to view a document, the computer informs him of the viewing and buying cost per page (or line, or bit, or second). If the user only wants to view it once or twice, he or she can elect to pay the "viewing" cost each time; but if it will be used often, the user can pay the higher "buying" cost, and a copy of the document is transferred to the mass storage device for unlimited free-of-charge use. This copy, however, retains the original authorship and "copyright" and cannot be re-sold. The buying and viewing costs are, of course, set by the publisher, and could vary within the document. Once a document is published, it cannot be deleted or modified, even by the original publisher, because that would allow *1984*-like re-writing of history. A document could, however, be added to (with a clear indication given by the system of the date of addition) or be superseded by a new, additional version (with retention of the old one).

GENIE will, as I stated earlier, certainly make the life of authors much easier. Take, as an example of what can be done, this article. I am not an efficient writer, and these may be worst-case statistics, but they are illustrative. I had the basic ideas for GENIE worked out in about January, 1977. The system had assumed its present form, more or less, by June of that year. I started writing this article in January of 1978; it was completed in August. I went through four major drafts, consuming over a hundred sheets of paper and innumerable snacks of various sorts; this labor took about a hundred hours. With the publication delay on this sort of article running about six months, I hope to see it in print by early 1979. This means that the ideas you are now reading are fully two years out of date. Using GENIE, I should have been able to write a preliminary version in January, 1977, and supersede it with a final version in June; both of these could have been published in a matter of milliseconds on the GENIE system. Due to lack of time and publication delay the above schedule was quite impossible. However, I estimate that GENIE could at least halve and perhaps quarter the amount of time needed in writing by eliminating mechanical problems of error correction, paste-up, Xeroxing, and the necessity of complete re-typing to create a new draft.

Similarly, it is quite conceivable that another article saying much the same thing as this one has already appeared in one of the several dozen computer-related periodicals to which I do not have access. Using GENIE, I could instantly find any article from any periodical, or simply view the table of contents to see if anytthing with a name like "Home Computer Networking in the 1980s" has appeared recently.

The creation and viewing of public documents is not the only intercomputer transaction that can take place: "mail" sending and private document requests are others. Most current computer networks and timesharing systems allow users to send "mail" to each other, and this has proved a most useful facility. "Mail"on GENIE could be written, but it could also be sound or video, provided the sender and recipient have the necessary equipment: microphone, speakers, and video camera. Thus, "mail" includes telephone conversations and computer conferencing.[1 2] The other main type of inter-unit communication is the **request for information**. Suppose you are a manufacturer of subminiaturized hyperatomic flustulators; you have written up a brochure telling how your flustulators can cut mid-range transduction in egg beaters, but you don't want to pay the high price of publishing it. As a new company, you haven't got much of a mailing list yet. What to do? Simply tag the document as "requestable." Then when potential customers read your advertising, they can have their GENIE unit send a request for the brochure to your unit, which sends a copy of the document back to theirs. The information-request system has another application: you can specify that the document is requestable only by a restricted set of persons. Thus, semi-secret documents could be "published" without making them public.

Security

Some of the information sent between units or stored on them will undoubtedly be private or sensitive, and thus require **security**. This is, of course, doubly true if GENIE is to include an EFTS, as seems inevitable. Three types of security are available on GENIE: logins, unit identifications, and data scrambling.

Logins (a word formed by crunching "log"and "in" together) are standard on most current timesharing systems. In the context of the GENIE system, this would consist of requiring the user to give a password each time he started a session with his unit. This would help prevent the use of stolen units or unauthorized personnel gaining access to sensitive files.

Unit identification is a sort of login for units. It is a specialized form of document request in which the "document" requested is actually a serial number unique to each unit, stored on a special ROM and also engraved on the case. The unit cannot be reprogrammed or rebuilt to return anything other than the ROMmed serial number. It should, however, be possible for the user to suppress sending of the identification altogether, to preserve anonymity. The unit I.D. system

should prevent a number of informational crimes. For example, without it, and if the complex home-computer/pocket computer mass storage scheme explained earlier is adopted, it might be possible to pretend that one's pocket computer was someone else's and thus gain access to another's home computer's documents.

Data scrambling is another security device used in many current systems. It has two applications: scrambling of transmission, and scrambling of stored data. Scrambling of stored documents has two main uses: it makes it impossible for some one who figures out your login password to read your stored documents, and it makes it possible to lend your unit to a friend without giving access to all your files. Two forms of data scrambling are available on GENIE: low and high security.

Both techniques work on the same basis: a **key** – a number or an alphanumeric string that is memorized by the user and kept secret – is used to encode the document, which is then sent or stored. When it is received or retrieved, it is decoded using another key. The essential difference is that the low security technique can be used to decode a document in real time, and uses a short (five letter, say) key, and can be broken; whereas, the high security technique does not operate in real time (at least with current equipment), uses a hundred-digit key, and cannot be broken. The high-security technique (known as one-way trap-door function public-key encryption) has a number of other advantages. First, your encoding key (which is different from your decoding key) can be made public without endangering the security of the system! What this means is that anyone can get your encoding key (via a document request) and encode a message to you – but then cannot read it! Secondly, it is possible to attach a "signature" to any message which can be verified as being uniquely yours, because it is based on your secret decoding key. This has far-reaching implications. Eventually, perhaps using special hardware and an improved encoding algorithm, it may be possible to use public-key encryption in real time, in which case all data might be automatically encoded with this system; but for the time being, only very important documents will be worth the effort.

Applications and Implications

Any national information network will undoubtedly have an enormous impact on our nation and society. Many pages could be written about this topic, but due to time limitations and the uncertainty of any such predictions, I will comment on just a few of the possible problems caused by and effects of, such a system.

All information-oriented industries will probably convert to the GENIE system. Newspapers and magazines will exist simply as repositories of articles in no particular order (but probably hypertext-linked), updated continuously rather than periodically. TV will loose its sequential nature, currently dictated by the mechanics of sending and receiving programs. Users will be able to choose any program that has ever been "broadcast" (that is, published) for viewing; this will probably eliminate serials. It may also make for higher-quality, more sophisticated programming, since more than one thing can be broadcast at a time, programs will no longer have to be aimed at the lowest common denominator. Similarly, it may become worth publishers' while to publish a document of which only a few hundred copies are expected to sell.

Advertising will probably work in two different ways, depending on the type of ads involved. Informative ads, like those in this magazine, will exist as public documents with a zero (or even negative!) viewing fee that list manufacturers' names and what they make in a hypertext web, so that if one was looking for, say a floppy disc unit, you could tell the document so and it would give you all the makers of floppy drives. You could then zap their names to get a complete catalog (likewise organized as an interactive hypertext). On the other hand, persuasive ads like those found in, say, *Time* magazine or on TV, will be inserted into other documents for a fee by the publisher, as with current practice. Actually, it would be nice if this sort of advertising could be eliminated altogether; but probably the best we can do is to make it possible to suppress them for a small extra charge.

I think that the boundaries between the various media will be broken down, and writing styles radically changed, by the combination of hypertext and animated graphics. In fact, probably no one now could write a very good hypertext (try it if you don't believe it!); totally new conventions will have to be developed. While we are at it some serious thought should be given to scrapping English in favor of Loglan [13] or the like, or at the very least regularizing English spelling and eliminating some of the silliest syntactic peculiarities. In any case, I expect – and hope – to see increased informality, unconventionality, and humor in writing.

Whoever manufactures and maintains the hardware and software of GENIE, and particularly of the central nodes, if any, will have a tremendous amount of power, and an enormous temptation to misuse it. Thus they must either be above suspicion or easily monitored by any knowledgeable person. One bad choice for the controller of GENIE is the federal government, because it tends to become bureaucratically inefficient and resistant to change, is difficult to monitor, and hardly, in these post-Watergate days, above suspicion. They'll probably get into the act by regulating and red-taping the system anyway. It would be better if the system were thrown open to free enterprise; this raises problems of its own, however. If (as seems most likely) several companies produce their own incompatible systems, chaos will reign; it will be as if each TV station had its own encoding scheme that required special receivers, or as if each section of the country had its own, incompatible telephone system, only many times worse. If, on the other hand, a single company were to maintain the net, there would be the problem of power-abuse again. Ideally, several companies might decide on a common standard; but, judging by the example of the current computer industry, this seems highly unlikely. A reasonable compromise might be for each company to supply software that would allow one to convert and transfer documents from one system to another, and a reciprocity agreement whereunder any document published on one system would automatically be published on all the others at no extra cost.

One political application of a national information network might be the creation of a **direct Athenian-style democracy** in which anyone who wanted to could vote for any piece of legislation. The people could appoint executive officers for fast decisions, or not, as they chose. Such a system might work like this: suppose I come up with a new piece of legislation I think should be passed. I then write it up as a document and have it published, or, if that is beyond my means, collect enough "signatures" on a "petition" to have it published free

Continued on pg. 30

PASCAL BIBLIOGRAPHY

BY MIKE GABRIELSON
Box 2692
Stanford, California 94305

Alagic, Suad and Arbib, Michael A, The Design of Well-Structured and Correct Programs, Springer-Verlag, 1978.

Alpert, Stephen R, "Pascal: A Structurally Strong Language", Byte, 3, 8, August 1978, p 78.

Apt, K R and de Bakker, J W, "Semantics and Proof Theory of Pascal Procedures", Automata, Languages and Programming, Springer-Verlag, 1977.

Ammann, Urs, "On Code Generation in a Pascal Compiler", Software - Practice and Experience, 7, 3, June-July 1977, p 391.

Bates, D and Cailliau, R, "Experience with Pascal Compilers on Mini-Computers", ACM SIGPLAN Notices, 12, 11, November 1977, p 10.

Biedl, Albrecht, "An Extension of Programming Languages For Numerical Computation in Science and Engineering With Special Reference to Pascal", ACM SIGPLAN Notices, 12, 4, April 1977, p 31.

Bowles, Kenneth L, Microcomputer Problem Solving Using Pascal, Springer-Verlag, 1977.

Bowles, Kenneth L, "Pascal versus COBOL: Where Pascal Gets Down to Business", Byte, 3, 8, August 1978, p 122.

Bowles, Kenneth L, "Status of the UCSD Pascal Project", Dr Dobb's, 3, 3, (23), March 1978, p 16.

Brinch Hansen, Per, The Architecture of Concurrent Programs, Prentice-Hall, 1977.

Brinch Hansen, Per, "A Comparison of Two Synchronizing Concepts", Acta Informatica, 1, 3, 1972, p 190.

Brinch Hansen, Per, "Concurrent Programming Concepts", Assoc Comp Mach Computing Surveys, 5, 4, December 1973, p 223.

Brinch Hansen, Per, "Disk Scheduling at Compile Time", Software - Practice and Experience, 6, 2, April-June 1976, p 201.

Brinch Hansen, Per, "Experience with Modular Concurrent Programming", IEEE Trans Software Engineering, March 1977, p 156.

Brinch Hansen, Per, "The Solo Operating System: A Concurrent Pascal Program", Software - Practice and Experience, 6, 2, April-June 1976, p 141.

Brinch Hansen, Per, "The Solo Operating System: Job Interface", Software - Practice and Experience, 6, 2, April-June 1976, p 151.

Brinch Hansen, Per, "The Solo Operating System: Processes, Monitors, and Classes", Software - Practice and Experience, 6, 2, April-June 1976, p 165.

Brinch Hansen, Per, Operating System Principles, Prentice-Hall, 1973.

Brinch Hansen, Per, "The Programming Language Concurrent Pascal", IEEE Transactions on Software Engineering, 1, 2, June 1975, p 199.

Brinch Hansen, Per, "Structured Multiprogramming", Comm ACM, 15, 7, July 1972, p 574.

Brinch Hansen, Per, "Universal Types in Concurrent Pascal", Information Processing Letters, 3, 6, July 1975, p 165.

Bron, C and DeVries, W, "A Pascal Compiler for PDP 11 Minicomputers", Software - Practice and Experience, 6, 1, January-March 1976, p 109.

Chung, Kin-Man and Yuen, Herbert, "A 'Tiny' Pascal Compiler, Part 1: The P-Code Interpreter", Byte, 3, 9, September 1978, p 58.

Chung, Kin-Man and Yuen, Herbert, "A 'Tiny' Pascal Compiler, Part 2: The P-Compiler", Byte, 3, 10, October 1978, p 34.

Chung, Kin-Man and Yuen, Herbert, "A 'Tiny' Pascal Compiler, Part 3: P-Code to 8080 Conversion", Byte, 3, 11, November 1978, p 182.

Condict, Michael N, "The Pascal Dynamic Array Controversy and a Method for Enforcing Global Assertions", ACM SIGPLAN Notices, 12, 11, November 1977, p 23.

Conradi, Reidar, "Further Critical Comments on Pascal, Particularly as a Systems Programming Language", ACM SIGPLAN Notices, 11, 11, November 1976, p 8.

Conway, Richard, Gries, David and Zimmerman, E Carl, A Primer on Pascal, Winthrop, 1976.

Desjardins, Pierre, "A Pascal Compiler for the Xerox Sigma 6", SIGPLAN Notices, 8, 6, June 1973, p 34.

Edwards, Roy, "Is Pascal a Logical Subset of ALGOL 68 or Not?", ACM SIGPLAN Notices, 12, 6, June 1977, p 184.

Feiereisen, Lucien, "Implementation of Pascal on the PDP 11/45", Proceedings of the DECUS, 1, 1, September 1974, p 259.

Fischer, Charles N and LeBlanc, Richard J, "Efficient Implementation and Optimization of Run-Time Checking in Pascal", ACM SIGPLAN Notices, 12, 3, March 1977, p 19.

Forsyth, Charles H and Howard, Randall J, "Compilation and Pascal on the New Microprocessors", Byte, 3, 8, August 1978, p 50.

Friesland, G, Grosse-Lindemann, C O, Lorenz, P W, Nagel, H H and Stirl, P J, "A Pascal Compiler Bootstrapped on a DECSystem-10", Lecture Notes in Computer Science, 7, Springer-Verlag, 1974, p 101.

Grogono, Peter, Programming in Pascal, Addison-Wesley, 1978.

Gross-Lindemann, C O and Nagel, H H, "Postlude to a Pascal-Compiler Bootstrap on a DECSystem-10", Software - Practice and Experience, 6, 1, January-March 1976, p 29.

Habermann, A N, "Critical Comments on the Programming Language Pascal", Acta Informatica, 3, 1973, p 47.

Hartmann, Alfred C, "A Concurrent Pascal Compiler for Minicomputers", Lecture Notes in Computer Science, 50, Springer-Verlag, 1977.

Heimbigner, Dennis, "Writing Device Drivers in Concurrent Pascal", ACM Operating Systems Review, 12, 4, October 1978, p 16.

Helmers, Carl, "Is Pascal the Next BASIC?", Byte, 2, 12, December 1977, p 6.

Hoare, C A R, "Monitors: An Operating System Structuring Concept", Comm ACM, 17, 10, October 1974, p 549.

Hoare, C A R and Wirth, Niklaus, "An Axiomatic Definition of the Programming Language Pascal", Acta Informatica, 2, 1973, p 335.

Hueras, Jon and Ledgard, Henry, "An Automatic Formatting Program for Pascal", ACM SIGPLAN Notices, 12, 7, July 1977, p 82.

Iglewski, M, Madey, J and Matwin, S, "A Contribution to an Improvement of Pascal", ACM SIGPLAN Notices, 13, 1, January 1978, p 48.

Jensen, Kathleen, and Wirth, Niklaus, Pascal User Manual and Report, second edition, Springer-Verlag, 1975.

Kittlitz, Edward N, "Another Proposal for Variable Size Arrays in Pascal", ACM SIGPLAN Notices, 12, 1, January 1977, p 82.

Kittlitz, Edward N, "Block Statements and Synonyms for Pascal", ACM SIGPLAN Notices, 11, 10, October 1976, p 32.

Knobe, B and Yuval, G, "Some Steps Towards Better Pascal", Journal of Computer Languages, 1976.

Lampson, B W, "An Operating System for a Single-User Machine", Lecture Notes in Computer Science, 16, Springer-Verlag, 1974, p 208.

LeBlanc, Richard J, "Extensions to Pascal for Separate Compilation", ACM SIGPLAN Notices, 13, 9, September 1978, p 30.

Lecarme, Olivier, "An Experience in Structured Programming and Transferability", ACM SIGPLAN Notices, 8, 9, 1973.

Lecarme, Olivier, "Structured Programming, Programming Teaching, and the Language Pascal", ACM SIGCSE Bulletin, 6, 1, Febr ary 1974, ACM SIGPLAN Notices, 9, 7, July 1974, p 15.

Lecarme, Olivier and Desjardins, Pierre, "More Comments on the Programming Language Pascal", Acta Informatica, 4, 1975, p 231.

Lecarme, Olivier and Desjardins, Pierre, "Reply to a Paper by A N Habermann on the Programming Language Pascal", SIGPLAN Notices, 9, October 1974, p 21.

Ledgard, Henry, Singer, Andrew and Hueras, Jon, "A Basis for Executing Pascal Programmers", ACM SIGPLAN Notices, 12, 7, July 1977, p 101.

MacLennan, B J, "A Note on Dynamic Arrays in Pascal", ACM SIGPLAN Notices, 10, 9, September 1975, p 39.

Matwin, S and Missala, M, "A Simple, Machine Independent Tool for Obtaining Rough Measures of Pascal Programs", ACM SIGPLAN Notices, 11, 8, August 1976, p 42.

Merritt, Jim, "Pascal: BEGINning to END", Creative Computing, 4, 5, September-October 1978, p 149.

Mickel, Andy, Pascal Newsletter, University of Minnesota Computer Center.

Mohilner, Patricia R, "Prettyprinting Pascal Programs", ACM SIGPLAN Notices, 13, 7, July 1978, p 34.

Mohilner, Patricia R, "Using Pascal in a FORTRAN Environment", Software - Practice and Experience, 7, 3, June-July 1977, p 357.

Mundie, David A, "In Praise of Pascal", Byte, 3, 8, August 1978, p 110.

Nutt, Gary J, "A Comparison of Pascal and Fortran as Introductory Programming Languages", ACM SIGPLAN Notices, 13, 2, February 1978, p 57.

Peterson, James L, "On the Formatting of Pascal Programs", ACM SIGPLAN Notices, 12, 12, December 1977, p 83.

Pokrovsky, Sergei, "Formal Types and Their Application to Dynamic Arrays in Pascal", ACM SIGPLAN Notices, 11, 10, October 1976, p 36.

Russell, David L and Sue, Jeffrey Y, "Implementation of a Pascal Compiler for the IBM 360", Software - Practice and Experience, 6, 3, July-September 1976, p 371.

Schmidt, J W, "Some High Level Language Constructs for Data of Type Relation", ACM Trans Database Systems, 2, September 1977, p 247.

Schneider, G M, Weingart, S and Perlman, D, An Introduction to Programming and Problem Solving with Pascal, Wiley, 1973.

Schwartz, Allan M, "Pascal versus BASIC: An Exercise", Byte, 3, 8, August 1978, p 168.

Silberschatz, A, Kieburtz, R B and Bernstein, A J, "Extending Concurrent Pascal to Allow Dynamic Resource Management", IEEE Trans Software Engineering, May 1977, p 210.

Solntseff, N and Wood, D, "Pyramids: A Data Type for Matrix Representation in Pascal", BIT, 17, 3, 1977, p 344.

Steensgaard-Madsen, J, "More on Dynamic Arrays in Pascal", ACM SIGPLAN Notices, 11, 5, May 1976, p 63.

Stein, Herbert, "A Homebrew Pascal Compiler", Byte, 3, 8, August 1978, p 46.

Tennent, R D, "Another Look at Type Compatibility in Pascal", Software - Practice and Experience, 8, 4, July-August 1978, p 429.

Tennent, R D, "A Note on Files in Pascal", BIT, 17, 3, 1977, p 362.

Thibault, D and Mancel, P, "Implementation of a Pascal Compiler for the CII Iris 80 Computer", ACM SIGPLAN Notices, 8, 6, 1973.

Travis, Larry, Masahiro, Honda, LeBlanc, Richard and Zeigler, Stephen, "Design Rationale for TELOS, a Pascal-Based AI Language", SIGPLAN Notices, 12, 8, August 1977, p 67.

Venema, Ted and des Rivieras, Jim, "Euclid and Pascal", ACM SIGPLAN Notices, 13, 3, March 1978, p 57.

Webster, C A G, Introduction to Pascal, Heyden, 1976.

Welsh, J and Quinn, C, "A Pascal Compiler for ICL 1900 Series Computers", Software - Practice and Experience, 2, 1, January-March 1972, p 73.

Welsh, J, Sneeringer, W J and Hoare, C A R, "Ambiguities and Insecurities in Pascal", Sofware - Practice and Experience, 7, 6, November-December 1977, p 685.

Wirth, Niklaus, Algorithms + Data Structures = Programs, Prentice-Hall, 1976.

Wirth, Niklaus, "An Assessment of the Programming Language Pascal", IEEE Transactions on Software Engineering, 1, 2, June 1975, p 192.

Wirth, Niklaus, "Comment on a Note on Dynamic Arrays in Pascal", ACM SIGPLAN Notices, 11, 1, January 1976, p 37.

Wirth, Niklaus, "The Design of a Pascal Compiler", Software - Practice and Experience, 1, 4, October-December 1971, p 309.

Wirth, Niklaus, "The Programming Language Pascal", Acta Informatica, 1, 1, 1971, p 35.

Wirth, Niklaus, Systematic Programming - An Introduction, Prentice-Hall, 1973.

Yuen, Herbert and Chung, Kin-Man, "A Proposed Pascal Compiler", Byte, 3, 8, August 1978, p 117.

Continued from pg. 28

automatically as a public service. Part of the document would be a program that would accept and collate suggestions for amendments. Eventually, if there seem to be enough people in favor of the bill, I write up a final version. Part of this document would be a vote collecting program; anyone interested could vote either "aye" or "nay" before a certain given date. If there is a majority of "ayes" and a certain minimum number of votes cast, the bill would become law, and a copy would be sent to all users.

Another application of GENIE that seems very probable is the creation of an *electronic fund transfer system* (EFTS, in financial jargon.) An extension of checks and credit cards, this would eliminate the paper in money: your cash would consist of a number in your bank's computer. When you buy something, the cost would be subtracted from your balance and added to the seller's; sales and even income tax could be skimmed off automatically. Among other advantages, this would make stealing money very difficult, would save a lot of green ink, and would make it possible to do all your shopping in the privacy of your own house. Obviously, such a system would require very good security.

References

1. Fylstra, Dave. "Personal Computing Network." *BYTE*, September, 1977, pp. 59-61; and Wilber, Mike. "CIE Net: A Design for a Network of Community Information Exchanges. " published as a three part series in *BYTE*: part 1, February 1978, pp. 14, 138-145; part 2, March 1978, pp. 152-164; part 3, April 1978, pp. 168-176.

2. Nelson, Theodore. *Dream Machines.* Publisher not apparent (bound with author's *Computer Lib*), 1974.

3. Chapman, David. "The Pocket Computer." *Interface Age*, December 1977, pp. 72-74.

4. Ahern, Richard A. "The Pocket Computer is (Almost) Here." *Creative Computing*, September-October 1977, pp. 88-89.

5. Steinwedel, Jeff. "Personal Computers in a Distributed Communications Network." *BYTE*, February 1978, pp. 80-82, 94-101.

6. "Business: Thinking Small." *Time*, February 20, 1978, p. 53.

7. Bush, Vannevar. "As We May Think." *Atlantic Monthly*, July 1945; cited in Nelson, *Dream Machines*.

8. Chapman, David. "Linearity, Hierarchy, and ComPlexity [sic]." To be written sometime in the next couple of years.

9. A good example of what not to do is found in Kenny, Donald. "Man/Machine Interface." *Creative Computing*, January-February 1977, pp. 60-61.

10. Winograd, Terry. "The Reactive Engine Paper." *Creative Computing*, November-December 1976, pp. 58-59.

11. Nelson, Theodore. "A Dream For Irving Snerd." *Creative Computing*, May-June 1977, pp. 79-81.

12. *Creative Computing*, September-October 1977, has a set of articles on computer conferencing: Turoff, Murray. "Computer Conferencing: Present and Future." pp. 54-57; Vallee, Jacques, Johansen, Robert, and Spangler, Kathleen. "The Computer Conference: An Altered State of Communication?" pp. 58-60; and North, Steve. "Computer Conferencing: A Personal View." p. 60.

13. Brown, James Cooke. *Loglan 1: A Logical Language,* and other publications. Available from The Loglan Institute Inc., 2261 Soledad Rancho Road, San Diego, California 92109.

Features for TDL Editor, Assembler and Text Output Processor

BY R.J. LONG
19 Cedarleigh Road
Kenmore, Qld. 4069
AUSTRALIA

The TDL editor can be used to produce source code for the assembler, or the 'raw text' to be processed by the Text Output Processor (a word processor). The programs as provided by TDL (now Xitan) expect that the assembler and text output processor (TOP) read their source code from a 'controlled reader,' i.e., one which can supply characters as demanded by the programs. Unfortunately the ordinary cassette recorder which is used for storage does not satisfy this requirement.

My solution to this problem was to make the assembler and TOP read from memory.

As a further convenience, the routines allow the appropriate pair of programs to reside together in memory—the editor and assembler, or the editor and TOP. This means that if an error is detected in either the assembly process or in the word processing, corrections can be made without reloading the text from tape.

The editor is loaded into low memory with its text buffer above it. The assembler or TOP is loaded above the editor buffer, and it is from this buffer that the programs read the text. The editor denotes the end of its buffer space with a 'FF' byte. If this byte is detected by the memory read routine then the current state of either the assembler or TOP is saved, and the user is requested to 'LOAD NEXT SEGMENT.' The assembler or TOP processing is then continued from the control routine by pressing 'C.' This means very large programs can be assembled or large amounts of text can be word-processed. I have assembled the additional routines to reside in the RAM at F800H above the Zapple monitor on the TDL System Monitor Board. This routine becomes a 'high level monitor' which selects which program is to be used—the editor, the assembler or word processor. The routine is entered by a JUMP to BEGIN or by pressing the 'K' key, which the Zapple Monitor treats as a user-defined monitor command, and in the specific location (F81E) a JUMP to BEGIN is inserted.

Once this 'control' routine is entered it will reply with the prompt']'. Several functions can then be performed by pressing one of the following keys (no carriage return is needed):

E—calls and initializes the editor.
R—restarts the editor (without re-initialization).
A—calls the assembler.
W—calls the word processor.
C—continues either the assembler or TOP after end-of-file break.
X—returns to the Zapple monitor.

OTHER FEATURES. Often when loading files from cassette with the editor, the first few characters at the front of the file are misread. One of the patches to the editor ensures that a valid front-of-file is encountered(and is indicated by ringing the bell) before reading the file itself. Also, when any of the programs is exited, control returns to the new control routine rather than the Zapple monitor.

EXAMPLE. Suppose you wish to assemble a program, the source code of which takes up more space than your available memory.

First load the editor, assembler and the control routine. I made an absolute hex file containing these three files (using the Zapple 'W' command with only one end-of-file using 'E' after the three) so that one 'R' (read) command need be given. Then press 'K' to get to the control routine. Press 'E' to initialize the editor, (setting the BUFFER END to 4FFF) and use it to load the first section of the source code with the 'A' (append) command. Exit from the editor and you will be back to the control routine. Now press 'A' and the assembler will ask for the PASS. Enter '1'. Because the first part of the source code for the assembly does not include the '.END' pseudo op, the assembler will 'crash' into the 'FF' end-of-file, so the prompt 'LOAD NEXT SEGMENT' will appear on the console. At this stage you will be back in the editor automatically, so simply empty the buffer using the editor's 'K' command then load the second part of the source code. Exit from the editor back to the control routine and press 'C'. This will continue pass one of the assembly.

When the assembler encounters the .END directive it will ask again which pass to execute. Reply '0' to get back to the control routine then press 'R' to restart the editor. Reload part one of the source code, get back to the assembler and execute pass 2 or 3 to generate a listing or punch the object code. Keep repeating this process to perform any other passes required.

If you are doing pass 3, the object code will be recorded onto cassette. When the assembler detects the 'FF' end-of-file, stop the cassette and take it out of the recorder without rewinding the tape. Then load part 2 of the source code into the editor buffer and replace the cassette to receive the object code. When the assembly is continued with 'C' a leader will be punched at the front of the next section of object code.

All of this sounds tedious and time consuming—and it is! At least it enables long assemblies to be performed without a disc system.

Operation of the word processor is simplified because only one pass through the raw text is needed for its operation.

IMPLEMENTATION. The routines and patches are specifically for the TDL software, and several routines within the Zapple monitor are utilized. The routine has been assembled to reside at F81EH, which is the destination of the monitor's user-defined 'K' command. The editor is assumed to be loaded at 100H, the assembler at 5000H and the text output processor at 8000H. These values are set as constants at the beginning of the program code, so can be readily changed.

PATCHES. Several patches are needed within the editor, assembler and word processor. They refer specifically to the following versions:

Editor version 1.0

Assembler version 2.2

Text Output Processor version 1.0

Editor patches (when loaded at 100H):

Replace:	with:	purpose:
028A: CD D7 01	CD 36 F8	Jumps to CLRED to reset FLAG.
06C2: 21 B2 00	CD 33 F8	Jumps to APPEND to find front of file.
0989: CD 1E 01	C3 1E F8	Jump to control routine instead of monitor trap.

Assembler patches (loaded at 5000H):

Replace:	with:	purpose:
5009: C3 06 F0	C3 F8 39	Jump to memory read routine instead of normal reader input.
5376: CD EC 6A	CD 3C F8	Calls RESETA.
547C: CD 1E 50	C3 1E F8	Jumps to control routine instead of monitor trap when 'PASS=' is answered with '0'.
61F2: CC 1E 50	CA 1E F8	As above, when ^C is used.

Text output processor (loaded at 8000H):

Replace:	with:	purpose:
8009: C3 06 F0	C3 39 F8	Jump to memory read routine.
83AB: CD 0604	CD 3F F8	Calls RESETW.
877B: CC 1B 80	CA 1E F8	Exit to control routine when ^C.
8AAE: CD 1B 80	C3 1E F8	As above, normal termination.

ANOTHER MODIFICATION TO THE EDITOR. The editor has a command 'W' which writes to the punch a line at a time from the front of the editor buffer, then deletes that line. The 'W' command requires that the space bar be pressed before the line is actually punched. This is inconvenient if the punch device is the cassette and many lines are to be punched out. The call to the console input routine which waits for this character can be removed so that multiple W's can be executed together without continuously pressing the space bar. The address to change is 0983. Substitute CD 49 02 with 00 00 00 (NOP's).

Don't forget when recording files on tape with the editor to put some nulls on the front—otherwise the routine which looks for the front of the file will never find it! I use NNNNUNQ$$, which also rings the bell when it has finished the operation.

The source code and object code are available from the author on TDL format 1200 baud cassette for $10 (Australian dollars, please).

```
TDL Z80 RELOCATING ASSEMBLER VERSION 2.2
Patches for resident TDL Editor and Assembler or Word Processor
R.J. Long  June 1978.
                        ;
                        ;
                        ; (C) Copyright 1978 by:
                        ;
                        ; R.J Long,
                        ; 19 Cedarleigh Rd.,
                        ; Kenmore, Qld. 4069
                        ; Australia.
                        ;
                        ; Routines and patches to allow TDL Zapple Editor
                        ; (Ver 1.0) and TDL Z80 Relocating/Linking
                        ; Assembler (Rev. 2.2),
                        ; or Editor and Text Output Processor (Rev. 1.0) to
                        ; reside together in memory. Also adds additional
                        ; features.
                        ;
                        ; These routines assume that the Editor is loaded
                        ; at 0100H, the Assembler at 5000H and the
                        ; TOP at 8000H.
                        ;
                        ; This 'control' routine calls either the Assembler
                        ; or the Editor, or the Word Processor or Editor,
                        ; depending on which pair is in memory. This routine
                        ; is accessed from the Zapple Monitor by the
                        ; command 'K'.
                        ;
                        ; This routine announces itself by the prompt ']'.
                        ;
                        ; The following commands can be used (no carriage
                        ; return is required):
                        ;
                        ;       E - calls and initialises the Editor.
                        ;       R - restarts the Editor.
                        ;       A - calls the Assembler.
                        ;       W - calls the Word Processor.
                        ;       C - continues either the Assembler or
                        ;           TOP after E-O-F break.
                        ;       ^X - returns to the Zapple monitor.
                        ;
                        ; Any other characters will be echoed then ignored.
                        ;
                        ; We use Zapple Monitor routines as follows:
                        ;
F512                    MCRLF   == 0F512H       ; Output CR/LF.
F736                    MTI     == 0F736H       ; Get keyboard character in
                                                ;   upper case.
F48A                    MCO     == 0F48AH       ; Output a character from reg-C.

F474                    MRIFF   == 0F474H       ; Get  byte from tape.
F452                    MTOM    == 0F452H       ; Prints Text Message on console

F5A3                    MLEAD   == 0F5A3H       ; Punch leader on tape
                        ;
0100                    EDITOR  == 0100H        ; Start of Editor.
0103                    RSTED   == 0103H        ; Editor restart.
```

```
5000                ASSMB   == 5000H        ; Start of Assembler.
8000                TOP     == 8000H        ; Start of Word Processor.
F053                ZSTART  == 0F053H       ; Zapple Monitor (soft start).
                    ;
00B2                OLDED   == 00B2H        ; Existing Editor value.
01D7                EDPRMT  == 01D7H        ; Editor routine for prompt '*'.

0C01                EDBUFF  == 0C01H        ; Start of Editor buffer area.
6AEC                PASS    == ASSMB+1AECH  ; Assembler routine to
                                            ;  print 'PASS='
8006                CONCHR  == TOP+6H       ; TOP routine to get
                                            ;   console char.
                    ;
                    .PABS   ; Punch absolute code.
F81E                .LOC    0F81EH  ; Destination of Zapple 'K' command.
                    ;
F81E    C3 F842             JMP     BEGIN   ; Go around vectors.
                    ;
                    ; Jump vectors TO outside.
                    ;
F821    C3 F512     CRLF:   JMP     MCRLF   ; Jump to appropriate Zapple
F824    C3 F736     TI:     JMP     MTI     ;   routines.
F827    C3 F48A     CO:     JMP     MCO
F82A    C3 F474     RIFF:   JMP     MRIFF
F82D    C3 F452     TOM:    JMP     MTOM
F830    C3 F5A3     LEAD:   JMP     MLEAD
                    ;
                    ; Jump vectors FROM outside
                    ;
F833    C3 F874             JMP     APPEND  ; From Editor
F836    C3 F89B             JMP     CLRED
                    ;
F839    C3 F8A4             JMP     MEMRD   ; From Assembler
F83C    C3 F904             JMP     RESETA
F83F    C3 F90B             JMP     RESETW  ; From TOP
                    ;
F842    CD F821     BEGIN:  CALL    CRLF
F845    0E5D                MVI     C,']'
F847    CD F827             CALL    CO      ; Output prompt
F84A    CD F824             CALL    TI      ; Get keyboard character
F84D    FE45                CPI     'E'     ; Editor?
F84F    CA 0100             JZ      EDITOR  ; Yes, jump to editor
F852    FE52                CPI     'R'     ; Restart Editor?
F854    CA F86E             JZ      RESTRT  ; Yes
F857    FE41                CPI     'A'     ; Assembler?
F859    CA 5000             JZ      ASSMB   ; Yes
F85C    FE57                CPI     'W'     ; Word processor
F85E    CA 8000             JZ      TOP     ; Yes
F861    FE43                CPI     'C'
F863    CA F8D7             JZ      CONT    ; Continue after E-O-F break
F866    FE18                CPI     18H     ; ^X?
F868    CA F053             JZ      ZSTART  ; Return to Monitor
F86B    C3 F842             JMP     BEGIN   ; Invalid - try again
                    ;
F86E    CD F821     RESTRT: CALL    CRLF
F871    C3 0103             JMP     RSTED   ; Restart Editor
                    ;
                    ; The next routine is called from a patch in the Editor
                    ; when an 'append' command is given.  Its purpose is to
                    ; find the front of the file (to avoid spurious
                    ; characters) but only for the first append in a string
                    ; of appends.
                    ;
F874    21 00B2     APPEND: LXI     H,OLDED ; Existing Editor instruction
F877    3A F927             LDA     FLAG    ; Don't look for front of file
F87A    FE00                CPI     0       ;   if flag is set
F87C    C0                  RNZ
F87D    3E01                MVI     A,1     ; Set flag
F87F    32 F927             STA     FLAG
F882    CD F886             CALL    FRONT   ; Look for front of file
F885    C9                  RET
                    ;
F886    C5          FRONT:  PUSH    B       ; Finds front of file
F887    D5                  PUSH    D
F888    1600                MVI     D,00    ; NULL
F88A    0620        ..L0:   MVI     B,20H   ; Number of nulls to count
F88C    CD F82A     ..L1:   CALL    RIFF    ; Gets byte from tape and
                                            ;   compares with reg-D (NULL)
F88F    20F9                JRNZ    ..L0    ; Not NULL, try again
F891    10F9                DJNZ    ..L1    ; Loop until get enough NULLs
F893    0E07                MVI     C,07    ; Got leader so ring bell
F895    CD F827             CALL    CO      ;   and return
F898    D1                  POP     D
F899    C1                  POP     B
F89A    C9                  RET
                    ;
                    ; This routine is called from a patch in the Editor
                    ; to clear FLAG when the Editor prompt '*' is printed.
                    ;
F89B    3E00        CLRED:  MVI     A,0
F89D    32 F927             STA     FLAG
F8A0    CD 01D7             CALL    EDPRMT  ; Prompt routine in Editor
F8A3    C9                  RET
                    ;
                    ; This routine is used to read from memory, rather than
                    ; from cassette. It is jumped to from a modified jump
                    ; vector at the front of the assembler or TOP.
                    ;
F8A4    E5          MEMRD:  PUSH    H       ; Save H & L
F8A5    2A F928             LHLD    PTR     ; Get pointer to memory
F8A8    7E                  MOV     A,M     ; Get what is pointed to
F8A9    FEFF                CPI     0FFH    ; E-O-F?
F8AB    2807                JRZ     SAVE
F8AD    23                  INX     H       ; Increment pointer
F8AE    22 F928             SHLD    PTR     ; Store updated pointer
F8B1    E1                  POP     H       ; Restore H & L
F8B2    B7                  ORA     A       ; Clear carry
F8B3    C9                  RET
                    ;
F8B4    E1          SAVE:   POP     H       ; Restore H & L from MEMRD
F8B5    ED43 F92A           SBCD    SAVEBC  ; Save all registers
F8B9    ED53 F92C           SDED    SAVEDE
F8BD    22 F92E             SHLD    SAVEHL
F8C0    DD22 F930           SIXD    SAVEIX
F8C4    FD22 F932           SIYD    SAVEIY
F8C8    ED73 F934           SSPD    SAVESP
F8CC    21 F912             LXI     H, MSG  ; Print 'LOAD NEXT SEGMENT'
F8CF    0615                MVI     B, MSGL
F8D1    CD F82D             CALL    TOM     ; In Zapple Monitor
F8D4    C3 F86E             JMP     RESTRT  ; Jump to Editor
                    ;
                    ; Continues from the E-O-F break by restoring
                    ; the registers and stack pointer from save area.
                    ;
F8D7    CD F8FA     CONT:   CALL RESET      ; Reset buffer pointer
F8DA    CD F821             CALL    CRLF
```

Continued on pg. 13

A 2708 PROM Programmer for the 6800

Dear Dr. Dobbs,

Here is a simple 2708 programmer for the 6800. It operates from a single 6820 device and obtains the 26 volts or so required for programming from an inbuilt oscillator using a 555 timer.

To operate the program, first load the address of the start of memory to be copied to the 2708 in location 0000 hex, then load the end address into location 0002 hex. The program can be started from location 2000 hex. Operation is self-explanatory.

The program can be relocated in any portion of memory, as all the jumps are relative. Be careful, however, to relocate the message start addresses (at locations 20EC, 2178, 21AD and 21B9).

The PIA address must also be relocated if the original of 8100 hex is not used. The program was designed for the RT68 monitor, but will run just as well with MIKBUG or SWTBUG.

E. Insam

11, Nelson Rd.
Harrow on the Hill
Middlesex, England

```
00010                    NAM    PROMPROG
00020                *
00030                *                6800
00040                *        2708  PROM  PROGRAMMER
00050                *
00060                *   INTERFACES TO 6820 PIA
00070                *   A SIDE OF PIA CONNECTED TO 2708 DATA
00080                *   B SIDE OF PIA CONNECTED TO 2708 A0-A7
00090                *   ALSO PB0,PB1 ARE CONNECTED TO INPUTS OF
00100                *   D FLIPFLOP, OUTPUTS GO TO A8,A9 OF 2708
00110                *   FLIPFLOP IS CLOCKED FROM CA2.
00120                *   CB2 SWITCHES 26V PROGRAM PULSE (+VE GOING)
00130                *   PROGRAM RUNS 100 LOOPS OF 1MS PULSES
00140                *
00150                *
00160                        OPT    0,NOG
00170 A048                   ORG    $A048
00180 A048 2000              FDB    BEGIN     START ADDRESS
00190 0000                   ORG    $0000     RAM AREA
00200 0000 0002      SADD    RMB    2         RAM START OF COPY
00210 0002 0002      EADD    RMB    2         RAM  END OF COPY
00220 0004 0004      ADDR    RMB    4         SCRATCH AREA
00230 0008 0001      TALLY   RMB    1
00240        8100    PIAAD   EQU    $8100     A SIDE PIA
00250        8101    PIAAC   EQU    PIAAD+1
00260        8102    PIABD   EQU    $8102     B SIDE PIA
00270        8103    PIABC   EQU    PIABD+1
00280                *       MIKBUG LINKS
00290        E07E    OUTMES  EQU    $E07E
00300        E1D1    OUTCH   EQU    $E1D1
00310        E1AC    INCH    EQU    $E1AC
00320        E0C8    OUT4HX  EQU    $E0C8
00330        E0CA    OUT2HX  EQU    $E0CA
00340        E0E3    MIKBUG  EQU    $E0E3
00350                *
00360                *
00370                *
00380 2000                   ORG    $2000     PROGRAM START
00390 2000 8E A047   BEGIN   LDS    #$A047
00400 2003 7F 8101           CLR    PIAAC
00410 2006 7F 8103           CLR    PIABC
00420 2009 86 FF             LDA A  #$FF      INIT PIA'S
00430 200B B7 8100           STA A  PIAAD
00440 200E B7 8102           STA A  PIABD
00450 2011 8D 55             BSR    CLR1
00460 2013 7F 0008           CLR    TALLY
00470 2016 CE 20EC           LDX    #MSG0
```

```
00480 2019 BD E07E        JSR    OUTMESS
00490 201C BD E1AC        JSR    INCH
00500 201F 81 56          CMP A  #'V
00510 2021 27 49          BEQ    VER       VERIFY  2708
00520 2023 81 50          CMP A  #'P
00530 2025 27 0E          BEQ    PROGR     PROGRAM 2708
00540 2027 7C 0008        INC    TALLY
00550 202A 81 4C          CMP A  #'L
00560 202C 27 3E          BEQ    VER       LIST 2708
00570 202E 81 4D          CMP A  #'M
00580 2030 26 CE          BNE    BEGIN
00590 2032 7E E0E3        JMP    MIKBUG
00600 2035 DE 00   PROGR  LDX    SADD
00610 2037 DF 04          STX    ADDR
00620 2039 8D 76          BSR    OUTADD    OUTPUT FIRST
00630 203B 7D 0008        TST    TALLY     DATA & ADDRESS
00640 203E 26 11          BNE    IN1
00650 2040 CE 2178        LDX    #MSG1
00660 2043 BD E07E        JSR    OUTMESS
00670 2046 BD E1AC        JSR    INCH
00680 2049 81 50          CMP A  #'P
00690 204B 26 B3          BNE    BEGIN
00700 204D 86 64          LDA A  #100
00710 204F 97 08          STA A  TALLY     100 PROGRAM LOOPS
00720 2051 8D 17   IN1    BSR    PUL1      26V PULSE
00730 2053 8D 5C          BSR    OUTADD
00740 2055 09             DEX
00750 2056 9C 02          CPX    EADD      FINISHED?
00760 2058 26 F7          BNE    IN1
00770 205A 8D 0E          BSR    PUL1      PROG LAST LOCATION
00780 205C 86 2A          LDA A  #'*
00790 205E BD E1D1        JSR    OUTCH
00800 2061 7A 0008        DEC    TALLY
00810 2064 26 CF          BNE    PROGR
00820 2066 20 98   IN2    BRA    BEGIN
00830 2068 20 54   CLR1   BRA    CLEAR     BRANCH LINKS
00840 206A 20 70   PUL1   BRA    PULSE
00850 206C CE 21AD VER    LDX    #MSG2     VERIFY
00860 206F BD E07E        JSR    OUTMESS
00870 2072 7F 8101        CLR    PIAAC
00880 2075 7F 8100        CLR    PIAAD
00890 2078 8D 44          BSR    CLEAR
00900 207A DE 00          LDX    SADD
00910 207C 20 07          BRA    NEXT2
00920 207E DE 04   NEXT   LDX    ADDR
READD
00940 2082 27 2B          BEQ    END
00950 2084 08             INX
00960 2085 DF 04   NEXT2  STX    ADDR
00970 2087 8D 3E          BSR    OUTADR
00980 2089 B6 8100        LDA A  PIAAD
00990 208C 7D 0008        TST    TALLY
01000 208F 26 04          BNE    NEXT3     LIST FLAG TEST SKIP
01010 2091 A1 00          CMP A  0,X
01020 2093 27 E9          BEQ    NEXT      NOT RIGHT, OUTPUT
01030                *                 PROM     PROM     RAM
01040                *                 ADDR     DATA     DATA
01050 2095 97 06   NEXT3  STA A  ADDR+2
01060 2097 A6 00          LDA A  X
01070 2099 97 07          STA A  ADDR+3
01080 209B CE 21B9        LDX    #MSG3
01090 209E BD E07E        JSR    OUTMESS
01100 20A1 CE 0004        LDX    #ADDR
01110 20A4 BD E0C8        JSR    OUT4HX
01120 20A7 BD E0CA        JSR    OUT2HX
01130 20AA BD E0CA        JSR    OUT2HX
01140 20AD 20 CF          BRA    NEXT
01150 20AF 20 B5   END    BRA    IN2       BACK TO BEGIN
01160                *          OUTPUT ADDRESS & DATA TO 2708
01170 20B1 DE 04   OUTADD LDX    ADDR
01180 20B3 A6 00          LDA A  X
01190 20B5 B7 8100        STA A  PIAAD
01200 20B8 8D 0D          BSR    OUTADR
01210 20BA 08        OUT  INX
01220 20BB DF 04          STX    ADDR
01230 20BD 39             RTS
01240                *          RESET PIA'S
01250 20BE C6 34   CLEAR  LDA B  #$34
01260 20C0 F7 8101        STA B  PIAAC
01270 20C3 F7 8103        STA B  PIABC
01280 20C6 39             RTS
01290 20C7 8D F5   OUTADR BSR    CLEAR
01300 20C9 96 04          LDA A  ADDR
01310 20CB B7 8102        STA A  PIABD
01320 20CE 86 3C          LDA A  #$3C
01330 20D0 B7 8101        STA A  PIAAC      PULSE 74175
01340 20D3 F7 8101        STA B  PIAAC
01350 20D6 96 05          LDA A  ADDR+1
01360 20D8 B7 8102        STA A  PIABD
01370 20DB 39             RTS
01380 20DC 8D E0   PULSE  BSR    CLEAR
01390 20DE 86 3C          LDA A  #$3C
01400 20E0 B7 8103        STA A  PIABC      UP PROG PULSE
01410 20E3 CE 0064        LDX    #100
01420 20E6 09        DELAY DEX              4
```

Continued on pg. 45

PROGRAMMING PASTIMES AND PLEASURES

BY CHARLES WETHERELL

1979 © Charles Wetherell

"Why Can't I Remember That?"

One of the major frustrations in my life is my inability to remember to use facts and ideas I know perfectly well. When I look at programs written by my friends, I suspect the affliction may be well-nigh universal. Let me illustrate with a common programming situation.

The Table Search Problem

Let's consider a program which includes a table of values stored in consecutive memory locations—a table we need to search for particular values which may be stored there (of course, the table might also be in an array in a language like BASIC or PASCAL). Whenever this situation occurs, I immediately think of linear search—just walk straight up the table looking for the value. A bastard ALGOL fragment which does linear search looks like

```
for i :=1 to n do
   if TABLE [i] = KEY
      then (*Eureka!*) goto SUCCESS; fi;

end;
(* Abject failure if you get here.*)
```

This algorithm always works, but its cost is terrible: n/2 on average if KEY is in TABLE and guaranteed cost of n if KEY is not in TABLE. Understand we count cycles through the loop as the cost.

Binary Search Algorithm

As soon as I see how slow this is, I always remember binary search—keep dividing the table in two until the Key is trapped. Binary search does require that the table be sorted, but very often a sorted table is useful for some other part of the program as well, so it is a kind of free requirement. The following bastard ALGOL fragment does a binary search.

```
LO := 1; HI := n;
while LO <=HI do
   PTR := (LO+HI)/2;
   (*Use integer division *)
   if TABLE [PTR] =KEY
      then goto SUCCESS
      else if TABLE [PTR] < KEY
         then HI := PTR - 1;
         else LO := PTR + 1;
      fi;
   fi;
end;
(* If you get here, too bad bunky. *)
```

Binary search is a little more complicated than linear search, but it will never take more than $\log_2(n)+1$ loop cycles (round the logarithm up to the next biggest integer). Even for a table of only 100 entries, this is 8 cycles versus an average of at least 50 cycles for linear search. By the way, this is the only common algorithm I know where a three-way branch is useful; I wonder how FORTRAN ever decided it was a good idea.

Linear Interpolation Search

Now this is what I *always* forget. Linear interpolation search does even better than binary search on average, because it doesn't just blindly divide the table in two; it uses the value of the KEY to make a better guess at the KEY's location in TABLE. Of course, the table entries must be something like numbers, but it is often easy to convert names or characters to numbers, particularly on small machines. The binary search algorithm converts to linear interpolation search by replacement of the line labelled TEST with

```
TEST:     PTR := ((HI+LO)*(KEY-TABLE[PTR]))
                 /(TABLE[HI]-TABLE[LO]);
```

Make sure to do the multiplication before the division. If you try linear interpolation search by hand, you will see how it can make smart guesses. In fact, it probably works about the same way you look up words in the dictionary. Linear interpolation search can be very slow—as bad as linear search at the worst. But on average, it should do even better than binary search because the extra information improves each guess.

And Now The Moral

No doubt you thought that once I remembered linear interpolation search, my problems would be over. No, because what I really forgot to check was what other people know about searching. When I look up search in KNUTH, *The Art of Programming*, he goes through this same discussion. Binary search is better than linear search and linear interpolation search should be better yet. *But it isn't.* When simulation experiments were tried, Knuth reports, the extra arithmetic in linear interpolation cost too much. So it's really no improvement at all. One more time I learn the most important lesson —the one I really always forget: DON'T GO OFF HALF-COCKED.

A Postscript

In some special cases, linear interpolation search might still be best after all. You may have one. To find out, why don't you run some tests? Let me know about the tests you run and what you think the results show. In a later column, I will discuss any interesting letters I receive.

Letters

CAVEAT EMPTOR

Dear Doc:

I thought I should bring you up to date on my hassle with MiniMicroMart.

Unfortunately, your comments at the end of my original complaint letter, published in *Dr. Dobb's* last month, leave the impression that I would get my money back and that would be the end of the whole sorry episode. Nothing could be further from the truth.

The last I ever heard from Maury Goldberg or MiniMicro-Mart was July 14, 1978, when I received two rather nasty letters from them. Since then, they have ignored three more letters I have written. They did send me their new catalog, though.

Meanwhile, in an attempt to resolve the problem, I have written to: *Personal Computing;* Byte Publications; Mr. John Hayes, Advertising Director for Byte Publications; Mrs. Gladys Mathieu, Classified Advertising for *Popular Electronics*; Chamber of Commerce, Syracuse; Better Business Bureau, Syracuse; Attorney General, State of New York; Consumer Protection Board, State of New York. Everyone that is, except Maury Goldberg's mother, Menachem Begin and Pope John Paul II. If I can't get results any other way, I just might try those last three.

One disturbing note arises on a close reading of the new MiniMicroMart catalog. They are now selling what they describe as "very high ticket items" (HIGH $$$$) and have "recently been burned rather severely". As a result, they are requiring hefty deposits and cashier's checks on big items and payment in full on smaller purchases. In view of my experience, this amounts to asking for donations to finance their operations. Caveat.

In the NINE MONTHS since I made the mistake of dealing with MMM, my two year old daughter has learned to walk and talk, the winter snows have melted, the summer has come and gone and the winter is here again, and still no computer. As a matter of fact, I could have made another baby in the amount of time, and no doubt had a lot more fun.

Sincerely,
Robert J. Retelle

2005 Whittaker Rd.
Ypsilanti, MI 48197

NOTE: As readers of DDJ *know, we have had many and frequent complaints about Mr. Goldberg and MMM. We are again writing to him and will publish his response—or lack thereof— next month.* —*SMR*

BETTER THAN MINI-MICRO MART

Dear Doc:

Received: 78 Oct 18

In reading the many complaints listed in your "Letter to the Editor" section voiced against the Mini-Micro Mart, I can only report my complete satisfaction with a competitor, Com puter Enterprises Inc. who were very prompt in dealing with correspondence, moderately prompt in shipping the items ordered and extremely helpful with technical questions concerning some of their kits. Considering the similarity in price between the two companies, there appears to be a vast gulf between the service rendered by Mini-Micro Mart and Computer Enterprises. It is unfortunate that some computer magazines still will accept advertising from Mini-Micro Mart in view of all the complaints that have been voiced. As an alternative I recommend Computer Enterprises. Feel free to print this last comment if you desire.

Sincerely yours,
Jan E. Leestma, M.D.

Northwestern University
Chicago, Illinois

RATING SUPPLIERS FROM ABROAD

Dear Doc:

I am currently building up my third micro system in the past three years and I would like to relate some recent experiences with mail order suppliers.

Ithaca Audio. I currently run their Z-80 CPU, two 8K RAM boards and a 16/32K PROM board in my system. All were purchased as "bare" printed circuits boards. One of the 8K boards had a bug which I tried to eliminate for about 6 months before returning it to them with a money order for their maximum repair charge plus postage. After about six weeks, the board came back in working condition along with a check for the amount I had sent. They admitted the problem was a bad plated thru hole, repaired it and returned it to me at no charge. As a general comment, their boards are very well designed and produced with every attention to speed and reliability. All parts in my system are specified to 450 nanoseconds but are running at 325 nanoseconds.

Mikos. I have made several orders to this company in the past 1 ½ years. They have a very rapid turnaround time and provide excellent quality parts. The Solid State Music VB-1 video display board in my system was purchased from this company and came with excellent documentation in less than two weeks from the time I mailed the order. In one of my

orders, I had included a hex listing for PROM programming in which a mistake was made by their programmer. I returned it and in less than two weeks had it back with a refund of the postage it cost me to return it to them. A nice letter of apology was also included.

Jade. I have only recently been ordering from this company, but have been very impresed. Among specifics: the packaging they use for mailing is good. That is important to us who are stationed overseas (Germany) and whose mail is subject to a nasty beating. To date, 48 memory chips have been ordered with every one exceeding specified speed. In another order, I had been working from an old catalog and a price reduction in the item had occurred. To my surprise, I received more items from the same price. Another factor easing dealings with this firm is that they will not hold an order if adequate postage is not included, they merely request you send the required amount with the next order.

Minimicromart. Several orders were sent a couple of years ago and they will receive no more. Quality and service were about the worst that I have experienced.

S.D. Computer Products. Before they changed their name, they were fair to good. I have a serious problem now which you can see from the enclosed copy of the second letter to them on a recent purchase. I do not think they will receive more of my business.

Thank you and I look forward to more *DDJ*.

Sincerely,
Cpt. Harold F. Bower.

HQ 5th SIG CMD, DCSPI
APO NY 09056

A PAT ON THE BACK FROM POLAND

Dear *Dr. Dobb's*:

I found *Dr. Dobb's Journal* during my stay and practice at Matsushita's Wireless Laboratory in Osaka, Japan last summer, and *DDJ* became my favorite magazine on microcomputers. There are a few reasons why I like it. First one is I'm a graduate student at the Computer Science Department at Technical University of Warsaw. Second one is I've been working and learning at the Institute of Nuclear Research and I'm interested in the design of on-line and real-time systems for controlling processes based upon microcomputers. Third one is I have got an MCS-80, Intel 8080 cross-assembler and simulator both running on a PDB 11/45 developed at our institute. Fourth one is very good level of *DDJ* papers (McKeeman's programming language translation techniques, J. Starkweather's Guide to 8080 Pilot, etc.).

So please enter my subscription to *DDJ* and send me magazines by surface mail.

Thank you for your very good journal.

Sincerely,
Marek W. Michalski

15-427 Lipowa 12 M 13
Biatystok, Poland

STRUBAL+

Gentlepeople:

I am very pleased that my letters to your magazine have inspired so much interest among your readers. It is useful for the entire programming community to discuss the relative benefits and drawbacks of various languages. I would like to take this opportunity to respond to the letters of Ancelme Roichel, John S. James, and John P. Oliver from the Nov-Dec 1978 issue. The points in these letters overlap, so I will first address the common points from all three letters.

First of all, the STRUBAL language has now been refined and extended . . . it is now called STRUBAL+. STRUBAL+ is a fully extensible language. STRUBAL+ provides the programmer with the ability to define a nearly unlimited set of new keywords and constructs. These new keywords, etc. are defined in terms of already-defined keywords and/or 6800 assembly language. It is possible to redefine the intrinsic functions of STRUBAL+ and to add new capabilities to the language through function and macro libraries. I have already written a STRUBAL+ function library that includes the set-operations of PASCAL, and I am presently working on a cross-assembler for the INTEL 8086 processor done as a macro library. STRUBAL+ functions may contain local labels and variables. STRUBAL+ functions may be nested and may be recursive. The facilities for conditional compilation/assembly are provided so that functions may change form depending on external conditions. Parameter strings may be passed to functions. Thus, the user of STRUBAL+ can customize the language to his needs. Enclosed is a simple example of the definition and use of STRUBAL+ functions. Two new keywords, CHAR and ASCII, are added to STRUBAL+. The ampersand symbol (&) is used to indicate an argument or parameter string.

Second, I feel that there is a fundamental difference of opinion between myself and the users of FORTH/SAM76 over how to optimize a small language. STRUBAL+ attempts to optimize execution speed of user programs at the expense of memory usage. FORTH/SAM76 and other such languages seek to minimize memory usage at the expense of execution speed. I question whether memory optimization makes sense on modern microcomputers. I doubt that anyone today who contemplates serious program development will have less than 16K in his machine. The end user may have a smaller machine, but the programs will be developed on a machine with extensive facilities. True, execution speed is a relative thing too . . . and languages like FORTH and SAM76 achieve remarkable speeds of interpretation. Compilers like STRUBAL+ need a package of runtime support routines, interpreters take up space . . . the whole argument is largely irrelevant to the average programmer. STRUBAL+ allows areas of imbedded assembler code where maximum speed is required. STRUBAL+ provides a very rich set of intrinsic functions . . . floating-point, scientific functions, file I/O, output formatting, etc. One new capability of STRUBAL+ is the facility to dynamically define the number of decimal digits of precision in floating-point calculations. STRUBAL+ can work in precisions from 4 to 14 digits . . . trading speed for accuracy. This covers applications from scientific simulations to business. Again, STRUBAL+ provides mechanisms to optimize for speed where needed. This richness requires a basic overhead of 3.7K bytes for a STRUBAL+ program. As the old adage states, "You get what you pay for."

I'd now like to comment on each letter in turn. Mr. Roichel's comments on "pseudo-English" are valid . . . so STRUBAL+ allows the renaming (through functions) of the keywords. The latin character set is required by ASCII coding – if one uses different characters with the same encoding, STRUBAL+ will run fine. STRUBAL+ can be extended in much the same ways as FORTH/SAM76. I feel that his comments on keystrokes are misguided. It has been shown in psy-

chological studies that people remember things in chunks . . . the type of "chunk" doesn't matter. I can remember about 5 or 6 nonsense syllables . . . but I can remember quite a number of simple English words. I can remember several hex digits, but not too many binary bits. I type the word "the" (which is 3 letters) in one motion. With some experience, typing standard key sequences becomes second-nature. Besides, newer terminals can have special keys for the keywords. I'd rather do a bit more typing when I enter a program than have to type extensive documentation afterwards.

In response to Mr. James, STRUBAL+ has features to allow quite complex data structures, including mixtures of the basic integer, string, and floating-points types, to be defined. STRUBAL+ can define new structures as desired. His comments on expanding the task size (the LIFE example) highly exaggerates the case. Few people would try to write such a large program as he envisions without breaking it into subroutines and procedures – in effect making a "program-directed interpreter." Any "structured programming" effort is going to define a set of internal functions anyway. Again, we get to the speed/size argument – all language implementors will have to wrestle with this. There are times when interpretation is called for: programs that must be frequently changed, "hands-on" systems for small development efforts. However, for large programs which are to be written and then used repeatedly, compilation is a better bet. One point not yet made by any of the writers . . . if one wants to sell software, a compiler gives the author a measure of protection in that the source code is not part of the sale. (Some might not consider that an advantage.)

What is the upshot of all this discussion? There is a wealth of programming tools becoming available for microcomputerists. They each have advantages and drawbacks. The prospective user of a language must evaluate it based on his needs and interests. I do not believe that there is one "best" language, and I do not think there ever will be one. I propose STRUBAL+ as a powerful, effective tool. Anyone who writes a new language faces a large task in disseminating it. I appreciate the opportunity to describe STRUBAL+ in these pages.

Sincerely yours,
Robert D. Grappel
V.P., Software Development

Hemenway Associates
151 Tremont Street, Suite 8P
Boston, MA 02111

RESET ADAPTER

Dear *Doc*:

Ever hit the RESET button on your APPLE II right in the middle of a long program or just as you were winning a serious battle or chess game? I know about hitting reset when things are going the other way, too. Or maybe your youngster, like mine, likes to hear the "beep" when RESET is pushed. In any event, if you have an application where accidental RESET can cause problems, you may want to install the adapter shown in the figure 1. This adapter let's you put the RESET button in a less accessible place. You can mount the parts on the rear panel or in a small utility box. The restore switch, S2, returns RESET back to the keyboard.

Build the adapter as indicated in figure 1, soldering all pins together except pin three. Then connect three wires of a suitable length from the adapter to the switches. Light gauge wire such as ribbon cable works fine if you're just going to the back panel. You may need heavier gauge wire if you're going very far away from your computer. Connect the wires to the parts as follows:

- One wire to the bent-up socket-pin (S)3.
- One wire to the component header-pin 8.
- Connect these wires to the push button S1.
- Connect another wire to the component header-pin (H)3.
- Connect the other end of S2 to the same point on S1 that the pin 3 socket wire comes from.

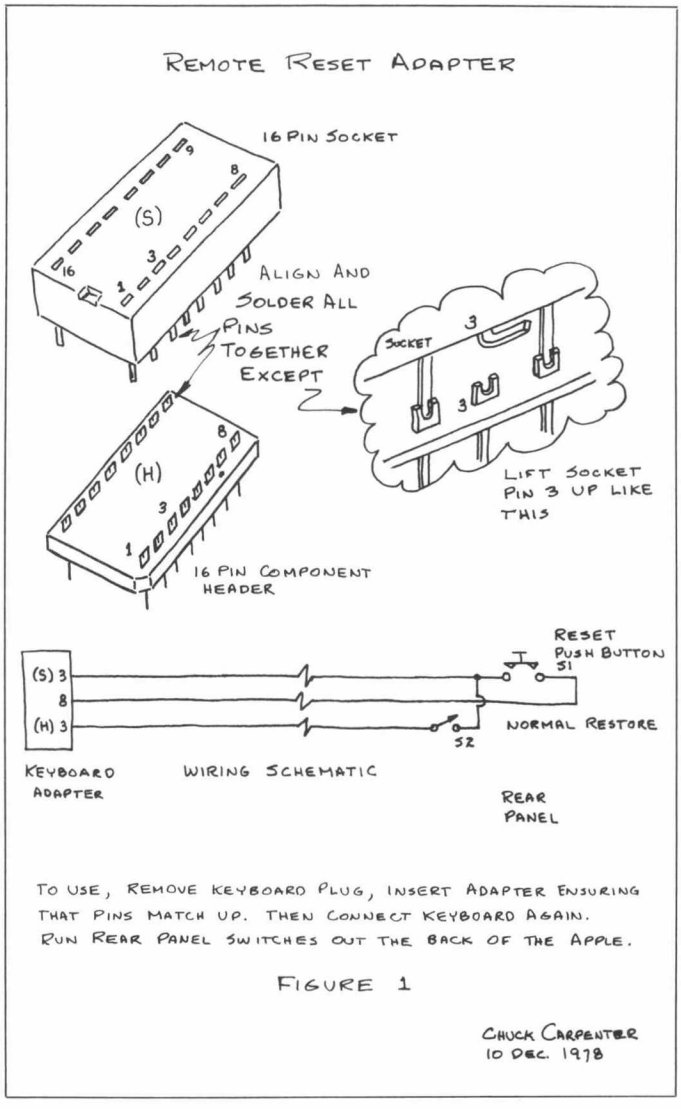
Carefully solder all the connections.

To install the adapter, first make sure the power is off. Remove the cover and unplug the keyboard connector. Ground yourself (figuratively) to the power supply case to eliminate any static problems. Now install the adapter making sure all pins are aligned and in proper sequence. Reconnect the keyboard plug and route the switches to the rear of the computer. Mount the switches however you choose. Put the cover back on and you're in business.

Now RESET has to be a deliberate action: no more accidents. You may have to take that chess loss rather than accidentally hit RESET.

Sincerely,
Chuck Carpenter

2228 Montclair Place
Carrollton, TX 75006



PAGED LISTINGS IN BASIC

Dear Editor:

Enclosed is a short routine to enable one to obtain paged listings of BASIC programs using Extended Benton Harbor (i.e. Heath) BASIC. Each page is numbered at the top. The user merely has to load the routine into high memory along with the patches from EBHB and then use the 'LIST' command in BASIC as usual. The program as written allows 56 lines of text per page and 10 spaces between pages. These values are easily alterable by the user.

Sincerely,
William W. Moss

1507 Riverview Lane
Brandenton, Florida 33505

```
BASIC LISTING PAGER - VERSION 1.4                    HASL  #04.01.00.
                                                      PAGE  1



                        * * * * * * * * * * * * * * * * *
                        * WILLIAM W. MOSS - 12/10/78  *
                        * * * * * * * * * * * * * * * * *

                        * USER DEFINED VARIABLES:

000.012                 SPCLEN  EQU     10          NUMBER OF SPACES BETWEEN PAGES
000.070                 TXTLEN  EQU     56          NUMBER OF LINES OF TEXT ON EACH PAGE
177.000                 ORIGIN  EQU     177000A     LOCATION OF THIS ROUTINE IN HIGH MEMORY ABOVE BASIC

                        * REFERENCES TO EXTENDED BENTON HARBOR BASIC VERSION 10.01.02:

101.122                 $CCJMP  EQU     101122A     SET CONTROL-C JUMP POINT
071.307                 $RITEDE EQU     071307A     WRITE REG DE AS DECIMAL INTEGER ON CONSOLE
100.011                 $CRLF   EQU     100011A     CARRIAGE RETURN / LINE FEED

                        * 'RST 7' IS A ROUTINE TO PRINT OUT THE TEXT FOLLOWING THE CALL;
                        * THE FIRST BYTE FOLLOWING THE CALL IS THE NUMBER OF CHARACTERS TO BE PRINTED.

046.043                         ORG     46043A      PATCH IN 'LIST' ROUTINE TO INITIALIZE LINE COUNTER
046.043  315 000 177            CALL    PAGESET
046.046  000                    NOP
046.047  000                    NOP

046.114                         ORG     46114A      PATCH IN 'LIST' FOR CHECKING LINE COUNT
046.114  315 047 177            CALL    NEWLINE

177.000                         ORG     ORIGIN
177.000  345            PAGESET PUSH    H           INITIALIZATION
177.001  325                    PUSH    D
177.002  315 122 101            CALL    $CCJMP      THIS REPLACES THE CODE OVERWRITTEN
177.005  236 064                DW      64236A              BY THE PATCH IN 'LIST'
177.007  041 113 177            LXI     H,COUNTER
177.012  066 002                MVI     M,2
177.014  043                    INX     H
177.015  066 001                MVI     M,1         SET PAGE COUNTER TO 1
177.017  315 025 177            CALL    PAGE
177.022  321                    POP     D
177.023  341                    POP     H
177.024  311                    RET

177.025  136            PAGE    MOV     E,M         PRINT PAGE HEADING
177.026  026 000                MVI     D,0         DE = PAGE NUMBER
177.030  377                    RST     7
177.031  006 015 120            DB      6,13,'PAGE '
177.040  315 307 071            CALL    $RITEDE     WRITE DE AS DECIMAL ON CONSOLE
177.043  315 011 100            CALL    $CRLF
177.046  311                    RET

177.047  345            NEWLINE PUSH    H           CALLED FROM 'LIST' IN LIEU OF CARRIAGE RETURN
177.050  325                    PUSH    D
177.051  365                    PUSH    PSW
177.052  041 113 177            LXI     H,COUNTER
177.055  064                    INR     M           INCREMENT LINE NUMBER COUNTER
177.056  076 070                MVI     A,TXTLEN
177.060  276                    CMP     M
177.061  322 104 177            JNC     LINE        IF PAGE NOT YET COMPLETED
177.064  036 012                MVI     E,SPCLEN
177.066  377            VERTAB  RST     7           VERTICAL TAB: SPACE DOWN 'SPCLEN' TIMES
177.067  001 012                DB      1,10
177.071  035                    DCR     E
177.072  362 066 177            JP      VERTAB
177.075  066 003                MVI     M,3         RESET LINE COUNTER
177.077  043                    INX     H
177.100  064                    INR     M           INCREMENT PAGE COUNTER
177.101  315 025 177            CALL    PAGE
177.104  315 011 100  LINE     CALL    $CRLF
177.107  361                    POP     PSW
177.110  321                    POP     D
177.111  341                    POP     H
177.112  311                    RET

177.113  000 000        COUNTER DW      0           LINE & PAGE NUMBER COUNTERS

177.115                         END     40100A      SET PROGRAM COUNTER TO START OF BASIC


STATEMENTS = 00077
FREE BYTES = 18410
NO ERRORS DETECTED.
```

REVIEW OF TINY-C OWNER'S MANUAL

I was excited to see an announcement of Tiny-c in a recent personal computing magazine. 'C' is a modern language developed at Bell Labs and used to write the popular UNIX operating system.

Since Tiny-c Associates was located in Holmdel, NJ, which also happens to be the home of Bell Labs, I took a chance and sent off a check for $40.

Two weeks later I received the Tiny-c Owner's Manual containing one of the best documented programs I have seen in years. Buying the manual means you own a copy of Tiny-c and have a serially numbered registration form you send in to be notified of updates.

The manual is contained in an attractively silk-screened, loose-leaf binder with about 350 pates of legibly offset-printed pages. The complete source listings of Tiny-c for the 8080 and PDP-11 are included and are legible.

The manual assumes no familiarity with 'C' and starts with a well-written introduction to the language.

Chapter 2 describes the language – definitely a stripped down version of 'C'. The two types of data supported are character and integers. Unfortunately, there is no summary of the language, but comparing it to 'C' as described in the book by Kernihan and Ritchie we find: Only two unary operators ('+' and '–') are supported compared to 12 in 'C'. All of the binary operators are there except for shift. There are no logical or bitwise operators. Arrays, functions and pointers are supported, but not structures or unions. There are no storage classes or initialization procedures. 'IF' and 'WHILE' are supported, but 'DO' and 'FOR' are not. 'BREAK' is supported, but 'CONTINUE' is not. 'SWITCH/CASE' is not supported, nor is there a preprocessor.

The syntax is changed slightly, statements are bracketed by square brackets instead of curly brackets and array elements are surrounded by parentheses instead of square brackets.

On the whole, Tiny-c seems to be a useful subset and the 8080 listing in appendix A shows that it will fit in a little over 4K bytes, with additional space required for the symbol table and source program.

Chapter 3 describes the program preparation system which is written in Tiny-c. It contains the functions of editor, executive and the load-store interface to cassette tape or floppy disk.

Chapter 4 has many Tiny-c program examples, including an original game called 'Piranha fish' which is complex enough to show the advantages of structured programming in a language such as Tiny-c.

Chapter 5 documents the operation of the Tiny-c interprefer. The source is interpreted directly; it is not compiled to an intermediate form. This chapter should be taken as a model of good documentation for software. It documents both the 8080 and the PDP-11 assembly language versions.

Chapter 6 tells how to customize Tiny-c for an 8080 system. This manual contains all of the code so you could type the system in by hand; however, customized systems are already available for about 6 kinds of 8080 systems on cassette tape or floppy disk. Customizing Tiny-c is not a job for a novice, but complete directions are supplied.

Chapter 7 describes installation of a PDP-11 version.

The three appendices give the interpreter codes for the 8080 and PDP-11 and the Tiny-c listing of the program preparation system.

Yours,
Ted Shapin

5110 Elsinore Ave.
Orange,CA

Note: address for Tiny-c Owner's Manual is P.O. Box 269, Holmdel, N.J. 07733.

A PORTABLE LISP INTERPRETER

Dear *Dr. Dobb's*,

Your readers may be interested in a Lawrence Livermore Lab report titled "A Portable LISP Interpreter" by L.A. Cox, Jr. and W. P.. Taylor (Report UCRL-52417. dated May31, 1978; available for $4 from National Technical Information Service, US Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161).

The report is 17 pages long and consists mainly of a listing of the interpreter, which has been implemented in PASCAL. The PASCAL program is 1056 lines long, well commented, and includes a cross reference table. To quote the report:

All the major LISP functions are here, including a comprehensive garbage-collector algorithm. The program was coded with expansion and modification in mind. Input/Output, for example, currently depends on the local PASCAL implementation. Comments in the code strive to explain not only the immediate implementation, but the global problems that drove it.

The program is well worth examining as an example of systems implementation using PASCAL, and for insight into the implementation of one of the most misunderstood of computer languages, LISP.

Sincerely,
Mike Gabrielson

Box 2692
Stanford, Calif. 94305

RUMORS

In the first quarter of 1979, Intel will be announcing the availability of the 8088, an 8-bit subset machine compatible with the 8086.

They are currently spending $1 million per month on 8086 software development.

POW! MODIFIED FOR SOLOS

Dear Doctor:

Herman Watson is to be congratulated on POW! (*DDJ* #29) I have successfully modified it for my SOL computer, linking it to Processor Technology's EDIT program. While writing and modifying the text, the various commands specified by POW can be inserted. It is convenient to employ an unused character as a paragraph break. Then, using the C command of EDIT, change it to the correct string. Then exit to SOLOS, using the H$$ command.

Next, set output port, using SOLOS SET 0 = command. Follow by an EX command as follows: >EX addr1 addr2 where addr1 is the entry to POW and addr2 is the buffer origin. POW will dump the buffer to the selected port.

To modify for some other editor, only the first line of the program needs changing, as follows:

::Change first line, labeled ENTRY to either an LXI H, or an LHLD command to load the address for beginning of dump into the HL registers.

Watson uses a colon to introduce a command. For SOL computers, some control character is better. I have selected ctl-Q for this purpose. Commands must always be terminated by comma, space, or CR. See full description in article.

I have modified the output routine, so that output can be interrupted by pressing any key. POW then waits for further input from the keyboard. The operator has three options. The run can be aborted by pressing MODE. The output can be switched to another pseudoport by pressing one of the number keys 0-3. Finally, output can be resumed by pressing any other key.

Line preceded by :: describe modifications to listing as it appears in published article.

Sincerely yours,
Leonard Morgenstern

304 Rheem Blvd.
Moraga, CA 94556

```
::Insert following lines at start

CR EQU 13
LF EQU 10
SP EQU 32
SOLOS EQU 0C000H Origin of SOLOS or CUTER
SOUT EQU SOLOS+19H
SINP EQU SOLOS+1FH
OPORT EQU 0C807H
PSW EQU 6

ENTRY CALL SOLOS+33AH  Modify for other editors
 SHLD APNT  Initialize dump pointer
 LDA OPORT  store current OPORT
 PUSH PSW
 XRA A
 STA OPORT  Force screen during CLOS and NEWL
 CALL CLOS
 CALL NEWL
 POP PSW
 STA OPORT  Restore OPORT
PRIN LHLD APNT
PRLP MOV A,M
 INX H
 ORA A
 JZ EOF In EDIT, null is end of buffer
 CPI 11H  Defines ctl-Q as introducing a command
 JZ CMND
 MOV B,A


::Copy lines 3B8 to 3DB without change

::CHANGE line 3DE to
 JMP SOLOS+4  Control returns to SOLOS on exit,
  whether due to end of printing, or to interrupt

::Omit lines 3E1 to 427 (Case processing)
::Copy lines 428 to 461 without change
::Omit lines 464 to 4D6
::Copy lines 4D7 to 52F (Command table) WITH THE
     FOLLOWING CHANGE:  Insert two-letter commands
     in normal order, not reversed.
::Copy lines 530 to 5CC without change
::Omit lines 5CD and 5D0
::Copy lines 5D3 to 5EF without change
::Omit lines 5F0 and 5F3
::Copy lines 5F6 to 6FF without change
::Copy parameter list (lines 702 to 757)
    (A maximum buffer length of 132 is specified.
    This.  can be changed by suitably altering
    lines 737 and 757)
::Substitute the following for OUTC and CRLF
          (lines 7E2 to 7BF)
(OUTC, as originally written, assumes that the char
 to be output is in A register.  It also uses
different  assumptions regarding interrupt,
stack, etc.)
OUTC PUSH B  Pusn all
 PUSH D
 PUSH H
 PUSH PSW
 MOV B,A  Move character to B-reg
 LDA OPST  OPST is print parameter in POW
   set and reset by commands ON and OF, respectively
 ORA A
 CNZ SOUT  If OPST=0, then suppress this character
 CALL SINP  Press any key to cause pause
 JZ OUTQ
OUTZ CALL SINP  Wait for another key
 JZ OUTZ
 CPI 80H  MODE key will return to SOLOS
 JZ EOF
 SUI 30H Digits 0-3 switch output to that pseudoport.
Any other key will cause normal processing to resume.
 JM OUTQ
 CPI 4H
 JNC OUTQ
 STA OPORT
OUTQ POP PSW
 POP H
 POP D
```

```
 POP B
 RET
CRLF PUSH B
 CALL SOLOS+2F9H  Call SOLOS CRLF
 POP B
 MVI A,CR  Return with CR in A-reg.
 RET
::Omit lines 7FC to end of text
::Return to beginning of listing, page 20
::Omit lines 0 to AA
::Copy lines AC to 2E8  without change
::Omit lines 2E9 to 2F8
::ATTACH LABEL DVDR TO LINE 2FB

::Copy lines 2FE to 32A without change
::Omit lines 32B and 32E
::Copy lines 32F to 339 without change
::Omit lines 33A and 33D
::Copy lines 340 to 350 without change
::Omit lines 353 and 354
::Copy lines 357 to 39A without change
```

NORTH STAR PATCHES

Dear Dr. Dobbs:

I have been reading *Dr. Dobb's Journal* since the first issue and hope you keep up the policy of providing good articles for the small system user and computer hobbyists.

The following is a list of patches to North Star DOS which allows 40 track disk drives to use all 40 tracks (102.5K).

Steve Lang

Rt. 1
Beloit, WI 53511

DOS 3.0 ADDRESS	DOS 4.0 ADDRESS	OLD DATA	NEW DATA
2255	22A7	A1	6F
22D7	22E3	23	28
2329	2335	5D	90
2345	2354	5E	93
2468	245E	A8	72
2517	251D	5F	91

LOADING KIM'S CASSETTES

Gentledragons:

While trying to adapt OSI/Microsoft's BASIC to my KIM, I discovered a way to load their cassettes. I am enclosing a copy of the program, which is basically MOS Technology's paper tape loader adapted to the KC standard's inverted data and given the ability to read accurately at 300 baud. The KIM's tape interface must be adjusted downward by turning the potentiometer fully counter-clockwise (after noting its present position, of course), and the terminal set to 300 baud, or set $17F2=EB and $17F3=00. Since OSI never uses the ";000000" trailer that indicates end-of-data to the KIM, this routine must be stopped by the use of either the ST or RS keys.

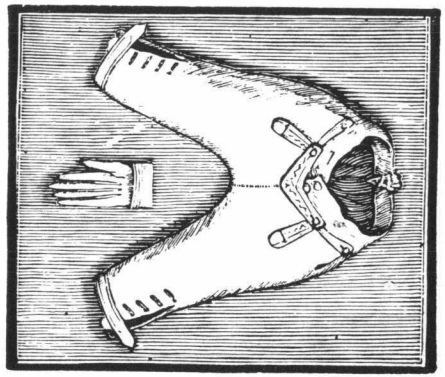
Sincerely,
Jonathan M. Prigot

35 Cummings Road, Apt 6
Brighton, MA 02146


```
                0010  ;COPYRIGHT 1978 BY J.M. PRIGOT.  ALL RIGHTS RESERVED.
                0020  ;
                0030        .OR 0000
0000  D8       0040        CLD
0001  206B00    0050  LOAD  JSR GETC
0004  C93B      0060        CMP  ;
0006  D0F9      0070        BNE LOAD
0008  A900      0080        LDA 00
000A  85F7      0090        STA *CKSM
000C  85F6      0100        STA *CKHI
000E  209600    0110        JSR GETB
0011  AA        0120        TAX
0012  20911F    0130        JSR CHKD
0015  209600    0140        JSR GETB
0018  85FB      0150        STA *PNTH
001A  20911F    0160        JSR CHKD
001D  209600    0170        JSR GETB
0020  85FA      0180        STA *PNTL
0022  20911F    0190        JSR CHKD
                0200  ;
                0210  ;ADD LOW AND HIGH ORDER OFFSETS (IF ANY)
                0220  ;
0025  08        0230        PHP
0026  A5FA      0240        LDA *PNTL
0028  18        0250        CLC
0029  6900      0260        ADC 00          OPERAND=AMOUNT OF LOW-ORDER OFFSET
002B  85FA      0270        STA *PNTL
002D  A5FB      0280        LDA *PNTH
002F  6900      0290        ADC 00          OPERAND=AMOUNT OF HIGH-ORDER OFFSE
0031  85FB      0300        STA *PNTH
0033  28        0310        PLP
                0320  ;
0034  8A        0330        TXA
0035  F00F      0340        BEQ LOD3
0037  209600    0350  LOD2  JSR GETB
003A  91FA      0360        STA (PNTL),Y
003C  20911F    0370        JSR CHKD
003F  20631F    0380        JSR INCP
0042  CA        0390        DEX
0043  D0F2      0400        BNE LOD2
0045  E8        0410        INX
0046  209600    0420  LOD3  JSR GETB
0049  C5F6      0430        CMP *CKHI
004B  D017      0440        BNE LDE1
004D  209600    0450        JSR GETB
0050  C5F7      0460        CMP *CKSM
0052  D013      0470        BNE LODR
0054  8A        0480        TXA
0055  D0AA      0490        BNE LOAD
0057  A20C      0500        LDX 0C
0059  A927      0510  LOD8  LDA 27
005B  8D4217    0520        STA SBD.
005E  20311E    0530        JSR PRTS
0061  4C4F1C    0540        JMP STRT
0064  209600    0550  LDE1  JSR GETB
0067  A211      0560  LODR  LDX 11
0069  D0EE      0570        BNE LOD8
                0580  ;
                0590  ;
006B  86FD      0600  GETC  STX *TMPX
006D  A208      0610        LDX 08
006F  A900      0620        LDA 00
0071  2C4217    0630  GET1  BIT SBD.
0074  10FB      0640        BPL GET1
0076  20D41E    0650        JSR DELA
0079  20EB1E    0660        JSR DEHF
007C  AD4217    0670  GET2  LDA SBD.
007F  2980      0680        AND 80
0081  46FE      0690        LSR *CHAR
0083  05FE      0700        ORA *CHAR
0085  85FE      0710        STA *CHAR
0087  20D41E    0720        JSR DELA
008A  CA        0730        DEX
008B  D0EF      0740        BNE GET2
008D  A6FD      0750        LDX *TMPX
008F  A5FE      0760        LDA *CHAR
0091  2A        0770        ROL
0092  4A        0780        LSR
0093  497F      0790        EOR 7F          INVERT THE DATA
0095  60        0800        RTS
                0810  ;
                0820  ;
0096  206B00    0830  GETB  JSR GETC
0099  20AC1F    0840        JSR PACK
009C  206B00    0850        JSR GETC
009F  20AC1F    0860        JSR PACK
00A2  A5F8      0870        LDA *INL.
00A4  60        0880        RTS
                0890  ;
                0900  CKHI .DL 00F6
                0910  CKSM .DL 00F7
                0920  INL. .DL 00F8
                0930  PNTL .DL 00FA
                0940  PNTH .DL 00FB
                0950  TMPX .DL 00FD
                0960  CHAR .DL 00FE
                0970  SBD. .DL 1742
                0980  STRT .DL 1C4F
                0990  PRTS .DL 1E31
                1000  DELA .DL 1ED4
                1010  DEHF .DL 1EEB
                1020  INCP .DL 1F63
                1030  CHKD .DL 1F91
                1040  PACK .DL 1FAC
                1050  END. .EN

SYMBOL TABLE
LOAD    0001
LOD2    0037
LOD3    0046
LOD8    0059
LDE1    0064
LODR    0067
GETC    006B
GET1    0071
GET2    007C
GETB    0096
CKHI    00F6
CKSM    00F7
INL.    00F8
PNTL    00FA
PNTH    88FB
TMPX    00FD
CHAR    00FE
SBD.    1742
STRT    1C4F
PRTS    1E31
DELA    1ED4
DEHF    1EEB
INCP    1F63
CHKD    1F91
PACK    1FAC
END.    00A5
```

Continued from pg. 35

```
01430 20E7 01              NOP             2 = 10 US
01440 20E8 26 FC           BNE    DELAY    4
01450 20EA 20 D2           BRA    CLEAR    DOWN PROG PULSE
01460 20EC 0D0A     MSG0   FDB    $0D0A
01470 20EE 32              FCC    /2708 PROGRAMMER START ADDRESS:0000/
01480 2110 0D0A            FDB    $0D0A
01490 2112 20              FCC    /                END   ADDRESS:0002/
01500 2134 0D0A            FDB    $0D0A
01510 2136 53              FCC    /SWITCH "PROGRAM" OFF!! /
01520 214D 0D0A            FDB    $0D0A
01530 214F 45              FCC    /ENTER M(MIK),P(PRG),V(VERIFY),L(LIS
T
01540 2174 0D0A            FDB    $0D0A,$4
01550 2178 55       MSG1   FCC    /UT PROM IN SOCKET,PRESS "PROGRAM"SW
I
01560 219F 0D0A            FDB    $0D0A
01570 21A1 20              FCC    / ENTER P/
01580 21A9 0D0A            FDB    $0D0A,$4
01590 21AD 0D0A     MSG2   FDB    $0D0A
01600 21AF 41              FCC    /ADDR PR ME/
01610 21B9 0D0A     MSG3   FDB    $0D0A,$0004
01620                      END

TOTAL ERRORS 00000
```



THE LARGEST KNOWN PRIME

2**23209 – 1

402874115778988778187332907
159176772243850689162242004102996357869459524088740086763986146146653710383329941358659235907550594256021538420320239250528294964596654681298670246293679559813925886213430552407311751401171085095757715576413452669807115402486657029165574361110652155261679243598639770340862427532627451407760338106482412910566360721371747330717615632276966454374276798396931845938405993837987806027557293129788208526372882811767972047853647539734797508580059680824143812665396814199720775822694029571422811120384556621974126218098769121046703851331469580449389637693057945446410758066275109556414379225335014944696332706216486506828452694695378794886597731643600416303767015628898162167541572915251374392867148834988998566477133254890271659341786418266716003149082077075819130286776273368730270198911017821994203875686749027085635603356654222499933446410042566253239062428308184689428990742349499939138597297565091489407937622350523978994767472433544821423418185666750799719662904545551001067948311228733169391906878070780760695075344445437830131751830993383018684952494603514377326291391461671204785509870160512816327881326056457208975094051001640878295466349732839019544150203173181210233434214421009242406304491625683205422660250291638987092995199313646011291485509185885005430715674826920389792071211651845958579726507940677217666633048775528775307355125418312546572211636885200104813266702629472920752149577319512193931311360717000667227690006706061913909910619732429675129112643519470432969800841665528609874717166187763102673117992818498256345720945863515689800012700076995002293116999651571595176656828314289491943888238134356695974575348674606207374249015590114664737390675524203141974659857345920454694440079875042682825674223619126079037828851992942537332753463401043838356143158894195357612153024339317466508455943366947037038077373071649628159782861008338286412064365386598078469217501419145086323286015398645143846658687517110789807148972381671805817286167530625075117472492458080059976221116418632001809075690481059645707924323186702496567189414787013866778869723384754775507562780489087458558571403961608409156400001529427570675790801059481007912360061138160272949540235176575452293684102115698974363028032532362568348021747081292083808815823224978742604974651803265653773566324633198814887977168788134020708650038333581093114016383997899403087553195715489167794426164464076243984359747372548595672163303072512413561039826923076718844384203996647120457477173498533262791864969266274767028353656166351203952980949968134946402925376450259648067636179607575343175999065451862857799363622396409631572789627497352387340180319631371622945193530896716168398561406129344233762928003570023379482911952318919163192545876380237199917398941353576675090388906937622402919803365510062430824750373742704546340555899548295292676532944312559196818662647081505151821208670072062662244135785787663162882922000802040699085958808649045935301957919928360662713973220163364014320769891734422589536470324305588771518925248525541461671468838537699105612463206947055124570889217119786440567031357709473866347297876502417369755432242551472237707416297656501714445411153411865771642670096665217784040079089425722044373575364470132023204490873743591198846888965706531536359656993498881668166753254034197399230120692850477507080580983695461480665305127059225057633393639734487435512752836286568346704027703389012035388258638852733727252968023311757112269544503634933557823964169012365540952244084485151548107189417781442530197357722865554948884829926768347366006636624318505506867453994455209613903356661381551303170707697066559695466821068447680650509746035384405738968952295270831628476713602096813200521662297702228641383037720127445077295416522867251086259308924256949130546482529205107067023263892250887651080363021211434271433397608989308330364300732585576535892044008832875413115983363441152995391893368657891779064964168643604233277423002160282408650623560137375330537862115987368055570643448866407958522158010929886419239414559369282323995582350982708215349265311035134009700812402989211058114698144382506102890169695315562317736449187133984510136109015630882603126705678523924068885469132194085851583130600527208280199569403550998247082238995498157197338242163511583218466756077718103341385716065590415776510605985864109452424143510105710136533402909514179370387940205512396293869817982858458904343162346156521709247471250410809550718513708436140629037494897533788407388963337967019452014316250553598248055724074565078950423399377604871096508073885741648498853100338089580311470238617139347091804873541481838039245807119326864157984193826793548100978606147490699272167835318448719485024999063698211242578366742398776579568702614008789586297723548299737682672331537411323791490133380537171174143349785173683206999265385587000255356918684065909860721703394780414441674592801942409869215643511998397650823181631838937922195762880843293326015802569692130149889856075048804362100069485396073838839866313202368738877263582786544468490404384228812146483253624666106598128595021876652656304044747719532951373264109049549817756983623389046479884300530772600321667698144843391181435818783099756242619763936377048473380949365998918337881431961023816361930620858134690116948733230732499928981637440392950625116696620369791068804420190437337504706557753990341231870234321605170515557947761192933448713338636368080179510595537709371721774062906763611519438033187285051664873543460767663152274573392269144056818971746620021379034199200600241092469942996433550737726490057174886308000943528027580668323191993263432819944316604346404680550290495282346830424043178905143114575017478057302915937660889742311665775464699337725205899658431702118651785954540294855464081581258011181213571924409492777760513891824057453481356182500330654244571788812378520927064783435122985854726038472730062101845156241692016024941211265239314166182038758409312685384765540405212510964896838371037304156419933879530352086544022535109789602027747223650858853482671648101755365481354228091863805011621578883875669404903355089888549376208723682090564341102930475228163313059215959271226081949848477889353337470684677801928746328252980236259643089082064765819268977684271297091434221420402203204816051843790924312901306330991292496802744363898396613407782196679612383549415610569783891526919396336077461916416010666683822481030966646694241533372863428935782537171867644640829719942202233252584945211114561669570501261079454174419086577258210544440106662438203017786099053725933264345420108025868071488561671372008089600018372791881207012696110945775614694650737518542322556296640977293935894214414869438855358753606003397001637907083353780707434779613877126249098033493565718268305571578182703693122089543445743786026405786163677455358760270973348463589339098079294709002676276094965852942303062635499907856117575009515746557862539764756574427752110896827606786025282039152876055050854511817293890036743355523779264511

ON 2/9/79, CURT NOLL OF CAL STATE HAYWARD FOUND THIS NUMBER TO BE PRIME USING AN IMPLEMENTATION OF THE LUCAS-LEHMER TEST ON A CDC CYBER 174. THE PROOF OF PRIMALITY TOOK 8 HOURS, 40 MINUTES.

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 33 | March 1979 | Volume 4, Issue 3

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

Falconer Floating Point Arithmetic, Part I 4
Charles B. Falconer

CONVPATB *Paul Holliday* 15
Palo Alto Tiny BASIC to ASCII

Label BASIC *Sylvan Rubin* 40

QUOLST *Gary Gaugler* 24

Vice Versa *Jim Gagne* 26
Pencil to CPM and Reverse

Sorting Sequential Files in Place 25
David Burris *Wayne Byrd*

A Critical Look at the MC6800 38
Curtis Roads

Reader Exchange 45
... and much more

Falconer Floating Point Arithmetic

BY CHARLES B. FALCONER
Chief Instrumentation Engineer
Yale University
School of Medicine
333 Cedar Street
New Haven, Connecticut 06510

© Charles B. Falconer

This system has been tried and tested for a couple of years, and we're happy to bring it to you. Due to its length, the article will continue in Part II next month. –SR

This software is released only for publication by *Doctor Dobb's Journal*, and for the private non-profit, non-commercial use of individuals and educational institutions. Incorporation in any instrument, software package, ROM, etc. for sale is expressly forbidden without written consent from myself, at which time any such use must be clearly identified as licensed.

I would appreciate any *DDJ* reader who observes such unlicensed use advising me of the circumstances.

This software package provides full floating point arithmetic capability for 8080, 8085, and Z80 based machines, with very little speed penalty over use of integer arithmetic. The system can handle all 16 bit integer values (treated either as positive integers in the range 0 to 65535, or as signed integers in the range –32768 to 32767) without any loss of information, yet the extended range (approximately 10 ↑ -38 to 10 ↑ 38) convenience of floating point is directly available, together with various mathematical functions (i.e. logarithms, exponentials, etc.) I hope that this publication will discourage use of integer arithmetic in various interpreters, compilers, etc.

The operand size is such as to allow virtually all inner loops to be performed entirely in registers, with attendant speed benefits. At a 518 nanosec. clock, most arithmetic operations are performed in one millisec. or less, division in about 1.5 millisecs. A 4 MHZ Z80 system can halve this. This is not far removed from integer execution times.

I strongly recommend that any user of this package do so without any lateration (except deletion of routines not needed in the application). This system has been in routine operation since April 1976, and is incorporated in other systems. The lease exercised routines are in FLTINPUT and FUNCTION. I would appreciate being advised of any bugs or anomalies found.

While the calling convention to IVAL, for example, may appear unusual, it has been specifically designed to allow re-entrant use with various data sources. A calling routine of the form:

```
GETNUM: PUSH B
        LXI  B, (address of char. input routine)
        CALL IVAL;   (or IVALC)
        POP  B
        RET
```

will customize this as required and avoid conflict with any other software.

The system does not have the awkward range limitation of the EA chip (approximately 10 ↑-18 to 10 ↑ 18) and some other systems, which I have found to bite the user all too often. Due to the careful treatment of rounding and over/underflow, the system will usually give the expected results (inversion of large matrices is the only exception noted so far, yet systems of 5 to 10 simultaneous equations, on our typical input data, can be handled adequately), and the error and overflow reporting will provide adequate warnings of run-time misuse. Error analysis is mathematically tractable.

The AERC (arithmetic error) connector allows trappings of illegal operands (Division by zero, logarithm of negative number, etc.). If desired this can be specified as:

```
AERC:   STC
        RET
```

which will cause all such occurrences to be treated as overflows.

In general, any error is signaled by carry set on routine exit. The individual routine functions are stated in the commentary, and registers disturbed are listed.

Notice that the unspecified I/O routines output register (C), and return with (A) set to (C) on entry. Input routines return characters in (A).

Also notice that all routines (except those specifically using I/O devices) are re-entrant, and use only stack assigned temporary storage. The code is thus inherently suitable for the time sharing or interrupt driven applications. Users who have been using integer arithmetic may find that extra stack space must be allocated. An extra 32 bytes should normally suffice.

For the benefit of readers without a macro assembler, macros used generally depend on the definition of registers as B=0, C=1, D=2, E=3, H=4, L-5, M=SP=PSW=6, A=7. Between the macro definitions and the actual code generated, users should be able to substitute suitable source code. Our assembler permits "a" and "." in names, automatically increments and decrements the global variable. LVL on each PUSH or POP instruction respectively, and can generate relocatable code. Source code between IF and ENDIF statements is assembled only if the IF operand is non-zero, and between IFZ and ENDIF only if the IFZ operand is zero. This listing uses this construct only within macros. The "EXTRN" lists how external addresses required within a module, and the "ENTRY" lists show address required by other modules. Note that some character constants may be in lower case (and are so marked). The printer used translated these to upper case. This should be sufficient to allow the user to customize the source to his assembler.

I originally intended to submit listings with all macros fully expanded, but the results were excessively cluttered, no longer held any clarity, and the listing length was virtually doubled. As submitted, all information remains present, but the reader may have to refer to the macro definitions fairly often.

The load map shows the location of all routines when loaded at location 1000 (hex) up, in the order shown. The "U" identifies lables undefined in the overall system.

Historically, this system replaced an earlier design (published in the Intel User Library) which used 2's complement mantissas and exponents. That system had 1 less bit of resolution, control of negative extremes and rounding has a problem, and was 5 to 10 times slower than the present system.

listing

```
FLOATMAP USLIST FALCONER

LOAD MAP FOR OBJECT FILE: $NEWPASS.PUB.FALCONER
   CODE     DATA     SEGMENT
========== ========= =======
1000-115A   NONE     INTARITH.USLOBJ.FALCONER
1158-1463   NONE     FLTARITH.USLOBJ.FALCONER
1464-15BB   NONE     FLTINPUT.USLOBJ.FALCONER
15BC-17D2   NONE     FLTOUT.USLOBJ.FALCONER
17D3-19FA   NONE     FUNCTION.USLOBJ.FALCONER

  .AND    1986    .EQU    19A6     .GE.    19D7     .GG.    19E0
  .GT.    19C5    .LE     19BC     .LT.    19CE     .NE.    19AF
  .OR.    1971    .XOR    1996   U-AERC    0000     BCLC    102B
  BCL2    102A    BCRA    1032     BCRC    1034     C1BC    1045
  C1DE    104D    C2BC    1044     C2DE    104C     C2DHL   1054
U-COUT    0000    DEBLK   1479     DERA    1038     DERC    1030
  DHL2    1022    DIV     1114     DQUIK   1082     DTEN    106A
  EXDG    15FF    EXP2    18E1     EXPX    1931     FADD    1303
  FCMP    1387    FDIV    1248     FDIVR   12A5     FDIVI   1202
  FINT    130C    FIXR    1439     FIXT    1404     FLOAD   11DC
  FLOT    13E1    FLOTA   1389     FLOTD   13D7     FLOTP   13CE
  FMAT    1728    FMOD    17EE     FMUL    1219     FMULT   11F4
  FRACT   1703    FRCIP   12B1     FSTOR   11E8     FSUB    137A
  FSUBR   12FF    FXCHG   1185     IDIV    10A7     IMUL    108A
  IVAL    1482    IVALC   1485     JBC     1477     LAS     101A
  LBCS    1011    LDES    1008     LFBIS   11A9     LFBS    119C
  LFDIS   1189    LFDS    118F     LFLT    170C     LOG2    1823
  LOGB    188C  U-LOUT    0000     MUL     10D9     MULIO   1061
  NLRD    1158    OD2S    15E1     OFLT    171C     ONEG    15DC
  OPT     15D7    OTCBK   15BC     OTCCL   15BE     POLY    18AB
  QMAX    1468    QNUM    1464     SFDIS   11C9     STADR   1000
  TFLT    1714

;*****************************************
;*******   SEGMENT INTARITH   ************
;*****************************************
0001000 ; COPYRIGHT YALE UNIVERSITY
0002000 ;
0003000 ; BY C.B. FALCONER                (203) 436-2603
0004000 ;                    ROOM 6016CB
0005000 ;                    YALE SCHOOL OF MEDICINE
0006000 ;                    789 HOWARD AVENUE
0007000 ;                    NEW HAVEN, CONN 06504
0008000 ;
0009000 ;                    **************************
0010000 ;                    RSEG:  * INTEGER ARITHMETIC *
0011000 ;                    **************************
0012000 ;
0013000 ; ALLOWABLE ENTRY POINTS
0014000 ;       ENTRY  IMUL,IDIV,MUL,DIV
0015000 ;
0016000 ; ***************************************
0017000 ; ENTRY POINTS FOR UTILITY ROUTINES
0018000 ;       ENTRY  STADR,LDES,LBCS,LAS; STACK ADDRESSING
0019000 ;       ENTRY  BCL2,BCLC,BCRA,BCRC; SHIFTS AND COMPLEMENTS
0020000 ;       ENTRY  DHL2,DERA,DERC
0021000 ;       ENTRY  C2BC,C1BC,C2DE,C1DE,C2DHL
0022000 ;       ENTRY  MUL10,DTEN,DQUIK; FAST ARITHMETIC
0023000 ; ***************************************
0024000 ;
0025000 ; MACRO DEFINITIONS
0026000 ;
0027000 ; "RETURN" AND CHECK STACK LEVEL ZERO
0028000 RTN     MACRO
0029000         IF      LVL
0030000         ERROR   "0"+LVL
0031000 LVL     SET     0
```

```
                        0032000          ENDIF
                        0033000          RET
                        0034000 '        ENDM
                        0035000 ;
                        0036000 ; ********************
                        0037000 ; * UTILITY ROUTINES *
                        0038000 ; ********************
                        0039000 ;
                        0040000 ; STACK ADDRESSING ROUTINES OPERATE ON AN INPUT
                        0041000 ; STACK LEVEL, SUPPLIED VIA THE (A) REGISTER
                        0042000 ; THIS SPECIFIES THE STACK LEVEL WITH RESPECT
                        0043000 ; TO THE CALLING ROUTINE, DERIVED BY COUNTING
                        0044000 ; "PUSHES" SINCE THE ITEM WAS PUSHED.  IF THE ITEM
                        0045000 ; WAS STORED BY THE LAST "PUSH" IT'S ADDRESS IS
                        0046000 ; ZERO.  THE ADDRESS MAY NOT EXCEED 252
                        0047000 ;
                        0048000 ; GENERATE STACK ABSOLUTE ADDRESS FOR
                        0049000 ; STACK ADDRESSING ROUTINES
                        0050000 ; A,F,H,L
0000 210300             0051000 STADR:  LXI     H,3;        ALLOW FOR PUSH H & 2 RET'S
0003 85                 0052000         ADD     L;          MAX. STACK LEVEL IS 252
0004 6F                 0053000         MOV     L,A
0005 29                 0054000         DAD     H;          CONVERT TO BYTE ADDRESS
0006 39                 0055000         DAD     SP;         MEMORY ADDRESS FORMED
0007 C9                 0056000         RTN
                        0057000 ;
                        0058000 ; LOAD (DE) FROM STACK LEVEL (A)
                        0059000 ; A,F,D,E
0008 E5                 0060000 LDES:   PUSH    H
0009 CD0000             0061000         CALL    STADR;      GET ABS. ADDRESS
000C 5E                 0062000         MOV     E,M
000D 23                 0063000         INX     H
000E 56                 0064000         MOV     D,M
000F E1                 0065000         POP     H
0010 C9                 0066000         RTN
                        0067000 ;
                        0068000 ; LOAD (BC) FROM STACK LEVEL (A)
                        0069000 ; (A) IS STACK LEVEL W.R.T. CALLING ROUTINE
                        0070000 ; A,F,B,C
0011 E5                 0071000 LBCS:   PUSH    H
0012 CD0000             0072000         CALL    STADR;      FORM ABSOLUTE ADDRESS
0015 4E                 0073000         MOV     C,M
0016 23                 0074000         INX     H
0017 46                 0075000         MOV     B,M
0018 E1                 0076000         POP     H
0019 C9                 0077000         RTN
                        0078000 ;
                        0079000 ; LOAD (A) FROM STACK LEVEL (A)
                        0080000 ; VALUE WAS STORED BY PUSH PSW
                        0081000 ; A,F
001A E5                 0082000 LAS:    PUSH    H
001B CD0000             0083000         CALL    STADR
001E 23                 0084000         INX     H;          FORM ABS. ADDRESS
001F 7E                 0085000         MOV     A,M
0020 E1                 0086000         POP     H
0021 C9                 0087000         RTN
                        0088000 ;
                        0089000 ; SHIFT (DEHL) REGISTER LEFT, INSERT 0
                        0090000 ; ORIGINAL HIGH ORDER BIT TO CARRY
                        0091000 ; A,F,D,E,H,L (A=D ON EXIT)
0022 29                 0092000 DHLZ:   DAD     H
0023 7B                 0093000         MOV     A,E
0024 17                 0094000         RAL
0025 5F                 0095000         MOV     E,A
0026 7A                 0096000         MOV     A,D
0027 17                 0097000         RAL
0028 57                 0098000         MOV     D,A
0029 C9                 0099000         RTN
                        0100000 ;
                        0101000 ; (BC) LEFT SHIFT, ZERO INSERT, LEAVE (B) IN (A)
                        0102000 ; A,F,B,C
002A B7                 0103000 BCLZ:   ORA     A;          CLEAR CARRY
                        0104000 ;
                        0105000 ; (BC) LEFT SHIFT, CARRY INSERT, LEAVE (B) IN (A)
                        0106000 ; A,F,B,C
002B 79                 0107000 BCLC:   MOV     A,C
002C 17                 0108000         RAL
002D 4F                 0109000         MOV     C,A
002E 78                 0110000         MOV     A,B
002F 17                 0111000         RAL
0030 47                 0112000         MOV     B,A
0031 C9                 0113000         RTN
                        0114000 ;
                        0115000 ; ARITH. SHIFT RIGHT (BC), LEAVE (C) IN (A)
                        0116000 ; A,F,B,C
0032 78                 0117000 BCRA:   MOV     A,B
0033 17                 0118000         RAL
                        0119000 ;
                        0120000 ; (BC) RIGHT SHIFT, CARRY IN, LEAVE (C) IN (A)
                        0121000 ; A,F,B,C
0034 78                 0122000 BCRC:   MOV     A,B
0035 1F                 0123000         RAR
0036 47                 0124000         MOV     B,A
0037 79                 0125000         MOV     A,C
0038 1F                 0126000         RAR
0039 4F                 0127000         MOV     C,A
003A C9                 0128000         RTN
                        0129000 ;
                        0130000 ; ARITHMETIC RIGHT SHIFT (DE), LEAVE (E) IN (A)
                        0131000 ; A,F,D,E
003B 7A                 0132000 DERA:   MOV     A,D
003C 17                 0133000         RAL
                        0134000 ;
                        0135000 ; (DE) RIGHT SHIFT, CARRY INSERT, LEAVE (E) IN (A)
                        0136000 ; A,F,D,E
003D 7A                 0137000 DERC:   MOV     A,D
003E 1F                 0138000         RAR
003F 57                 0139000         MOV     D,A
0040 7B                 0140000         MOV     A,E
0041 1F                 0141000         RAR
0042 5F                 0142000         MOV     E,A
0043 C9                 0143000         RTN
                        0144000 ;
                        0145000 ; 2'S COMPLEMENT (BC), LEAVE (B) IN (A)
                        0146000 ; A,B,C
0044 0B                 0147000 C2BC:   DCX     B
                        0148000 ;
                        0149000 ; 1'S COMPLEMENT (BC), LEAVE (B) IN (A)
                        0150000 ; A,B,C
0045 79                 0151000 C1BC:   MOV     A,C
0046 2F                 0152000         CMA
0047 4F                 0153000         MOV     C,A
0048 78                 0154000         MOV     A,B
0049 2F                 0155000         CMA
004A 47                 0156000         MOV     B,A
004B C9                 0157000         RTN
                        0158000 ;
                        0159000 ; 2'S COMPLEMENT (DE), LEAVE (D) IN (A)
                        0160000 ; A,D,E
004C 1B                 0161000 C2DE:   DCX     D
                        0162000 ;
                        0163000 ; 1'S COMPLEMENT (DE), LEAVE (D) IN (A)
                        0164000 ; A,D,E
004D 7B                 0165000 C1DE:   MOV     A,E
004E 2F                 0166000         CMA
004F 5F                 0167000         MOV     E,A
0050 7A                 0168000         MOV     A,D
0051 2F                 0169000         CMA
0052 57                 0170000         MOV     D,A
0053 C9                 0171000         RTN
                        0172000 ;
                        0173000 ; 2'S COMPLEMENT (DEHL)
                        0174000 ; A,F,D,E,H,L
0054 EB                 0175000 C2DHL:  XCHG
0055 CD4C00             0176000         CALL    C2DE
0058 EB                 0177000         XCHG
0059 CD4D00             0178000         CALL    C1DE
005C 7C                 0179000         MOV     A,H
005D B5                 0180000         ORA     L
005E C0                 0181000         RNZ
005F 13                 0182000         INX     D;          PROPAGATE CARRY
0060 C9                 0183000         RTN
```

```
                        0184000 ;
                        0185000 ; MULTIPLY (HL) BY 10 (MODULO 65536)
                        0186000 ; NO OVERFLOW SIGNAL
                        0187000 ; F,H,L
0061 D5                 0188000 MUL10:  PUSH    D
0062 54                 0189000         MOV     D,H
0063 5D                 0190000         MOV     E,L;        COPY HL TO DE
0064 19                 0191000         DAD     D;          2*
0065 29                 0192000         DAD     H;          4*
0066 19                 0193000         DAD     D;          5*
0067 29                 0194000         DAD     H;          10*
0068 D1                 0195000         POP     D;          RESTORE DE
0069 C9                 0196000         RTN
                        0197000 ;
                        0198000 ; DIVIDE INTEGER (HL) BY 10
                        0199000 ; REMAINDER APPEARS IN (A) WITH FLAGS SET
                        0200000 ; A,F,H,L
006A C5                 0201000 DTEN:   PUSH    B;          SAVE BC
006B 0E0A               0202000         MVI     C,10;       DIVISOR
006D AF                 0203000 DTEN1:  XRA     A;          CLEAR
006E 06F0               0204000         MVI     B,-16;      ITERATION COUNT
0070 29                 0205000 DTEN2:  DAD     H
0071 17                 0206000         RAL;                SHIFT OFF INTO (A)
0072 DA7900             0207000         JC      DTEN3;      ALLOW FOR DQUIK
0075 B9                 0208000         CMP     C;          TEST
0076 DA7B00             0209000         JC      DTEN4;      NO BIT
0079 91                 0210000 DTEN3:  SUB     C;          BIT=1
007A 23                 0211000         INX     H
007B 04                 0212000 DTEN4:  INR     B;          DONE?
007C FA7000             0213000         JM      DTEN2;      NO
007F B7                 0214000         ORA     A;          SET FLAGS FOR RDR., CLEAR CARRY
0080 C1                 0215000         POP     B;          RESTORE
0081 C9                 0216000         RTN
                        0217000 ;
                        0218000 ; ** THIS ROUTINE IS NOT USED IN THE FLTARITH SYSTEM **
                        0219000 ; INTEGER DIVIDE 16 BY 8 BIT QUANTITIES
                        0220000 ; (HL)/(A) => (HL); REMAINDER => (A)
                        0221000 ; SET CARRY FOR DIVISION BY ZERO, PRESERVE HL
                        0222000 ; A,F,H,L
0082 B7                 0223000 DQUIK:  ORA     A
0083 37                 0224000         STC
0084 C8                 0225000         RZ;                 DIVISION BY ZERO
0085 C5                 0226000         PUSH    B
0000                    0227000  LVL    SET     .LVL-1
0086 4F                 0228000         MOV     C,A
0087 C36D00             0229000         JMP     DTEN1
                        0230000 ;
                        0231000 ; * END UTILITY ROUTINES *
                        0232000 ; ************************
                        0233000 ;
                        0234000 ; INTEGER (POS.) MULTIPLY DE*BC->DEHL
                        0235000 ; OPERAND RANGE 0 TO 65535
                        0236000 ; D,E,H,L
008A F5                 0237000 IMUL:   PUSH    PSW
008B 210000             0238000         LXI     H,0;        CLEAR ACCUMULATOR
008E 3EF0               0239000         MVI     A,-16;      ITERATION COUNT
0090 F5                 0240000 IMUL1:  PUSH    PSW;        SAVE ITERATION COUNT
0091 29                 0241000         DAD     H;          LEFT SHIFT, CARRY OUT
0092 7B                 0242000         MOV     A,E;        LEFT SH M'PLIER, INSERT D'FLOW
0093 17                 0243000         RAL
0094 5F                 0244000         MOV     E,A
0095 7A                 0245000         MOV     A,D
0096 17                 0246000         RAL
0097 57                 0247000         MOV     D,A
0098 D2A000             0248000         JNC     IMUL2;      NO BIT
009B 09                 0249000         DAD     B;          ADD IN MULTIPLICAND
009C D2A000             0250000         JNC     IMUL2;      NO OVERFLOW
009F 13                 0251000         INX     D;          KEEP OVERFLOW
00A0 F1                 0252000 IMUL2:  POP     PSW;        ITERATION COUNT
00A1 3C                 0253000         INR     A
00A2 FA9000             0254000         JM      IMUL1;      DO AGAIN
00A5 F1                 0255000         POP     PSW;        RESTORE
00A6 C9                 0256000         RTN
                        0257000 ;
                        0258000 ; INTEGER (POS.) DIVIDE. (DEHL)/(BC)=>(DE)
                        0259000 ; REMAINDER APPEARS IN (HL)
                        0260000 ; CARRY FOR OV, WHEN REGISTERS UNCHANGED
                        0261000 ; DIVISOR, REMAINDER AND QUOTIENT RANGE 0 TO 65535
                        0262000 ; DIVIDEND RANGE 0 TO 4295*10^6 (APPROX)
                        0263000 ; F,D,E,H,L
00A7 F5                 0264000 IDIV:   PUSH    PSW
00A8 7B                 0265000         MOV     A,E;        CHECK FOR OVERFLOW
00A9 91                 0266000         SUB     C
00AA 7A                 0267000         MOV     A,D
00AB 98                 0268000         SBB     B
00AC DAB200             0269000         JC      IDIV1;      NO OVERFLOW
00AF F1                 0270000         POP     PSW;        RESTORE (A)
00B0 37                 0271000         STC;                MARK OVERFLOW
00B1 C9                 0272000         RTN
0001                    0273000  LVL    SET     .LVL+1
00B2 C5                 0274000 IDIV1:  PUSH    B
00B3 CD4400             0275000         CALL    C2BC;       CHANGE (BC) SIGN
00B6 EB                 0276000         XCHG;               DO ARITHMETIC IN (HL)
00B7 3EF0               0277000         MVI     A,-16;      ITERATION COUNT
00B9 F5                 0278000 IDIV2:  PUSH    PSW;        SAVE ITERATION COUNT
00BA 29                 0279000         DAD     H;          LEFT SHIFT (HLDE)
00BB 1F                 0280000         RAR;                SAVE CARRY OUT
00BC EB                 0281000         XCHG
00BD 29                 0282000         DAD     H
00BE EB                 0283000         XCHG
00BF D2C300             0284000         JNC     IDIV3;      NO CARRY INTO L
00C2 23                 0285000         INX     H
00C3 17                 0286000 IDIV3:  RAL;                REGAIN CARRY FROM H
00C4 DACE00             0287000         JC      IDIV4;      YES, GENERATE QUOTIENT BIT
00C7 7D                 0288000         MOV     A,L
00C8 81                 0289000         ADD     C;          TEST FOR QUOTIENT BIT
00C9 7C                 0290000         MOV     A,H
00CA 88                 0291000         ADC     B
00CB D2D000             0292000         JNC     IDIV5;      NO BIT
00CE 09                 0293000 IDIV4:  DAD     B;          SUBTRACT
00CF 13                 0294000         INX     D;          INSERT QUOTIENT BIT
00D0 F1                 0295000 IDIV5:  POP     PSW;        GET ITERATION COUNT
00D1 3C                 0296000         INR     A
00D2 FAB900             0297000         JM      IDIV2;      NOT DONE
00D5 C1                 0298000         POP     B;          RESTORE BC
00D6 F1                 0299000         POP     PSW;        RESTORE A
00D7 B7                 0300000         ORA     A;          CLEAR ANY CARRY, NO OVERFLOW
00D8 C9                 0301000         RTN
                        0302000 .
                        0303000 ; ** THIS ROUTINE NOT USED IN THE FLTARITH SYSTEM **
                        0304000 ; SIGNED MULTIPLY (DE)*(BC)->(DEHL)
                        0305000 ; F,D,E,H,L
00D9 F5                 0306000 MUL:    PUSH    PSW
00DA C5                 0307000         PUSH    B
00DB 7A                 0308000         MOV     A,D
00DC B7                 0309000         ORA     A
00DD FAF100             0310000         JM      MUL3;       (DE) -VE
00E0 78                 0311000         MOV     A,B
00E1 B7                 0312000         ORA     A
00E2 F2FC00             0313000         JP      MUL4;       BOTH +VE
00E5 CD4400             0314000 MUL1:   CALL    C2BC;       2'S COMP  BC
00E8 CD8A00             0315000 MUL2:   CALL    IMUL;       RESULT -VE
00EB CD5400             0316000         CALL    C2DHL;      2'S COMPLEMENT DEHL
00EE C3FF00             0317000         JMP     MUL5
00F1 CD4C00             0318000 MUL3:   CALL    C2DE;       (DE) -VE
00F4 78                 0319000         MOV     A,B
00F5 B7                 0320000         ORA     A
00F6 F2E800             0321000         JP      MUL2;       (DE) -VE, (BC) +VE
00F9 CD4400             0322000         CALL    C2BC;       (DE) -VE, (BC) -VE
00FC CD8A00             0323000 MUL4:   CALL    IMUL;       RESULT +VE
00FF C1                 0324000 MUL5:   POP     B
0100 F1                 0325000         POP     PSW
0101 B7                 0326000         ORA     A;          RESET CARRY, NO OV
0102 C9                 0327000         RTN
                        0328000 ;
                        0329000 ; ** THIS ROUTINE NOT USED IN THE FLTARITH SYSTEM **
                        0330000 ; DO IDIV ON SIGNED + NO'S & CHECK O'FLOW
                        0331000 ; EXPECTING +VE RESULT
                        0332000 ; A,F,D,E,H,L
0103 CDA700             0333000 IDIV0:  CALL    IDIV
0106 D8                 0334000         RC
0107 7A                 0335000         MOV     A,D
```

```
0108 17          0336000          RAL
0109 C9          0337000          RTN;              RESULT SHOULD BE +VE
                 0338000 ;
                 0339000 ; ** THIS ROUTINE NOT USED IN THE FLTARITH SYSTEM **
                 0340000 ; DO IDIV ON SIGNED +NO'S & CHECK OVERFLOW
                 0341000 ; INPUTS MAY INCLUDE 8000 HEX
                 0342000 ; EXPECTING -VE RESULT, ALLOW 8000 HEX
                 0343000 ; A,F,D,E,H,L
010A CDA700      0344000 IDIVN:   CALL    IDIV
010D D8          0345000          RC;               OVERFLOW
010E CD4C00      0346000          CALL    C2DE;     COMPLEMENT QUOTIENT
0111 17          0347000          RAL;              RESULT SHOULD BE -VE
0112 3F          0348000          CMC
0113 C9          0349000          RTN
                 0350000 ;
                 0351000 ; ** THIS ROUTINE NOT USED IN THE FLTARITH SYSTEM **
                 0352000 ; SIGNED DIVIDE (DEHL)/(BC)->(DE)
                 0353000 ; REMAINDER APPEARS IN (HL)
                 0354000 ; CARRY INDICATES OVERFLOW
                 0355000 ; WHEN INPUTS ARE PRESERVED, EXCEPT FLAGS
                 0356000 ; F,D,E,H,L  (9)
0114 F5          0357000 DIV:     PUSH    PSW
0115 C5          0358000          PUSH    B
0116 D5          0359000          PUSH    D
0117 E5          0360000          PUSH    H;        SAVE IN CASE OF OVERFLOW
0118 7A          0361000          MOV     A,D
0119 B2          0362000          ORA     D
011A FA3C01      0363000          JM      DIV4;     DIVIDEND NEGATIVE
011D B0          0364000          ORA     B
0004             0365000 @01      SET     .LVL
011E FA2D01      0366000          JM      DIV2;     +/-
0121 CD0301      0367000          CALL    IDIVQ;    +/+
0124 DA3601      0368000          JC      DIV3;     OVERFLOW
0127 C1          0369000 DIV1:    POP     B;        PURGE STACK, NO OVERFLOW
0128 C1          0370000          POP     B
0129 C1          0371000          POP     B;        RESTORE (BC)
012A F1          0372000          POP     PSW
012B B7          0373000          ORA     A;        RESET CARRY, NO OVERFLOW
012C C9          0374000          RTN
0004             0375000  LVL     SET     @01
012D CD4400      0376000 DIV2:    CALL    C2BC;     +/-, COMPLEMENT BC
0130 CD0A01      0377000          CALL    IDIVN
0133 D22701      0378000          JNC     DIV1;     NO OVERFLOW
0136 E1          0379000 DIV3:    POP     H;        RESTORE ENTRY, OVERFLOW
0137 D1          0380000          POP     D
0138 C1          0381000          POP     B
0139 F1          0382000          POP     PSW
013A 37          0383000          STC;              MARK OVERFLOW WITH CARRY
013B C9          0384000          RTN
013C CD5400      0385000 DIV4:    CALL    C2DHL;    -/?, COMP. DEHL
013F 78          0386000          MOV     A,B
0140 B7          0387000          ORA     A
0141 FA5201      0388000          JM      DIV7;     -/-
0144 CD0A01      0389000          CALL    IDIVN
0147 DA3601      0390000 DIV5:    JC      DIV3;     OVERFLOW
014A EB          0391000 DIV6:    XCHG
014B CD4C00      0392000          CALL    C2DE
014E EB          0393000          XCHG;             COMPLEMENT REMAINDER
014F C32701      0394000          JMP     DIV1
0152 CD4400      0395000 DIV7:    CALL    C2BC;     -/-, COMPLEMENT BC
0155 CD0301      0396000          CALL    IDIVQ
0158 C34701      0397000          JMP     DIV5
                 0398000 ;
0000             0399000          END     0
PASS 2 ERRORS = 0
.LVL   REL    00000H    @01    REL    00004H    BCLC   IDR    0002BH
BCLZ   IDR    0002AH    BCRA   IDR    00032H    BCRC   IDR    00034H
C1BC   IDR    00045H    C1DE   IDR    0004DH    C2BC   IDR    00044H
C2DE   IDR    0004CH    C2DHL  IDR    00054H    DERA   IDR    0003BH
DERC   IDR    0003DH    DHLZ   IDR    00022H    DIV    IDR    00114H
DIV1   IDR    00127H    DIV2   IDR    0012DH    DIV3   IDR    00136H
DIV4   IDR    0013CH    DIV5   IDR    00147H    DIV6   IDR    0014AH
DIV7   IDR    00152H    DQUIK  IDR    00082H    DTEN   IDR    0006AH
DTEN1  IDR    0006DH    DTEN2  IDR    00070H    DTEN3  IDR    00079H
DTEN4  IDR    0007BH    IDIV   IDR    000A7H    IDIV1  IDR    000B2H
IDIV2  IDR    000B9H    IDIV3  IDR    000C3H    IDIV4  IDR    000CEH
IDIV5  IDR    000D0H    IDIVN  IDR    0010AH    IDIVQ  IDR    00103H
IMUL   IDR    0008AH    IMUL1  IDR    00090H    IMUL2  IDR    000A0H
LAS    IDR    0001AH    LBCS   IDR    00011H    LDES   IDR    00008H
MUL    IDR    000D9H    MUL1   IDR    000E5H    MUL10  IDR    00061H
MUL2   IDR    000E8H    MUL3   IDR    000F1H    MUL4   IDR    000FCH
MUL5   IDR    000FFH    RTN    MACRO             STADR  IDR    00000H

##########################################
########    SEGMENT FLTARITH    ##########
##########################################

                 0001000 ; COPYRIGHT YALE UNIVERSITY
                 0002000 ;
                 0003000 ; BY C.B. FALCONER                (203) 436-2603
                 0004000 ;                 ROOM 6016CB
                 0005000 ;                 YALE SCHOOL OF MEDICINE
                 0006000 ;                 789 HOWARD AVENUE
                 0007000 ;                 NEW HAVEN, CONN 06504
                 0008000 ;
                 0009000 ;                      ********************************
                 0010000          RSEG;         * FLOATING ARITHMETIC ROUTINES *
                 0011000 ;                      ********************************
                 0012000 ;
                 0013000 ; EXTERNAL ROUTINES REQUIRED
                 0014000          EXTRN  AERC;      ARITH. ERROR TRAP
                 0015000 ;
                 0016000 ; EXTERNAL ROUTINES IN INTARITH
                 0017000          EXTRN  C1DE,C2BC,C2DE,DERC,DHLZ
                 0018000          EXTRN  IDIV,IMUL,STADR
                 0019000 ;
                 0020000 ; ALLOWABLE ENTRY POINTS
                 0021000 ;
                 0022000 ; DATA MANIPULATION
                 0023000          ENTRY  MLRD,FXCHG
                 0024000 ;
                 0025000 ; MEMORY ADDRESSING
                 0026000          ENTRY  LFDS,LFBS,FLOAD,FSTOR
                 0027000          ENTRY  LFBIS,LFDIS,SFDIS
                 0028000 ;
                 0029000 ; ARITHMETIC OPERATORS
                 0030000          ENTRY  FMULT,FDIVT,FMUL
                 0031000          ENTRY  FDIV,FDIVR,FRCIP
                 0032000          ENTRY  FADD,FSUB,FSUBR
                 0033000 ;
                 0034000 ; TESTING, INTEGER EXTRACTION
                 0035000          ENTRY  FCMP,FINT
                 0036000 ;
                 0037000 ; FORMAT CONVERSION
                 0038000          ENTRY  FLOTA,FLOTP FLOTD,FLOT
                 0039000          ENTRY  FIXT,FIXR
                 0040000 ;
                 0041000 ; **********************
                 0042000 ; * MACRO DEFINITIONS *
                 0043000 ; **********************
                 0044000 ;
                 0045000 ; CHANGE SIGN OF REAL OPERAND B OR D
                 0046000 FSIGN    MACRO  REG
                 0047000 BC.L     EQU    B
                 0048000 DE.M     EQU    D
                 0049000          IF     REG*(REG-D)
                 0050000          ERROR  "R"
                 0051000          ENDIF
                 0052000          MOV    A,REG
                 0053000          XRI    80H
                 0054000          MOV    REG,A
                 0055000          ENDM
                 0056000 ;
                 0057000 ; LOAD (& POP) REAL (REG) FROM TOS, STORED BY SFTS MACRO
                 0058000 LFTS     MACRO  REG
                 0059000 BC.L     EQU    B
                 0060000 DE.M     EQU    D
                 0061000          POP    REG
                 0062000          MOV    S-REG/2,REG
                 0063000          POP    REG
                 0064000          IF     REG*(REG-D)
                 0065000          ERROR  "R"
```

```
0066000          ENDIF
0067000          ENDM
0068000 ;
0069000 ; LOAD (REG) FROM TOS & LEAVE ON STACK   (REG)
0070000 LTOS     MACRO  REG
0071000          POP    REG
0072000          PUSH   REG
0073000          ENDM
0074000 ;
0075000 ; MOVE OPERATION ON REGISTER PAIR B,D, OR H
0076000 MOVD     MACRO  R1,R2
0077000          IF     ((R1-D)*(R1-H)*R1) OR ((R2-D)*(R2-H)*R2)
0078000          ERROR  "R"
0079000          ENDIF
0080000          MOV    R1,R2
0081000          MOV    R1+1,R2+1
0082000          ENDM
0083000 ;
0084000 ; MOVE FLOATING OPERAND FROM REG1 TO REG2
0085000 MOVF     MACRO  R2,R1
0086000 BC.L     EQU    B
0087000 DE.H     EQU    D
0088000          IF     (R1*(R1-D)) OR (R2*(R2-D))
0089000          ERROR  "R"
0090000          ENDIF
0091000          MOV    R2,R1
0092000          MOV    R2+1,R1+1
0093000          MOV    5-R2/2,5-R1/2
0094000          ENDM
0095000 ;
0096000 ; RELOAD (BC.L), STORED BY PUSH B, PUSH H SEQUENCE
0097000 RELOAD   MACRO  REG
0098000 BC.L     EQU    B
0099000          IF     REG-B
0100000          ERROR  'R'
0101000          DB     0,0,0
0102000          ENDIF
0103000          IFZ    REG-B
0104000          POP    B
0105000          MOV    L,C
0106000          POP    B
0107000          ENDIF
0108000          ENDM
0109000 ;
0110000 ; "RETURN" AND CHECK STACK LEVEL ZERO
0111000 RTN      MACRO
0112000          IF     .LVL
0113000          ERROR  "0"+.LVL
0114000 .LVL     SET    0
0115000          ENDIF
0116000          RET
0117000          ENDM
0118000 ;
0119000 ; SAVE (BC.L), TO BE RESTORED BY RELOAD BC.L LATER
0120000 SAVE     MACRO  REG
0121000 BC.L     EQU    B
0122000          IF     REG-B
0123000          ERROR  'R'
0124000          DB     0,0
0125000          ENDIF
0126000          IFZ    REG-B
0127000          PUSH   B
0128000          PUSH   H
0129000          ENDIF
0130000          ENDM
0131000 ;
0132000 ; STORE REAL VALUE ON TOP OF STACK; NOTE SFTS B AFFECTS (A)
0133000 SFTS     MACRO  R
0134000 BC.L     EQU    B
0135000 DE.H     EQU    D
0136000          IF     R*(R-D)
0137000          ERROR  "R"
0138000          ENDIF
0139000          IFZ    R
0140000          MOV    A,L
0141000          ENDIF
0142000          PUSH   R
0143000          PUSH   PSW-R
0144000          ENDM
0145000 ;
0146000 ; ********* AUXILIARY ROUTINES **************
0147000 ;
0148000 ; NORMALIZE THE 32 BIT VALUE IN (DEHL) LEFT
0149000 ; AND ROUND TO 16 BITS. DISCARD THE HIGH ORDER
0150000 ; BIT. (B) RETURNS SHIFT COUNT (IN OFFSET BINARY)
0151000 ; RETURN CARRY FOR VALUE ZERO
0152000 ; A,F,B,D,E,H,L
0000 0680        0153000 NLRD:    MVI    B,80H
0002 7A          0154000          MOV    A,D
0003 B7          0155000          ORA    A
0004 FA1400      0156000          JM     NLRD2;    NORMALIZED
0007 B3          0157000          ORA    E
0008 B4          0158000          ORA    H
0009 B5          0159000          ORA    L
000A 37          0160000          STC
000B C8          0161000          RZ;               ZERO VALUE
000C 05          0162000 NLRD1:   DCR    B
000D CD0000      0163000          CALL   DHLZ;     LEFT SHIFT
0010 B7          0164000          ORA    A
0011 F20C00      0165000          JP     NLRD1
0014 E67F        0166000 NLRD2:   ANI    7FH
0016 57          0167000          MOV    D,A;      DISCARD HIGH ORDER BIT
0017 7C          0168000          MOV    A,H
0018 B7          0169000          ORA    A;        CHECK FOR ROUNDING
0019 F0          0170000          RP;               NOT NEEDED
001A 1C          0171000          INR    E;        ROUND UP
001B C0          0172000          RNZ;              NO CARRY
001C 14          0173000          INR    D
001D F0          0174000          RP;               NO OVERFLOW
001E 04          0175000          INR    B;        MODIFY SHIFT COUNT
001F 53          0176000          MOV    D,E;      SET RESULT
0020 C9          0177000          RTN
                 0178000 ;
                 0179000 ; SET THE HIGH ORDER BITS IN (BC) AND (DE) FOR
                 0180000 ; ARITHMETIC OPERATIONS. DISCARD ORIGINAL SIGNS
                 0181000 ; RESET CARRY
                 0182000 ; A,F,B,D
0021 78          0183000 SETHI:   MOV    A,B
0022 F680        0184000          ORI    80H
0024 47          0185000          MOV    B,A
0025 7A          0186000          MOV    A,D
0026 F680        0187000          ORI    80H
0028 57          0188000          MOV    D,A
0029 C9          0189000          RTN
                 0190000 ;
                 0191000 ; EXCHANGE FLOATING OPERANDS
                 0192000 ; A,B(=>D,C(=>E,H(=>L
002A 78          0193000 FXCHG:   MOV    A,B
002B 42          0194000          MOV    B,D
002C 57          0195000          MOV    D,A
002D 79          0196000          MOV    A,C
002E 4B          0197000          MOV    C,E
002F 5F          0198000          MOV    E,A
0030 7D          0199000          MOV    A,L
0031 6C          0200000          MOV    L,H
0032 67          0201000          MOV    H,A
0033 C9          0202000          RTN
                 0203000 ;
                 0204000 ; LOAD (DE.H) FROM STACK LEVEL (A)
                 0205000 ; VALUE WAS STORED WITH PUSH D, PUSH H SEQUENCE
                 0206000 ; A,F,D,E,H
0034 E5          0207000 LFDS:    PUSH   H
0035 CD0000      0208000          CALL   STADR;    GET ABS. ADDRESS
0038 23          0209000          INX    H
0039 7E          0210000          MOV    A,M;      GET EXPONENT
003A 23          0211000          INX    H
003B 5E          0212000          MOV    E,M
003C 23          0213000          INX    H
003D 56          0214000          MOV    D,M;      GET MANTISSA
003E E1          0215000          POP    H
003F 67          0216000          MOV    H,A
0040 C9          0217000          RTN
```

```
                        0218000 ;
                        0219000 ; LOAD (BC.L) FROM STACK LEVEL (A)
                        0220000 ; VALUE WAS STORED BY PUSH D, PUSH H SEQUENCE
                        0221000 ; A,F,B,C,L
0041 E5                 0222000 LFBS:   PUSH    H
0042 CD0000             0223000         CALL    STADR;      GET ABS. ADDRESS
0045 23                 0224000         INX     H
0046 7E                 0225000         MOV     A,M;        GET EXPONENT
0047 23                 0226000         INX     H
0048 4E                 0227000         MOV     C,M;        GET MANTISSA
0049 23                 0228000         INX     H
004A 46                 0229000         MOV     B,M
004B E1                 0230000         POP     H
004C 6F                 0231000         MOV     L,A
004D C9                 0232000         RTN
                        0233000 ;
                        0234000 ; LOAD INDIRECT (BC.L) VIA STACK LEVEL (A) PTR.
                        0235000 ; A,F,B,C,L
004E E5                 0236000 LFBIS:  PUSH    H
004F CD0000             0237000         CALL    STADR;      FORM ABS. ADDRESS
0052 7E                 0238000         MOV     A,M;        LS MEM. ADDRESS
0053 23                 0239000         INX     H
0054 66                 0240000         MOV     H,M;        MS MEM. ADDRESS
0055 6F                 0241000         MOV     L,A
0056 7E                 0242000         MOV     A,M;        GET EXPONENT
0057 23                 0243000         INX     H
0058 4E                 0244000         MOV     C,M
0059 23                 0245000         INX     H
005A 46                 0246000         MOV     B,M;        GET MANTISSA
005B E1                 0247000         POP     H
005C 6F                 0248000         MOV     L,A
005D C9                 0249000         RTN
                        0250000 ;
                        0251000 ; LOAD INDIRECT (DE.H) VIA STACK LEVEL (A) PTR.
                        0252000 ; A,F,D,E,H
005E E5                 0253000 LFDIS:  PUSH    H
005F CD0000             0254000         CALL    STADR;      FORM ABS. ADDRESS
0062 7E                 0255000         MOV     A,M;        LS MEM. ADDRESS
0063 23                 0256000         INX     H
0064 66                 0257000         MOV     H,M;        MS MEM. ADDRESS
0065 6F                 0258000         MOV     L,A
0066 7E                 0259000         MOV     A,M;        GET EXPONENT
0067 23                 0260000         INX     H
0068 5E                 0261000         MOV     E,M
0069 23                 0262000         INX     H
006A 56                 0263000         MOV     D,M;        GET MANTISSA
006B E1                 0264000         POP     H
006C 67                 0265000         MOV     H,A
006D C9                 0266000         RTN
                        0267000 ;
                        0268000 ; STORE (DE.H) INDIRECT VIA STACK LEVEL (A) PTR.
                        0269000 ; A,F
006E E5                 0270000 SFDIS:  PUSH    H
006F C5                 0271000         PUSH    B
0070 4C                 0272000         MOV     C,H;        KEEP EXPONENT
0071 3C                 0273000         INR     A;          ALLOW FOR PUSH B
0072 CD0000             0274000         CALL    STADR;      GET ABS. ADDRESS.
0075 7E                 0275000         MOV     A,M
0076 23                 0276000         INX     H
0077 66                 0277000         MOV     H,M
0078 6F                 0278000         MOV     L,A;        GET POINTER
0079 71                 0279000         MOV     M,C;        STORE EXPONENT
007A 23                 0280000         INX     H
007B 73                 0281000         MOV     M,E
007C 23                 0282000         INX     H
007D 72                 0283000         MOV     M,D;        STORE MANTISSA
007E C1                 0284000         POP     B
007F E1                 0285000         POP     H
0080 C9                 0286000         RTN
                        0287000 ;
                        0288000 ; LOAD (DE.H) VIA POINTER (BC); ADVANCE (BC)
                        0289000 ; B,C,D,E,H
0081 F5                 0290000 FLOAD:  PUSH    PSW
0082 0A                 0291000         LDAX    B
0083 67                 0292000         MOV     H,A
0084 03                 0293000         INX     B
0085 0A                 0294000         LDAX    B
0086 5F                 0295000         MOV     E,A;        MANTISSA
0087 03                 0296000         INX     B
0088 0A                 0297000         LDAX    B
0089 57                 0298000         MOV     D,A;        EXPONENT
008A 03                 0299000         INX     B;          SETUP FOR NEXT TIME
008B F1                 0300000         POP     PSW
008C C9                 0301000         RTN
                        0302000 ;
                        0303000 ; STORE (DE.H) VIA POINTER (BC); ADVANCE (BC)
                        0304000 ; B,C
008D F5                 0305000 FSTOR:  PUSH    PSW
008E 7C                 0306000         MOV     A,H
008F 02                 0307000         STAX    B
0090 03                 0308000         INX     B
0091 7B                 0309000         MOV     A,E
0092 02                 0310000         STAX    B
0093 03                 0311000         INX     B
0094 7A                 0312000         MOV     A,D
0095 02                 0313000         STAX    B
0096 03                 0314000         INX     B;          SETUP FOR NEXT TIME
0097 F1                 0315000         POP     PSW
0098 C9                 0316000         RTN
                        0317000 ;
                        0318000 ; FLOATING POINT ARITHMETIC SYSTEM FOR YALE 8080
                        0319000 ; BASED COMPUTERS  -  BY C.B. FALCONER, APR  1976
                        0320000 ;
                        0321000 ; REAL REPRESENTATION CAN EXPRESS VALUES IN THE
                        0322000 ; ABSOLUTE VALUE RANGE 0.29388 * 10^-38 THROUGH
                        0323000 ; 1.7018 * 10^38, AND ZERO, TOGETHER WITH SIGN,
                        0324000 ; WITH APPROXIMATELY 4.8 DECIMAL DIGIT ACCURACY
                        0325000 ; THE RESOLUTION OF A VALUE BETWEEN 1 AND 2 IS
                        0326000 ; APPROXIMATELY 0.00003.  THE SYSTEM IS DESIGNED
                        0327000 ; TO MAXIMIZE REGISTER, (AS OPPOSED TO MEMORY)
                        0328000 ; USE DURING COMPUTATION.
                        0329000 ;
                        0330000 ; A REAL (FLOATING POINT) VALUE IS REPRESENTED BY A
                        0331000 ; UNIPOLAR 16 BIT MANTISSA, WHOSE VALUE IS IN THE
                        0332000 ; RANGE 1.0 > MANTISSA > -1.0. THE MANTISSA ABSOLUTE
                        0333000 ; VALUE IS ALWAYS >= 0.5.  THUS THE HIGH ORDER
                        0334000 ; BIT OF THE MANTISSA IS ALWAYS A "ONE", AND IS
                        0335000 ; REPLACED BY A SIGN BIT IN INTERNAL REPRESENTATION.
                        0336000 ; A "ONE" SIGN BIT REPRESENTS NEGATIVE VALUES.
                        0337000 ;
                        0338000 ; THE BINARY POINT, RELATIVE TO THE LEFT OF THE
                        0339000 ; LEFTMOST BIT OF THE MANTISSA, IS REPRESENTED BY
                        0340000 ; AN 8 BIT OFFSET INTEGER. A VALUE OF 80 [HEX]
                        0341000 ; REPRESENTS ZERO; 0FF [HEX] REPRESENTS +127;
                        0342000 ; 01 [HEX] REPRESENTS -127; WHILE THE UNIQUE CODE 00
                        0343000 ; INDICATES A ZERO VALUE OF THE MANTISSA. POSITIVE
                        0344000 ; VALUES SPECIFY RIGHTWARDS MOVEMENT OF THE BINARY
                        0345000 ; POINT, ETC.
                        0346000 ;
                        0347000 ; REAL VALUES ARE STORED IN 3 ADJACENT MEMORY BYTES:
                        0348000 ;   LOWEST ADDRESS    EXPONENT
                        0349000 ;      NEXT     "       LEAST SIG. BYTE OF MANTISSA
                        0350000 ;   HIGHEST     "       MOST SIG. BYTE OF MANTISSA
                        0351000 ;
                        0352000 ; REAL OPERANDS CAN APPEAR IN EITHER OF TWO 8080
                        0353000 ; INTERNAL REGISTER CONFIGURATIONS.  THE NORMAL
                        0354000 ; POSITION (CONSIDERED THE REAL ACCUMULATOR) IS
                        0355000 ; THE DE.H REGISTER, IN WHICH THE D & E REGISTERS
                        0356000 ; HOLD THE MANTISSA (SIGN BIT IN D), AND THE H
                        0357000 ; REGISTER HOLDS THE EXPONENT. A SECOND OPERAND
                        0358000 ; MAY BE HELD IN THE BC.L REGISTER, WHERE THE
                        0359000 ; B & C REGISTERS HOLD THE MANTISSA, AND THE L
                        0360000 ; REGISTER HOLDS THE EXPONENT.
                        0361000 ;
                        0362000 ; NOTE THE STORAGE & LOAD MACROS SFTS AND LFTS
                        0363000 ; ABOVE FOR STACKING & UNSTACKING FLOATING VALUES.
                        0364000 ; ALSO NOTE THAT "SFTS B" WILL DISTURB THE A & F
                        0365000 ; REGISTERS, WHILE "SFTS D" WILL NOT.
                        0366000 ;
                        0367000 ; THE SAVE AND RELOAD MACROS ABOVE DO NOT USE THE
                        0368000 ; STANDARD MEMORY FORMAT, AND OPERATE ONLY ON THE
                        0369000 ; BC.L INTERNAL REGISTER GROUP.
```

```
                        0370000 ;
                        0371000 ; ****************************************
                        0372000 ; * CODE FOR THE ARITHMETIC SYSTEM PROPER *
                        0373000 ; ****************************************
                        0374000 ;
                        0375000 ; FLOATING MULTIPLY BY 10; (DE.H):=10*(DE.H)
                        0376000 ; CARRY FOR OVERFLOW, RETURNS MAX VALUE
                        0377000 ; A,F,D,E,H
0099 C5E5               0378000 FMULT:  SAVE    BC.L
009B 010020             0379000         LXI     B,2000H;  10.0
009E 2E84               0380000         MVI     L,84H
00A0 CDB500             0381000         CALL    FMUL
00A3 C169C1             0382000         RELOAD  BC.L
00A6 C9                 0383000         RTN
                        0384000 ;
                        0385000 ; FLOATING DIV BY 10; (DE.H):=(DE.H)*0.10000
                        0386000 ; CARRY FOR UNDERFLOW, RETURNS ZERO
                        0387000 ; A,F,D,E,H
00A7 C5E5               0388000 FDIVT:  SAVE    BC.L
00A9 01CD4C             0389000         LXI     B,4CCDH;  0.10000
00AC 2E7D               0390000         MVI     L,80H-3
00AE CDB500             0391000         CALL    FMUL
00B1 C169C1             0392000         RELOAD  BC.L
00B4 C9                 0393000         RTN
                        0394000 ;
                        0395000 ; FLOATING MULTIPLY (DE.H):=(DE.H)*(BC.L)
                        0396000 ; CARRY FOR OVERFLOW OR UNDERFLOW, WHEN
                        0397000 ; MAXIMUM OR ZERO VALUES ARE RETURNED.
                        0398000 ; A,F,D,E,H
00B5 7C                 0399000 FMUL:   MOV     A,H
00B6 B7                 0400000         ORA     A
00B7 C8                 0401000         RZ;                     ACC. ZERO, RETURN SAME
00B8 7D                 0402000         MOV     A,L
00B9 B7                 0403000         ORA     A
00BA C2BF00             0404000         JNZ     FMUL1;          (BC.L) NOT ZERO
00BD 65                 0405000         MOV     H,L;            (BC.L) ZERO, RETURN ZERO
00BE C9                 0406000         RTN
00BF 7A                 0407000 FMUL1:  MOV     A,D
00C0 A8                 0408000         XRA     B;              FORM RESULT SIGN
00C1 C5                 0409000         PUSH    B
00C2 E5                 0410000         PUSH    H;              SAVE (BC.L)
00C3 F5                 0411000         PUSH    PSW;            SAVE RESULT SIGN
00C4 CD2100             0412000         CALL    SETHI;          SET HI ORDER OPERAND BITS
00C7 CD0000             0413000         CALL    IMUL;           PERFORM MULTIPLICATION
00CA CD0000             0414000         CALL    NLRD;           NORMALIZE & ROUND
00CD F1                 0415000         POP     PSW
00CE E680               0416000         ANI     80H;            RESULT SIGN
00D0 B2                 0417000         ORA     D
00D1 57                 0418000         MOV     D,A;            SET RESULT SIGN
00D2 78                 0419000         MOV     A,B;            SHIFT COUNT
00D3 E1                 0420000         POP     H;              ORIGINAL EXPONENTS
00D4 C1                 0421000         POP     B;              ORIGINAL BC
                        0422000 ;
                        0423000 ; ADD EXPONENTS H:=H+L+A; ALL IN OFFSET CODE
                        0424000 ; CARRY FOR OVERFLOW, WHEN SET EXTREMES IN (DE.H)
                        0425000 ; A,F,H (DE)
00D5 84                 0426000 ADDX:   ADD     H
00D6 F5                 0427000         PUSH    PSW;            SAVE CARRY
00D7 85                 0428000         ADD     L
00D8 67                 0429000         MOV     H,A;            RESULT
00D9 DAE200             0430000         JC      ADDX1;          ONE OVERFLOW REQUIRED
0001                    0431000 @01     SET      LVL
00DC F1                 0432000         POP     PSW
00DD 3F                 0433000         CMC
00DE D0                 0434000         RNC;                    IN RANGE
00DF 2600               0435000         MVI     H,0;            UNDERFLOW
00E1 C9                 0436000         RTN
0001                    0437000  LVL    SET     @01
00E2 F1                 0438000 ADDX1:  POP     PSW;            HAD 1ST CARRY
00E3 D0                 0439000         RNC;                    IN RANGE
                        0440000 ;
                        0441000 ; SET MAX VALUE FOR EXPONENT OVERFLOW
                        0442000 ; A,F,D,E,H
00E4 26FF               0443000 OVEX:   MVI     H,0FFH;         OVERFLOW, SET MAX
00E6 5C                 0444000         MOV     E,H;            AND MANTISSA
00E7 7A                 0445000         MOV     A,D
00E8 F67F               0446000         ORI     7FH;            PRESERVE RESULT SIGN
00EA 57                 0447000         MOV     D,A
00EB 37                 0448000         STC;                    MARK OVERFLOW
00EC C9                 0449000         RTN
                        0450000 ;
                        0451000 ; FLOATING DIVIDE (DE.H):=(DE.H)/(BC.L)
                        0452000 ; CARRY FOR OVERFLOW OR UNDERFLOW WHEN
                        0453000 ; MAXIMUM OR ZERO VALUES ARE RETURNED.
                        0454000 ; DIVISION BY ZERO CAUSES A SYSTEM TRAP
                        0455000 ; A,F,D,E,H
00ED 7D                 0456000 FDIV:   MOV     A,L
00EE B7                 0457000         ORA     A
00EF CC0000             0458000         CZ      AERC;           DIVISION BY ZERO, FATAL
00F2 D8                 0459000         RC
00F3 7C                 0460000 FDIV1:  MOV     A,H
00F4 B7                 0461000         ORA     A
00F5 C8                 0462000         RZ;                     0/NON-ZERO=0
00F6 7A                 0463000         MOV     A,D
00F7 A8                 0464000         XRA     B;              FORM RESULT SIGN
00F8 C5                 0465000         PUSH    B
00F9 E5                 0466000         PUSH    H
00FA F5                 0467000         PUSH    PSW
00FB CD2100             0468000         CALL    SETHI
00FE CD0000             0469000         CALL    DERC;           EXTEND & POSITION DIVIDEND
0101 2E00               0470000         MVI     L,0
0103 7D                 0471000         MOV     A,L
0104 1F                 0472000         RAR;                    LAST BIT
0105 67                 0473000         MOV     H,A
0106 CD0000             0474000         CALL    IDIV;           RETURNS 15 OR 16 BITS
0109 D5                 0475000         PUSH    D;              SAVE QUOTIENT
010A 110000             0476000         LXI     D,0
010D CD0000             0477000         CALL    C2BC
0110 3EFE               0478000         MVI     A,-2;           NEED 2 MORE BITS FOR ROUNDING
0112 F5                 0479000 FDIV2:  PUSH    PSW;            SAVE ITERATIONS COUNT
0113 29                 0480000         DAD     H;              LEFT SHIFT (HLDE)
0114 1F                 0481000         RAR;                    SAVE CARRY OUT
0115 EB                 0482000         XCHG
0116 29                 0483000         DAD     H
0117 EB                 0484000         XCHG
0118 D21C01             0485000         JNC     FDIV3;          NO CARRY INTO L
011B 23                 0486000         INX     H
011C 17                 0487000 FDIV3:  RAL;                    REGAIN CARRY FROM H
011D DA2701             0488000         JC      FDIV4;          YES, GENERATE QUOTIENT BIT
0120 7D                 0489000         MOV     A,L
0121 81                 0490000         ADD     C;              TEST FOR QUOTIENT BIT
0122 7C                 0491000         MOV     A,H
0123 88                 0492000         ADC     B
0124 D22901             0493000         JNC     FDIV5;          NO BIT
0127 09                 0494000 FDIV4:  DAD     B;              SUBTRACT
0128 13                 0495000         INX     D;              INSERT QUOTIENT BIT
0129 F1                 0496000 FDIV5:  POP     PSW;            GET ITERATION COUNT
012A 3C                 0497000         INR     A
012B FA1201             0498000         JM      FDIV2;          NOT DONE
012E 7B                 0499000         MOV     A,E
012F 0F                 0500000         RRC
0130 0F                 0501000         RRC
0131 67                 0502000         MOV     H,A;            EXTEND QUOTIENT
0132 D1                 0503000         POP     D;              RESTORE QUOTIENT
0133 CD0000             0504000         CALL    NLRD;           NORMALIZE & ROUND
0136 04                 0505000         INR     B;              CORRECT BIN. POINT
0137 F1                 0506000         POP     PSW
0138 E1E5               0507000         LTOS    H;              ORIGINAL EXPONENTS
013A E680               0508000         ANI     80H
013C B2                 0509000         ORA     D
013D 57                 0510000         MOV     D,A;            FORM RESULT SIGN
013E 7D                 0511000         MOV     A,L
013F 2F                 0512000         CMA
0140 3C                 0513000         INR     A;              COMPLEMENT DIVISOR EXPONENT
0141 68                 0514000         MOV     L,B;            SHIFT COUNT
0142 CDD500             0515000         CALL    ADDX;           FORM RESULT EXPONENT
0145 7C                 0516000         MOV     A,H
0146 E1                 0517000         POP     H;              ORIGINAL EXPONENTS
0147 67                 0518000         MOV     H,A
0148 C1                 0519000         POP     B
0149 C9                 0520000         RTN;                    WITH ANY ADDX CARRY
                        0521000
```

```
                        0522000 ; FLOATING REV. DIV. (DE.H)!=(BC.L)/(DE.H)
                        0523000 ; CARRY FOR OVERFLOW OR UNDERFLOW WHEN
                        0524000 ; MAXIMUM OR ZERO VALUES ARE RETURNED.
                        0525000 ; DIVISION BY ZERO CAUSES A SYSTEM TRAP
                        0526000 ; A,F,D,E,H
014A C5E5               0527000 FDIVR:  SAVE    BC.L
014C CD2A00             0528000         CALL    FXCHG
014F CDED00             0529000         CALL    FDIV
0152 C169C1             0530000         RELOAD BC.L
0155 C9                 0531000         RTN
                        0532000 ;
                        0533000 ; FLOATING RECIPROCAL. (DE.H)!=1.0/(DE.H)
                        0534000 ; DIVISION BY ZERO (ORIG. (DE.H) CAUSES SYSTEM TRAP
                        0535000 ; A,F,D,E,H
0156 C5E5               0536000 FRCIP:  SAVE    BC.L
0158 424B6C             0537000         MOVF    B,D
015B 110000             0538000         LXI     D,0
015E 2681               0539000         MVI     H,81H;      FLOATING 1.0
0160 CDED00             0540000         CALL    FDIV
0163 C169C1             0541000         RELOAD BC.L
0166 C9                 0542000         RTN
                        0543000 ;
                        0544000 ; ALIGN OPERANDS FOR FADD
                        0545000 ; RETURNS TWO 24 BIT VALUES IN (BCL) AND (DEH)
                        0546000 ; WITH BINARY POINTS ALIGNED.  THE ACTUAL BINARY
                        0547000 ; POINT IS THAT OF THE LARGER (ON INPUT) MAGNITUDE
                        0548000 ; PLUS 1; I.E. RIGHT SHIFTED ONE PLACE  THIS ALLOWS
                        0549000 ; SPACE FOR OVERFLOWS ON ADDITION
                        0550000 ; A,F,B,C,D,E,H,L
0167 7C                 0551000 ALIN:   MOV     A,H
0168 95                 0552000         SUB     L
0169 B7                 0553000         ORA     A;          RESET ANY CARRY
016A F5                 0554000         PUSH    PSW;        RELATIVE MAGNITUDES
016B 78                 0555000         MOV     A,B;        BCL := (BC OR 8000H) SHR 1
016C F680               0556000         ORI     80H
016E 1F                 0557000         RAR
016F 47                 0558000         MOV     B,A
0170 79                 0559000         MOV     A,C
0171 1F                 0560000         RAR
0172 4F                 0561000         MOV     C,A
0173 3E00               0562000         MVI     A,0
0175 1F                 0563000         RAR
0176 6F                 0564000         MOV     L,A
0177 7A                 0565000         MOV     A,D;        DEH := (DE OR 8000H) SHR 1
0178 F680               0566000         ORI     80H
017A 1F                 0567000         RAR
017B 57                 0568000         MOV     D,A
017C 7B                 0569000         MOV     A,E
017D 1F                 0570000         RAR
017E 5F                 0571000         MOV     E,A
017F 3E00               0572000         MVI     A,0
0181 1F                 0573000         RAR
0182 67                 0574000         MOV     H,A
0183 F1                 0575000 ALIN1:  POP     PSW
0184 C8                 0576000         RZ;                 ALIGNED
0185 F29601             0577000         JP      ALIN2;      DE MAG > BC MAG
0188 3C                 0578000         INR     A;          BC MAG > DE MAG
0189 F5                 0579000         PUSH    PSW;        SAVE REL MAG
018A 7A                 0580000         MOV     A,D;        SHIFT DEH RIGHT, 0 IN
018B 1F                 0581000         RAR
018C 57                 0582000         MOV     D,A
018D 7B                 0583000         MOV     A,E
018E 1F                 0584000         RAR
018F 5F                 0585000         MOV     E,A
0190 7C                 0586000         MOV     A,H
0191 1F                 0587000         RAR
0192 67                 0588000         MOV     H,A
0193 C38301             0589000         JMP     ALIN1;      NOW TEST
0000                    0590000  LVL    SET      LVL-1
0196 3D                 0591000 ALIN2:  DCR     A
0197 F5                 0592000         PUSH    PSW
0198 78                 0593000         MOV     A,B;        SHIFT BCL RIGHT, 0 IN
0199 1F                 0594000         RAR
019A 47                 0595000         MOV     B,A
019B 79                 0596000         MOV     A,C
019C 1F                 0597000         RAR
019D 4F                 0598000         MOV     C,A
019E 7D                 0599000         MOV     A,L
019F 1F                 0600000         RAR
01A0 6F                 0601000         MOV     L,A
01A1 C38301             0602000         JMP     ALIN1
0000                    0603000 .LVL    SET     .LVL-1
                        0604000 ;
                        0605000 ; FLOATING REV. SUBTRACT (DE.H):=(BC.L)-(DE.H)
                        0606000 ; CARRY FOR OVER/UNDERFLOW, SETS EXTREME VALUE
                        0607000 ; A,F,D,E,H
01A4 7AEE8057           0608000 FSUBR:  FSIGN   D;          CHANGE D SIGN
                        0609000 ;
                        0610000 ; FLOATING ADD (DE.H):=(DE.H)+(BC.L)
                        0611000 ; CARRY FOR OVER/UNDERFLOW, SETS EXTREME VALUE
                        0612000 ; A,F,D,E,H
01A8 7D                 0613000 FADD:   MOV     A,L
01A9 B7                 0614000         ORA     A
01AA C8                 0615000         RZ;                 BC.L=0
01AB 7C                 0616000         MOV     A,H
01AC B7                 0617000         ORA     A
01AD C2B401             0618000         JNZ     FADD2;      DE.H <> 0
01B0 505965             0619000 FADD1:  MOVF    D,B;        DE.H << BC.L
01B3 C9                 0620000         RTN
01B4 95                 0621000 FADD2:  SUB     L
01B5 DABF01             0622000         JC      FADD3;      BC MAG. > DE MAG.
01B8 FE11               0623000         CPI     16+1
01BA D0                 0624000         RNC;                BC.L << DE.H
01BB 7C                 0625000         MOV     A,H;        WILL BE RESULT MAGNITUDE
01BC C3C601             0626000         JMP     FADD4
01BF FEF0               0627000 FADD3:  CPI     -16
01C1 3F                 0628000         CMC
01C2 D2B001             0629000         JNC     FADD1;      DE.H << BC.L
01C5 7D                 0630000         MOV     A,L;        WILL BE RESULT MAGNITUDE
01C6 C5E5               0631000 FADD4:  SAVE    BC.L
01C8 F5                 0632000         PUSH    PSW;        SAVE RESULT MAGNITUDE
01C9 78                 0633000         MOV     A,B
01CA AA                 0634000         XRA     D
01CB 78                 0635000         MOV     A,B
01CC F2F101             0636000         JP      FADD5;      SIGNS SAME
0003                    0637000 @01     SET     .LVL
01CF A0                 0638000         ANA     B;          SIGNS DIFFERENT
01D0 F42A00             0639000         CP      FXCHG;      DE.H NEG, BC.L POS
01D3 CD6701             0640000         CALL    ALIN;       NOW DE.H POS & BC.L NEG
01D6 7C                 0641000         MOV     A,H
01D7 95                 0642000         SUB     L
01D8 67                 0643000         MOV     H,A
01D9 7B                 0644000         MOV     A,E
01DA 99                 0645000         SBB     C;          PERFORM SUBTRACTION
01DB 5F                 0646000         MOV     E,A
01DC 7A                 0647000         MOV     A,D
01DD 98                 0648000         SBB     B
01DE 57                 0649000         MOV     D,A
01DF F5                 0650000         PUSH    PSW;        SAVE RESULT SIGN
01E0 F2FE01             0651000         JP      FADD6;      NO COMPLEMENT NEEDED
01E3 CD0000             0652000         CALL    CIDE
01E6 7C                 0653000         MOV     A,H
01E7 2F                 0654000         CMA
01E8 3C                 0655000         INR     A
01E9 67                 0656000         MOV     H,A
01EA C2FE01             0657000         JNZ     FADD6;      NO PROPAGATION
01ED 13                 0658000         INX     D
01EE C3FE01             0659000         JMP     FADD6;      NOW MAGNITUDE, SIGN IS STACKED
0003                    0660000 .LVL    SET     @01
01F1 F5                 0661000 FADD5:  PUSH    PSW;        RESULT SIGN
01F2 CD6701             0662000         CALL    ALIN
01F5 7C                 0663000         MOV     A,H
01F6 85                 0664000         ADD     L
01F7 67                 0665000         MOV     H,A;        ADD MANTISSAS
01F8 7B                 0666000         MOV     A,E
01F9 89                 0667000         ADC     C
01FA 5F                 0668000         MOV     E,A
01FB 7A                 0669000         MOV     A,D
01FC 88                 0670000         ADC     B
01FD 57                 0671000         MOV     D,A
01FE AF                 0672000 FADD6:  XRA     A
01FF 6F                 0673000         MOV     L,A
```

```
0200 B2          0674000          ORA    D
0201 B3          0675000          ORA    E
0202 B4          0676000          ORA    H
0004             0677000 001      SET     LVL
0203 C20C02      0678000          JNZ    FADD7;      RESULT NOT ZERO
0206 F1          0679000          POP    PSW;        PURGE SIGN
0207 F1          0680000          POP    PSW;        PURGE MAGNITUDE
0208 B7          0681000          ORA    A;          RESET ANY CARRY
0209 C31B02      0682000          JMP    FADD8
0004             0683000  LVL     SET    001
020C CD0000      0684000 FADD7:   CALL   NLRD
020F F1          0685000          POP    PSW
0210 E680        0686000          ANI    80H
0212 B2          0687000          ORA    D
0213 57          0688000          MOV    D,A;        SET RESULT SIGN
0214 60          0689000          MOV    H,B
0215 2E01        0690000          MVI    L,01H
0217 F1          0691000          POP    PSW;        SAVED RESULT MAGNITUDE
0218 CDD500      0692000          CALL   ADDX;       SET RESULT MAGNITUDE
021B C169C1      0693000 FADD8:   RELOAD BC.L
021E C9          0694000          RTN;               WITH ADDX CARRY IF OVERFLOW
                 0695000 ;
                 0696000 ; FLOATING SUBTRACT (DE.H):=(DE.H)-(BC.L)
                 0697000 ; CARRY FOR OVER/UNDERFLOW, SETS EXTREME VALUE
                 0698000 ; A,F,D,E,H
021F C5E5        0699000 FSUB:    SAVE   BC.L
0221 7BEE8047    0700000          FSIGN  B;          CHANGE B SIGN
0225 CDA801      0701000          CALL   FADD
0228 C169C1      0702000          RELOAD BC.L
022B C9          0703000          RTN
                 0704000 ;
                 0705000 ; FLOATING COMPARE.  SET FLAGS FOR (DE.H)-(BC.L)
                 0706000 ;   ZERO FLAG IF EQUAL.
                 0707000 ;   PLUS FLAG IF (DE.H) >= (BC.L)
                 0708000 ;  MINUS FLAG IF (DE.H) <  (BC.L)
                 0709000 ; A,F
022C 7D          0710000 FCMP:    MOV    A,L
022D B7          0711000          ORA    A
022E C23B02      0712000          JNZ    FCMP1;      BC.L <> ZERO
0231 7C          0713000          MOV    A,H
0232 B7          0714000          ORA    A
0233 C8          0715000          RZ;                BOTH ZERO
0234 7A          0716000          MOV    A,D
0235 F601        0717000          ORI    1;          SET FLAGS ACCORDING TO DE.H SIGN
0237 C9          0718000          RTN
0238 7C          0719000 FCMP1:   MOV    A,H
0239 B7          0720000          ORA    A
023A CA4502      0721000          JZ     FCMP2;      (DE.H)=0, FLAGS INVERSE OF
023D 95          0722000          SUB    L                              (BC.L) SIGN
023E CA4A02      0723000          JZ     FCMP4;      MAGNITUDES SAME
0241 7A          0724000          MOV    A,D
0242 F24702      0725000          JP     FCMP3;      DE.H CONTROLLING MAGNITURE
0245 78          0726000 FCMP2:   MOV    A,B;        BC.L CONTROLLING MAGNITUDE
0246 2F          0727000          CMA
0247 F601        0728000 FCMP3:   ORI    1;          SET FLAGS VIA APPROPRIATE
0249 C9          0729000          RTN                                OPERAND SIGN
024A B0          0730000 FCMP4:   ORA    B;          CHECK SIGNS
024B FA5502      0731000          JM     FCMP5;      (BC.L)<0
024E B2          0732000          ORA    D;          (BC.L)>0, CHECK (DE.H)
024F F8          0733000          RM;                (DE.H)<0
0250 7B          0734000          MOV    A,E;        BOTH >= 0
0251 91          0735000          SUB    C
0252 7A          0736000          MOV    A,D
0253 98          0737000          SBB    B
0254 C9          0738000          RTN
0255 7A          0739000 FCMP5:   MOV    A,D
0256 F601        0740000          ORI    1
0258 F0          0741000          RP;                (DE.H)>0 AND (BC.L)<0
0259 79          0742000          MOV    A,C
025A 93          0743000          SUB    E;          BOTH <0
025B 78          0744000          MOV    A,B
025C 9A          0745000          SBB    D
025D C9          0746000          RTN
                 0747000 ;
                 0748000 ; CONVERT SIGNED INTEGER (A) TO REAL (DE.H)
                 0749000 ; A,F,D,E,H
025E B7          0750000 FLOTA:   ORA    A
025F 67          0751000          MOV    H,A
0260 C8          0752000          RZ;                ZERO
0261 57          0753000          MOV    D,A
0262 1E00        0754000          MVI    E,0
0264 E680        0755000          ANI    80H
0266 F5          0756000 FLOT5:   PUSH   PSW;       SAVE SIGN
0267 2688        0757000          MVI    H,88H;     BINARY POINT
0269 7A          0758000          MOV    A,D
026A F29602      0759000          JP     FLOT2
026D 2F          0760000          CMA
026E 3C          0761000          INR    A
026F 57          0762000          MOV    D,A
0270 C39602      0763000          JMP    FLOT2;     -VE INPUT
0000             0764000 .LVL     SET     LVL-1
                 0765000 ;
                 0766000 ; CONVERT POSITIVE INTEGER (A) TO REAL (DE.H)
                 0767000 ; A,F,D,E,H
0273 B7          0768000 FLOTP:   ORA    A
0274 67          0769000          MOV    H,A
0275 C8          0770000          RZ;                ZERO
0276 57          0771000          MOV    D,A
0277 AF          0772000          XRA    A
027B 5F          0773000          MOV    E,A
0279 C36602      0774000          JMP    FLOT5
                 0775000 ;
                 0776000 ; CONVERT POSITIVE INTEGER (DE) TO REAL (DE.H)
                 0777000 ; A,F,D,E,H
027C AF          0778000 FLOTD:   XRA    A
027D 67          0779000          MOV    H,A
027E C38B02      0780000          JMP    FLOT1
                 0781000 ;
                 0782000 ; EXTRACT INTEGER PORTION OF (DE H) IN REAL FORM
                 0783000 ; A,F,D,E,H
0281 CDA902      0784000 FINT:    CALL   FIXT;       CONVERT TO INTEGER
0284 3F          0785000          CMC
0285 D0          0786000          RNC;               ALREADY INTEGRAL
                 0787000 ;
                 0788000 ; CONVERT SIGNED INTEGER (DE) TO REAL (DE.H)
                 0789000 ; A,F,D,E,H
0286 2600        0790000 FLOT:    MVI    H,0
0288 7A          0791000          MOV    A,D
0289 E680        0792000          ANI    80H
028B F5          0793000 FLOT1:   PUSH   PSW;       SAVE SIGN
028C FC0000      0794000          CM     C2DE;      MAGNITUDE
028F 7A          0795000          MOV    A,D
0290 B3          0796000          ORA    E
0291 CA9E02      0797000          JZ     FLOT3;     ZERO VALUE
0294 2690        0798000          MVI    H,90H;     BINARY POINT
0296 7A          0799000 FLOT2:   MOV    A,D
0297 B7          0800000          ORA    A
0298 F2A202      0801000          JP     FLOT4;     FURTHER NORMALIZING
029B E67F        0802000          ANI    7FH
029D 57          0803000          MOV    D,A
029E F1          0804000 FLOT3:   POP    PSW;       GET SIGN
029F B2          0805000          ORA    D
02A0 57          0806000          MOV    D,A
02A1 C9          0807000          RTN
02A2 EB          0808000 FLOT4:   XCHG
02A3 29          0809000          DAD    H;         LEFT SHIFT
02A4 EB          0810000          XCHG
02A5 25          0811000          DCR    H;         ADJUST BINARY POINT
02A6 C39602      0812000          JMP    FLOT2
                 0813000 ;
                 0814000 ; CONVERT REAL (DE.H) TO SIGNED INTEGER (TRUNCATE)
                 0815000 ; (DE):=SIGNED INTEGER RESULT, TRUNCATED
                 0816000 ; CARRY IF NOT 32767)=VALUE)=-32768, UNCONVERTED.
                 0817000 ; A,F,D,E,H
02A9 7C          0818000 FIXT:    MOV    A,H
02AA B7          0819000          ORA    A
02AB C2B202      0820000          JNZ    FIXT2;     NON-ZERO
02AE AF          0821000 FIXT1:   XRA    A
02AF 57          0822000          MOV    D,A;       ZERO INTEGER PART
02B0 5F          0823000          MOV    E,A
02B1 C9          0824000          RTN
02B2 F2AE02      0825000 FIXT2:   JP     FIXT1;     NO INTEGER PART
```

```
02B5 D681       0826000          SUI    81H
02B7 FAAE02     0827000          JM     FIXT1;    NO INTEGER PART
02BA D60F       0828000          SUI    15
02BC C2C802     0829000          JNZ    FIXT3;    MAGNITUDE < 32768
02BF 7A         0830000          MOV    A,D
02C0 D680       0831000          SUI    80H
02C2 37         0832000          STC
02C3 C0         0833000          RNZ;              NOT -32768
02C4 83         0834000          ORA    E
02C5 C8         0835000          RZ;               EXACTLY -32768
02C6 37         0836000          STC
02C7 C9         0837000          RTN;              OVERSIZE
02C8 3F         0838000 FIXT3:   CMC
02C9 D8         0839000          RC;               OVERSIZE
02CA 67         0840000          MOV    H,A;       BINARY POINT 0 FOR 1 TO 2
02CB 7A         0841000          MOV    A,D
02CC F5         0842000          PUSH   PSW
02CD F680       0843000          ORI    80H
02CF 57         0844000          MOV    D,A
02D0 87         0845000 FIXT4:   ORA    A
02D1 CD0000     0846000          CALL   DERC;      RIGHT SHIFT, 0 IN
02D4 24         0847000          INR    H
02D5 FAD002     0848000          JM     FIXT4
02D8 F1         0849000          POP    PSW
02D9 B7         0850000          ORA    A
02DA F0         0851000          RP;               POSITIVE
02DB C30000     0852000          JMP    C2DE;      INSERT SIGN
                0853000 ;
                0854000 ; FIX & ROUND (DE.H) TO SIGNED INTEGER IN (DE)
                0855000 ; RETURN CARRY IF MAG. > 32767, WITHOUT CONVERTING.
                0856000 ; A,F,D,E,H
02DE C5E5       0857000 FIXR:    SAVE   BC.L
02E0 D5E5       0858000          SFTS   D;         SAVE IN CASE OF ERROR
02E2 01FF7F     0859000          LXI    B,07FFFH
02E5 68         0860000          MOV    L,B;       0.49999 TO PREVENT FADD ROUNDUP
02E6 7A         0861000          MOV    A,D
02E7 B7         0862000          ORA    A
02E8 F2F402     0863000          JP     FIXR1;     (DE.H)>0
02EB 7C         0864000          MOV    A,H
02EC B7         0865000          ORA    A
02ED CAF402     0866000          JZ     FIXR1;     (DE.H)=0
02F0 78EE8047   0867000          FSIGN  B
02F4 CDA801     0868000 FIXR1:   CALL   FADD;      ROUND
02F7 CDA902     0869000          CALL   FIXT;       FIX
0004            0870000 @01      SET    .LVL
02FA DA0203     0871000          JC     FIXR2;     OVERFLOW ERROR
02FD C1         0872000          POP    B;         PURGE ORIGINAL ARGUMENT
02FE C1         0873000          POP    B
02FF C30503     0874000          JMP    FIXR3;     RESTORE BC.L
0004            0875000  LVL     SET    @01
0302 D162D1     0876000 FIXR2:   LFTS   D;         RESTORE ARGUMENT
0305 C169C1     0877000 FIXR3:   RELOAD BC.L
0308 C9         0878000          RTN
                0879000 ;
0000            0880000          END    0
PASS 2 ERRORS = 0
.LVL   RED   00000H   @01    RED   00004H   ADDX   IDR   000D5H
ADDX1  IDR   000E2H   AERC   EXT   00002H   ALIN   IDR   00167H
ALIN1  IDR   00183H   ALIN2  IDR   00196H   C1DE   EXT   00003H
C2BC   EXT   00004H   C2DE   EXT   00005H   DERC   EXT   00006H
DHLZ   EXT   00007H   FADD   IDR   001A8H   FADD1  IDR   001B0H
FADD2  IDR   001B4H   FADD3  IDR   001BFH   FADD4  IDR   001C6H
FADD5  IDR   001F1H   FADD6  IDR   001FEH   FADD7  IDR   0020CH
FADD8  IDR   0021BH   FCMP   IDR   00222CH  FCMP1  IDR   00238H
FCMP2  IDR   00245H   FCMP3  IDR   00247H   FCMP4  IDR   0024AH
FCMP5  IDR   00255H   FDIV   IDR   000EDH   FDIV1  IDR   000F3H
FDIV2  IDR   00112H   FDIV3  IDR   0011CH   FDIV4  IDR   00127H
FDIV5  IDR   00129H   FDIVR  IDR   0014AH   FDIVT  IDR   000A7H
FINT   IDR   00281H   FIXR   IDR   002DEH   FIXR1  IDR   002F4H
FIXR2  IDR   00302H   FIXR3  IDR   00305H   FIXT   IDR   002A9H
FIXT1  IDR   002AEH   FIXT2  IDR   002B2H   FIXT3  IDR   002C8H
FIXT4  IDR   002D0H   FLOAD  IDR   00081H   FLOT   IDR   00286H
FLOT1  IDR   00288H   FLOT2  IDR   00296H   FLOT3  IDR   0029EH
FLOT4  IDR   002A2H   FLOT5  IDR   00266H   FLOTA  IDR   0025EH
FLOTD  IDR   0027CH   FLOTP  IDR   00273H   FMUL   IDR   00085H
FMUL1  IDR   000BFH   FMULT  IDR   00099H   FRCIP  IDR   00156H
FSIGN  MACRO         FSTOR  IDR   0008DH   FSUB   IDR   0021FH
FSUBR  IDR   001A4H   FXCHG  IDR   0002AH   IDIV   EXT   00008H
IMUL   EXT   00009H   LFBIS  IDR   0004EH   LFBS   IDR   00041H
LFDIS  IDR   0005EH   LFDS   IDR   00034H   LFTS   MACRO
LTOS   MACRO         MOVD   MACRO         MOVF   MACRO
MLRD   IDR   00000H   MLRD1  IDR   0000CH   MLRD2  IDR   00014H
OVEX   IDR   000E4H   RELOAD MACRO         RTN    MACRO
SAVE   MACRO         SETHI  IDR   00021H   SFDIS  IDR   0006EH
SFTS   MACRO         STADR  EXT   0000AH

##############################################
########    SEGMENT FLTINPUT    #########
##############################################

0001000 ;  COPYRIGHT YALE UNIVERSITY
0002000 ;
0003000 ; BY C.B. FALCONER                    (203) 436-2603
0004000 ;                  RM 6016CB
0005000 ;                  YALE SCHOOL OF MEDICINE
0006000 ;                  789 HOWARD AVENUE
0007000 ;                  NEW HAVEN, CONN 06504
0008000 ;
0009000 ;                        **********************
0010000          RSEG;           *   NUMERIC INPUT    *
0011000 ;                        **********************
0012000 ;
0013000 ; EXTERNAL ROUTINES REQUIRED
0014000          EXTRN  FDIVT,FLOTD,FMULT; IN FLTARITH
0015000          EXTRN  MUL10;     IN INTARITH
0016000 ;
0017000 ; ENTRY POINTS ALLOWED
0018000 ; **********************
0019000 ;
0020000 ; UTILITY ROUTINES
0021000          ENTRY  DEBLK,JBC,QMAX,QNUM
0022000 ;
0023000 ; NUMERIC INPUT
0024000          ENTRY  IVAL,IVALC
0025000 ;
0026000 ; **********************
0027000 ; * MACRO DEFINITIONS *
0028000 ; **********************
0029000 ;
0030000 ; EXECUTE ROUTINE AT (BC) [NORMALLY GET NEXT CHARACTER]
0031000 GETCH    MACRO
0032000          CALL   JBC
0033000          ENDM
0034000 ;
0035000 ; CHANGE SIGN OF REAL OPERAND B OR D
0036000 FSIGN    MACRO  REG
0037000          IF     REG*(REG-D)
0038000          ERROR  "R"
0039000          ENDIF
0040000          MOV    A,REG
0041000          XRI    80H
0042000          MOV    REG,A
0043000          ENDM
0044000 ;
0045000 ; RELOAD (BC.L), STORED BY PUSH B, PUSH H SEQUENCE
0046000 RELOAD   MACRO  REG
0047000 BC.L     EQU    B
0048000          IF     REG-B
0049000          ERROR  'R'
0050000          DB     0,0,0
0051000          ENDIF
0052000          IFZ    REG-B
0053000          POP    B
0054000          MOV    L,C
0055000          POP    B
0056000          ENDIF
0057000          ENDM
0058000 ;
0059000 ; "RETURN" AND CHECK STACK LEVEL ZERO
0060000 RTN      MACRO
0061000          IF     .LVL
0062000          ERROR  "0"+.LVL
```

Continued on pg. 23

CONVPATB

BY PAUL HOLLIDAY
4807 Arlene St.
San Diego, CA 92117

Purpose. The purpose of CONVPATB is to convert a file from Palo Alto Tiny Basic (PATB) format to ASCII format. The reason conversion to ASCII is required is that Tiny Basic files contain intermixed binary and ASCII data which cannot be printed or listed as is. That is, only the original Tiny Basic interpreter can decode the binary data portions of the file. This program converts the first two bytes of every line (binary data) to a four digit ASCII number and adds a line feed after each carriage restore. The new (converted) file can then be printed, listed, or edited just like any other ASCII file.

Language And Minimum System Requirements. The CONVPATB program is written in 8080 assembly language and requires approximately 2K bytes for the object code. A typical minimum system requirement will usually be 16K, but 24K or more is an advantage if large files are to be converted. The actual memory requirement depends on the size of the Tiny Basic program (or file) to be converted. For example, a 24K system can convert a Tiny Basic program of about 7K bytes. In addition, the CP/M operating system and at least one floppy disk is required to use the program as is. (Note: it is possible to use the program with cassette mass storage system if disk I/O routines are replaced by the appropriate cassette read and write routines).

Loading And Using The Program. The program is loaded from disk by typing any of the following after the CP/M prompt:

```
CONVPATB
CONVPATB STARTREK
CONVPATB B:STARTREK
```

If no file name is specified, as in case 1 above, the program will prompt for a file name after it is loaded. Enter the 8 (or less) character name of the file to be converted. If more than 8 characters are entered, only the first 8 characters are used. Also, the read file type is always set at type .TBI, no matter what file type you enter.

If a file name is specified at load time, as in case 2 above, the file will be loaded and converted immediately without operator intervention. The file must be present on the currently logged disk with file type .TBI.

If the name of the file to be converted is preceded by the drive name, as in case 3 above, then the file will be loaded from the specified drive and the converted file will also be written on the same drive. The above example shows the file being loaded and written on drive B.

Input And Output File Formats. The Tiny Basic source file to be converted must be on the disk with file type .TBI. After the file is loaded and converted, the new file is written with file type .ASC. The original file is not changed or modified.

The program to be converted must be in the proper format which is given in figure 1. The format of the converted file is also shown in figure 1. To get the file in the correct form before conversion, you may have to use the DDT function of CP/M, deleting any extra bytes at the beginning, if necessary. Again, this will depend upon the particular version of Tiny Basic you are using. Don't forget to add end sentinel bytes at the end of the source to be converted, using DDT for this. The end sentinel is two consecutive bytes of FF hex (FF FF). See Table 1 for an example of how to prepare a program for conversion.

TABLE 1.

TO MOVE PATB SOURCE TO DISK AND PREPARE IT FOR CONVERSION WITH CONVPATB

1. Put source from cassette tape on disk using any method and make STARTREK.COM file. This is usually done using the monitor load routine, DDT, and SAVE xx filename.COM	
2. Mount CP/M SYSTEM disk in drive A Mount STARTREK.COM disk in drive B	
3. DDT B:STARTREK.COM	To load DDT and .COM file
4. Find end of source using DDT dump	
5. F1EC1,1ECF,FF	Fill last bytes at end with 'FF' end sentinel
6. M0916,1ECF,100	Move source to 100 hex. End of STARTREK is at 1ED0. Locations will vary with the program being moved.
7. Control-C	Exit from DDT and return to CP/M
8. Mount CONVPATB disk in drive A	
9. SAVE 22 TINYTREK.TBI	Saves new file to be converted.
10. Use CONVPATB to convert to ASCII	

Note that the addresses shown will depend upon the particular program to be moved and which Tiny Basic interpreter was used to input the program.

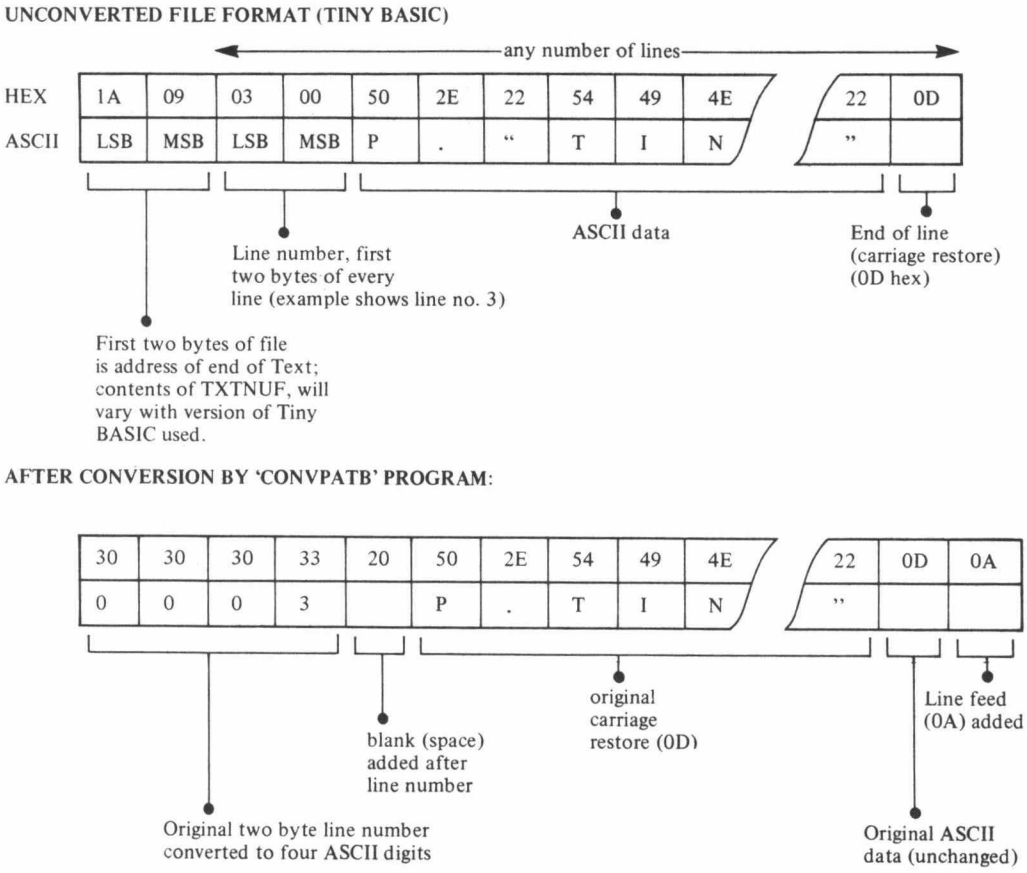


FIGURE 1. TINY BASIC AND CONVPATB DATA FORMATS.

What This Program Does. If the program to be converted is in the proper format, this program will:

1. Delete the first two bytes of the file. (TXTNUF)
2. Convert the first two bytes of each line from a binary line number (lsb msb) to a four digit ASCII line number.
3. Add a space (blank) after the line number.
4. Search for the next carriage restore (0D hex) and add a line feed (0A hex) at the end of each line.
5. Write a new file with file type .ASC.

Note that the last bytes of the text or program to be converted (input file) must be FF FF hex to terminate conversion.

Program Organization. The program is physically ordered as follows (refer to program listing 1):

1. Comment preambles
2. Equate dictionaries
3. Jump table and entry/external vectors
4. Initial start and main program
5. Subroutines in alphabetic order
6. I/O section, primitives; Console input, terminal output; File handling primitives
7. Messages area
8. Constants, data storage, variables
9. File control blocks and related strings
10. Text buffer

The equates are separated into functional groups and all the labels within each group are in alphabetic order where possible. This allows major program modifications for different systems to be made easily and quickly. In addition, the alphabetic order makes labels easy to find and insert. The alphabetic order concept can be found throughout the program and is a tremendous aid in keeping an assembly language program listing orderly and manageable.

Program Description. Subsequent paragraphs describe some of the major program functions. Refer to program listing 1 for the location of the various routines and labels.

Initial program entry is via the jump table at location BEGIN. The jump takes us to START where an initialization routine is called, the screen is cleared, and the sign-on identifier message is displayed. If a file name was specified at load time, it is used to setup the File Control Blocks (FCB's) for subsequent read and write. If a file name was not specified, the operator is prompted to enter the name of the file to be converted. The file name entered is then used to set the FCB's. Next, the user is requested to mount the disk with the program to be converted. When the operator acknowledges, the files are opened and the file to be converted is read into the memory buffer. (Program label reference: MAIN40).

For this version the entire program file is read into memory before any conversion takes place, limiting the size of the Tiny Basic program that can be converted to approximately 7K bytes in a 24K system. In most cases this limit will not pose

any problems since the majority of Tiny Basic programs will usually occupy only 4K to 6K maximum.

After the program is in the buffer, the pointer BUFF2 is set equal to the end of the text just input. BUFF2 points to the start location where the converted program will be stored. The first two bytes of the original file are discarded since they contain an address to the end of text and are not needed (see figure 1).

The main conversion loop begins at the label MOVE10. The first two bytes of each line are converted from binary to a four digit ASCII number and stored in the new file area (BUFF2). An ASCII space is added after the line number for readability. This could be deleted if memory space was a problem. A test is made to see if either the end of text (CHECKEND) or the end of memory (CHECKMEM) is reached. If either case is true, the conversion loop is terminated and control is passed to MOVE60.

When a carriage restore (CR) is encountered within the line conversion loop, an additional line feed (LF) character is added to the new file because most Tiny Basic's use only a single CR at the end of each line.

When conversion is complete (MOVE60), the characters END are added to the new ASCII file along with the end sentinel bytes. The END characters are just a formality to make it easy to identify the end of the file and are not required. The operator is notified with the message "WRITING FILE.." and the subroutine WRITE2 is called to write the entire memory buffer from BUFF2 to the end. Both files are closed, the "FINISHED" message is output, and control is returned to CP/M via the jump at EXIT2.

Programming Notes, Structured Programming. Most of the program follows the general rules of structured programming. For example, the "one entrance, one exit" rule is adhered to wherever possible. In some cases, where there is no chance for confusion, there may be two returns in order to save instructions but they are kept close together and commented. The use of structured programming concepts, while making the program easy to change, test, and add to, also consume a lot more memory space than non-structured counterparts. This is because of the additional 3 byte call and jump instructions required to follow the "rules." In any case, it is definitely worth the additional time, effort and memory for any non-trivial program since it reduces the level of expertise required to understand and modify the program.

Other Uses For The Program. The program can be used as a "baseline" or starting version for other conversion type programs. Most of the subroutines and file handling routines are general enough so that they can be used for a variety of purposes. In fact, most of the subroutines in this program originally came from my standard library source file. To make the program more sophisticated, additional code could be added within the conversion loop to expand or change the keywords or ASCII data to whatever you desire. For example, PR. statements may be expanded to PRINT or vice versa. Many other additions can be made and other functions may come to mind as the program is used.

```
CONVPATB.ASM  V0.7,28SEP78  (CPMKEY=YES)

>?

A>TYPE CONVPATB.PRN


                ;-----------------------------------------
                ;-              CONVPATB.ASM             -
                ;-                (8080)                 -
                ;-----------------------------------------
                ;
                ; V0.7,28SEP78
                ;
                ; PAUL HOLLIDAY
                ; 4807 ARLENE ST.
                ; SAN DIEGO, CA 92117
                ;
                ; THE PURPOSE OF THIS PROGRAM IS TO CONVERT A
                ; FILE FROM PALO ALTO TINY BASIC (PATB)
                ; FORMAT TO ASCII FORMAT.
                ;
                ; THE PATB SOURCE FILE MUST BE ON DISK WITH
                ; FILE TYPE  .TBI
                ; A NEW FILE IS WRITTEN WITH FILE TYPE '.ASC' WHICH
                ; IS THE CONVERTED FILE.
                ;
                ; FIRST, USE DDT TO MAKE SURE THE .TBI FILE IS IN
                ; THE CORRECT FORM AND ADD <FF> <FF> AT END.
                ;
                ; THIS PROGRAM WILL THEN:
                ; 1.    DELETE THE FIRST 2 BYTES OF FILE (TXTNUF)
                ; 2.    CONVERT THE FIRST 2 BYTES OF EACH LINE
                ;       FROM A BINARY LINE NUMBER <LSB> <MSB> TO
                ;       A 4 DIGIT ASCII LINE NUMBER.
                ; 3.    ADD A SPACE (BLANK) AFTER THE LINE NUMBER.
                ; 4.    SEARCH FOR THE NEXT CR (0D HEX) AND
                ;       ADD A LINE FEED (0A HEX) AT END OF LINE.
                ;
                ; LAST BYTES OF INPUT TEXT MUST BE <FF> <FF> HEX.
                ;
                ;
                ;----- CONDITIONAL ASSEMBLY EQUATES -----
FFFF =          YES:    EQU   -1        ;VALUE FOR CONDITIONAL 'IF' ASSY
0000 =          NO:     EQU   NOT YES   ;VALUE FOR CONDITIONAL ASSEMBLY
                ;
0000 =          ALONE:  EQU   NO        ;YES IF STAND ALONE KEYBOARD
FFFF =          CPMKEY: EQU   NOT ALONE ;TRUE IF CPM KEYBD USED
                ;
                ;
                ;-----  EQUATE DICTIONARY ----- AUG78
000D =          CR:     EQU  13         ;CARRIAGE RETURN CHAR
0003 =          CTRLC:  EQU  'C'-64     ;CONTROL-C CHARACTER
0004 =          CTRLD:  EQU  'D'-64     ;CONTROL-D CHARACTER
0010 =          CTRLP:  EQU  'P'-64     ;CONTROL-P CHARACTER
0013 =          CTRLS:  EQU  'S'-64     ;CONTROL-S CHARACTER
0015 =          CTRLU:  EQU  'U'-64     ;CONTROL-U CHARACTER
00FF =          ENDTEXT:EQU  0FFH       ;END OF TEXT SENTINEL
00FF =          ENDSENT:EQU  ENDTEXT    ;END OF SOURCE FILE SENTINEL
001B =          ESC:    EQU  27         ;ESC (ESCAPE) KEY CHARACTEF
000A =          LF:     EQU  10         ;LINE FEED CHARACTER
00FF =          MEND:   EQU  255        ;END SENTINEL FOR TBLS/MSGS
007F =          RUBOUTK:EQU  127        ;RUBOUT OR DELETE KEY
005F =          SHFTO:  EQU  95         ;SHIFT-O KEY
0050 =          SIZEKIN:EQU  80         ;MAX SIZE OF KEYBD INPUT BUFR
0020 =          SPC:    EQU  32         ;SPACE OR BLANK CHARACTER
0001 =          TRUE:   EQU  1          ;VALUE USED FOR FLAGS,ETC
                ;
                ;----- VECTOR MONITOR EQUATES -----
C000 =          MONTRADR:EQU 0C000H     ;RETURN TO VECTOR MONITOR ADDR
C700 =          OUTlV:  EQU  0C700H     ;OUTPUT BYTE IN 'A' REGISTER
005F =          RUBV:   EQU  SHFTO      ;SHIFT-O USED FOR VECTOR RUBOUT
0013 =          SPEED:  EQU  CTRLS      ;VECTOR VIDEO SPEED CONTROL
                ;
                ;----- VIDEO SCREEN EQUATES -----
0080 =          BIAS    EQU  80H        ;BIT 7=1 FOR POLY VTI
D000 =          VIDEO   EQU  0D000H     ;VIDEO SCREEN START ADDRESS
00D4 =          VIDEX   EQU  0D4H       ;MSB'S OF END OF SCREEN
0040 =          VCOLS   EQU  64         ;NUMBER OF COLUMNS
0010 =          VLINES  EQU  16         ;NUMBER OF LINES
0400 =          VSIZE:  EQU  VCOLS*VLINES  ;TOTAL SIZE OF SCREEN
                ;
                ;----- KEYBOARD EQUATES -----
00D0 =          KDATA   EQU  0D0H       ;KEYBOARD DATA IN PORT
0001 =          KRDA    EQU  01H        ;KEYBD DATA AVAIL BIT
0000 =          KRDAI   EQU  00H        ;NOP OR CMA(2F) TO/INVERT
00D1 =          KSTAT   EQU  0D1H       ;KEYBOARD STATUS IN PORT
                ;
                ;----- PRINTER EQUATES -----
CC00 =          CPMLINO:EQU  0CC00H     ;ADDRESS OF PRINTER DRIVER
                ;
                ;
                ;----- CP/M INTERFACE EQUATES -----
0005 =          BDOSADR:EQU  0005H      ;DOS ENTRY POINT (CPM BDOS)
0080 =          BDOSBUF:EQU  80H        ;INPUT DISK BUFFER ADDRESS
0010 =          CLOSEF: EQU  16         ;CLOSE FILE BDOS NUMBER
0001 =          CONS:   EQU  1          ;READ CONSOLE INPUT BYTE TO A
000B =          CONSTAT:EQU  11         ;READ CONSOLE STATUS COMMAND
0000 =          CPMADR: EQU  0000H      ;RETURN TO CP/M ADDRESS
0016 =          CREATEF:EQU  22         ;CREATE FILE COMMAND NUMBER
0013 =          DELETEF:EQU  19         ;DELETE FILE COMMAND NUMBER
000F =          OPENF:  EQU  15         ;OPEN FILE (BDOS CMD NUMBER)
0014 =          READF:  EQU  20         ;READ FILE BDOS NUMBER
0080 =          SECBUF: EQU  BDOSBUF    ;SECTOR BUFFER (PUTBYTE)
0080 =          SIZES:  EQU  128        ;SIZE OF SECTOR BUFFER
0080 =          TBUFF:  EQU  0080H      ;DEFAULT BUFFER HAS COMMAND
005C =          TFCB:   EQU  005CH      ;CPM DEFAULT FILE CONTROL BLK
0002 =          TYPEF:  EQU  2          ;TYPE (OUTPUT) TO CONSOLE (BDOS #)
0015 =          WRITEF: EQU  21         ;WRITE FILE (BDOS NUMBER)
                ;
                ;
                ;END OF EQUATES
                ;
                ;----- JUMP TABLE AND ENTRY/EXTERNAL VECTORS -----
                ;
0100            BEGIN:  ORG  100H       ;START FOR CP/M TPA AREA
0100 C32A01             JMP  START      ;GO TO PROGRAM START
```

```
0103 C37F01          JMP  RESUME     ;RESTART ENTRY
0106 C30500  KEYIN:  JMP  KEYINX     ;WAIT FOR KEYBD,INPUT IN 'A'
0109 C30F06  INKS:   JMP  INKSX      ;INPUT KEYBOARD STATUS
010C C32206  INKD:   JMP  INKDX      ;INPUT KEYBD DATA TO A
010F C32A06  OUT1:   JMP  OUT1X      ;OUTPUT BYTE IN 'A' TO CONSOLE
0112 C32A06  OUT2:   JMP  OUT1X      ;OUTPUT BYTE IN A TO VID SCREEN
0115 C300CC  OUT3:   JMP  CPMLINO    ;OUTPUT BYTE IN 'A' TO PRINTER
0118 C300C0  MONTR:  JMP  MONTRADR   ;RETURN TO MONITOR
011B C30000  CPM:    JMP  CPMADR     ;RETURN TO CP/M ADDRESS
011E C30500  BDOS:   JMP  BDOSADR    ;CALL BDOS ENTRY POINT
0121 C300C0          JMP  MONTRADR   ;SPARE
0124 C300C0          JMP  MONTRADR   ;SPARE
0127 C300C0          JMP  MONTRADR   ;SPARE
                     ;
                     ;----------------------------------------
                     ;     INITIAL START AND SETUP            ;
                     ;----------------------------------------
                     START:
012A CDAF03          CALL INITIAL    ;INITIALIZE IF NECESSARY
012D CDEF02          CALL CLEAR      ;CLEAR SCREEN
0130 017106          LXI  B,MSG1     ;GET ADDRESS OF SIGN-ON MSG
0133 CD0105          CALL PRNTM      ;SEND SIGN-ON MESSAGE
                     ;
                     ;-- CHECK FOR FILE NAME AT LOAD TIME --
0136                 STRT10: DS   0  ;
0136 3A8000          LDA  TBUFF+0    ;GET TRANSIENT BUFFER FIRST WRD
0139 FE02            CPI  2          ;SEE IF ANY INPUT
013B DA4401          JC   STRT20     ;NO INPUT IF 0 OR 1
013E CDF003          CALL MAKEFCBA   ;BUILD FCB1 AND FCB2 FROM NAME
0141 C37F01          JMP  STRT60     ;GO TO 'MOUNT DISK' POINT
                     ;
                     ;-- REQUEST FILE NAME FROM USER --
0144                 STRT20: DS   0  ;
0144 014C07          LXI  B,MSG8     ;'FILE NAME...'
0147 CD0105          CALL PRNTM      ;
014A CDCE03          CALL INPUT1     ;INPUT UNTIL CR OR CTRL-C
014D FE03            CPI  CTRLC      ;WAS CTRL-C ENTERED?
014F CAB902          JZ   EXIT2      ;YES, GO TO EXIT
0152 CD3603          CALL CRLF       ;RESPONSE
                     ;
                     ;-- MOVE KEYBOARD INPUT TO INNAME1 BUFFER --
0155 010D00          LXI  B,13       ;NUMBER OF CHARS IN 'INNAME'
0158 118D07          LXI  D,KINBUF   ;ADDRESS OF KEYBD INPUT BUFFER
015B 213208          LXI  H,INNAME1  ;ADDRESS TO STORE NAME
015E CDE004          CALL MOVEIT     ;MOVE NAME TO BUFFER
                     ;
                     ;-- MOVE NAME TO OUTNAME1 BUFFER --
0161 213F08          LXI  H,OUTNAME1 ;ADDRESS TO STORE NAME
0164 CDE004          CALL MOVEIT     ;MOVE NAME TO BUFFER
                     ;
                     ;-- SETUP FCB'S AND SEND MESSAGE IF ERROR --
0167 CD7C05          CALL SETFCBS    ;SETUP FCB1 AND 2 FROM INNAME,
016A D27C01          JNC  STRT50     ;CONTINUE IF NO ERROR
016D CD5D05          CALL SENDERR    ;ERROR MESSAGE
0170 012707          LXI  B,MSG6     ;'INVALID NAME..
0173 CD0105          CALL PRNTM      ;
0176 CDC405          CALL WAITKEY    ;WAIT FOR OPERATOR
0179 C34401          JMP  STRT20     ;TRY INPUTTING NAME AGAIN
                     ;
017C                 STRT50: DS   0  ;
017C CD0204          CALL SETTYPE    ;SET FCB1 AND FCB2 FILE TYPES
                     ;
017F                 STRT60: DS   0  ;ENTRY FROM ABOVE
                     ;
017F                 RESUME: DS   0  ;RESTART/RE-ENTRY POINT
                     ;
                     ;-- REQUEST USER TO MOUNT DISK --
017F                 MAIN05: DS   0  ;
017F CD6405          CALL SEND2      ;'MOUNT DISK IN DRIVE ?,...'
0182 CD0601          CALL KEYIN      ;GET RESPONSE
0185 FE0D            CPI  CR         ;IS IT <RETURN> ?
0187 C27F01          JNZ  MAIN05     ;NO, REPEAT
                     ;
                     ;-- SETUP AND PREPARE TO READ FILE --
018A                 MAIN10: DS   0  ;
018A 01FA06          LXI  B,MSG3     ;'READING FILE' MESSAGE
018D CD0105          CALL PRNTM      ;SEND MESSAGE
0190 010800          LXI  B,8        ;LENGTH TO PRINT
0193 11EB07          LXI  D,FCB1+1   ;START OF FILE NAME IN FCB
0196 CD0705          CALL PRNTX      ;PRINT FILE NAME
                     ;
                     ;-- OPEN FILE TO READ --
0199                 MAIN20: DS   0  ;
0199 11EA07          LXI  D,FCB1     ;ADDRESS OF FILE CONTROL BLOCK
019C CD3606          CALL OPEN       ;OPEN FILE
019F C2B101          JNZ  MAIN30     ;CONTINUE IF NO ERROR
01A2 CD5D05          CALL SENDERR    ;GENERAL ERROR MESSAGE
01A5 017507          LXI  B,MSG9F    ;'FILE NOT FOUND..
01A8 CD0105          CALL PRNTM      ;SEND MESSAGE
01AB CDC405          CALL WAITKEY    ;WAIT FOR ANY KEY
01AE C3B002          JMP  EXIT1      ;GO TO EXIT
                     ;
                     ;-- FILE OPENED, FINISH SETUP --
01B1                 MAIN30: DS   0  ;
01B1 CDC203          CALL INITIAL3   ;RESET BYTE COUNT AND SECTOR
01B4 214C06          LXI  H,TEXTBUF  ;GET START OF INPUT BUFFER
01B7 22E307          SHLD P2         ;SET CURRENT POINTER
01BA AF              XRA  A          ;GET A ZERO
01BB 32A008          STA  FCB1+32    ;SET NEXT RECORD=0
01BE 32BC07          STA  GETFLAG    ;SET GETBYTE FLAG TO ZERO
01C1 CDBC02          CALL BLANKSB    ;BLANK SECTOR BUFFER
                     ;
                     ;-- LOOP AND READ/STORE FILE FROM DISK TO MEM --
01C4                 MAIN40: DS   0  ;
01C4 2AE507          LHLD P3         ;GET CURRENT END POINTER
01C7 CDDA02          CALL CHECKMEM   ;SEE IF AT END OF MEMORY
01CA DAEC01          JC   MAIN50     ;STOP IF NO MEMORY LEFT
01CD CD5E03          CALL GETBYTE    ;GET A BYTE FROM DISK
01D0 DAEC01          JC   MAIN50     ;STOP READ IF END OF FILE
01D3 CDA805          CALL STOREP2    ;STORE BYTE AND END SENTINEL
01D6 FEFF            CPI  ENDTEXT    ;IS IT END OF TEXT?
01D8 C2C401          JNZ  MAIN40     ;NO, CONTINUE READING
01DB CD5E03          CALL GETBYTE    ;MAYBE END, GET ANOTHER BYTE
01DE DAEC01          JC   MAIN50     ;STOP IF END OF FILE
01E1 CDA805          CALL STOREP2    ;STORE BYTE IN BUFFER
01E4 FEFF            CPI  ENDTEXT    ;IS IT SECOND END SENTINEL?
01E6 CAEC01          JZ   MAIN50     ;YES, STOP READING
01E9 C3C401          JMP  MAIN40     ;NO, READ AGAIN
                     ;
                     ;-- SETUP POINTER TO STORE CONVERTED FILE --
01EC                 MAIN50: DS   0  ;
01EC 2AE107          LHLD P1         ;GET START ADDR OF INPUT TEXT
01EF 228707          SHLD BUFF1      ;SET START OF SOURCE TO CONVERT
01F2 2AE507          LHLD P3         ;GET END OF INPUT TEXT
01F5 23              INX  H          ;P3=P3+1
01F6 36FF            MVI  M,ENDTEXT  ;PUT 2ND END SENTINEL AT P3+1
01F8 23              INX  H          ;P3=P3+1 SO TOTAL=P3+2
01F9 228907          SHLD BUFF2      ;SET START OF NEW TEXT
                     ;
                     ;-- CONVERT FILE IN BUFF1 AND STORE IN BUFF2 --
01FC 013807          LXI  B,MSG7     ;'CONVERTING FILE..' MESSAGE
01FF CD0105          CALL PRNTM      ;PRINT MESSAGE
                     ;
                     KI
                     ;-- DELETE FIRST 2 BYTES OF FILE (TXTNUF) --
0202 2A8707          LHLD BUFF1      ;GET POINTER TO START OF FILE
0205 23              INX  H          ;ADVANCE ONE BYTE
0206 23              INX  H          ;ADVANCE SECOND BYTE
0207 228707          SHLD BUFF1      ;RESTORE NEW POINTER
                     ;
                     ;-- LOOP AND CONVERT EACH LINE --
020A                 MOVE10: DS   0  ;
020A 2A8707          LHLD BUFF1      ;GET DATA TO MOVE
020D CDC802          CALL CHECKEND   ;SEE IF AT END OF TEXT
0210 DA7302          JC   MOVE60     ;YES, GO TO END
0213 4E              MOV  C,M        ;GET LSB OF LINE # FROM BFFR
0214 23              INX  H          ;NEXT SOURCE ADDR
0215 46              MOV  B,M        ;PUT MSB OF LINE # IN B
0216 23              INX  H          ;INCREMENT POINTER
0217 228707          SHLD BUFF1      ;UPDATE SOURCE  MEM POINTER
021A CDF502          CALL CONVASCI   ;CONVERT TO 4 DIGIT ASCII
                     ;
021D 2A8907          LHLD BUFF2      ;GET DESTINATION POINTER
0220 CDDA02          CALL CHECKMEM   ;SEE IF AT END OF MEMORY
0223 DA7302          JC   MOVE60     ;NO MEMORY LEFT
0226 3A3403          LDA  M4         ;GET NEXT ASCII LINE # DIGIT
0229 77              MOV  M,A        ;STORE NEXT DIGIT
022A 23              INX  H          ;NEXT DEST ADDR
022B 3A3303          LDA  M3         ;GET ASCII DIGIT
022E 77              MOV  M,A        ;STORE NEXT LINE # DIGIT
022F 23              INX  H          ;NEXT DEST. ADDR
0230 3A3203          LDA  M2         ;GET ASCII DIGIT
0233 77              MOV  M,A        ;STORE ASCII LINE #
0234 23              INX  H          ;NEXTT DEST. ADDR
0235 3A3103          LDA  M1         ;GET LSB OF ASCII LINE #
0238 77              MOV  M,A        ;STORE LSB OF ASCII LINE #
0239 23              INX  H          ;NEXT DESTINATION ADDRESS
023A 3620            MVI  M,' '      ;STORE ASCII BLANK AFTER NMBR
023C 23              INX  H          ;INCREMENT MEM POINTER
023D CDDA02          CALL CHECKMEM   ;SEE IF AT END OF MEMORY
0240 DA7302          JC   MOVE60     ;NO MEMORY LEFT
0243 228907          SHLD BUFF2      ;SAVE POINTER
                     ;
                     ;-- MOVE REST OF LINE UNTIL <CR> --
0246                 MOVE30: DS   0
0246 2A8707          LHLD BUFF1      ;GET SOURCE POINTER
0249 CDC802          CALL CHECKEND   ;IS IT END OF SOURCE?
024C DA7302          JC   MOVE60     ;YES, GO TO END
024F 7E              MOV  A,M        ;PICKUP SOURCE DATA
0250 23              INX  H          ;INCREMENT POINTER
0251 228707          SHLD BUFF1      ;SAVE NEW SOURCE PICKUP POINTR
0254 2A8907          LHLD BUFF2      ;GET DESTINATION POINTER
0257 77              MOV  M,A        ;STORE BYTE AT NEW PLACE
0258 23              INX  H          ;INCREMENT DESTINATION POINTER
0259 228907          SHLD BUFF2      ;SAVE NEW DESTINATION POINTER
025C FE0D            CPI  CR         ;WAS IT "RETURN"?
025E CA6A02          JZ   MOVE50     ;YES, GO STORE LINE FEED
0261 CDDA02          CALL CHECKMEM   ;SEE IF AT END OF MEMORY
0264 DA7302          JC   MOVE60     ;NO MEMORY LEFT
0267 C34602          JMP  MOVE30     ;LOOP UNTIL END OF BUFFER 1
                     ;
026A 360A            MOVE50: MVI  M,LF       ;STORE LINE FEED AFTER CR
026C 23              INX  H          ;NEXT DESTINATION ADDRESS
026D 228907          SHLD BUFF2      ;SAVE NEW POINTER
0270 C30A02          JMP  MOVE10     ;REPEAT FOR NEXT LINE
                     ;
                     ;-- END OF SOURCE MOVE AND CONVERT, SETUP TO WRITE --
                     ;
0273                 MOVE60: DS   0  ;
0273 2A8907          LHLD BUFF2      ;GET DESTINATION POINTER
0276 3620            MVI  M,' '      ;STORE A BLANK
0278 23              INX  H          ;NEXT
0279 3645            MVI  M,'E'      ;STORE AN 'E'
027B 23              INX  H          ;NEXT
027C 364E            MVI  M,'N'      ;STORE AN 'N'
027E 23              INX  H
027F 3644            MVI  M,'D'      ;STORED "END" FOR CPM ASSEMBLER
0281 23              INX  H          ;NEXT ADDRESS
0282 36FF            MVI  M,ENDTEXT  ;STORE END SENTINEL
0284 23              INX  H          ;NEXT ADDRESS
0285 36FF            MVI  M,ENDTEXT  ;STORE ANOTHER END SENTINEL
                     ;
0287 010B07          LXI  B,MSG4     ;'WRITING FILE'
028A CD0105          CALL PRNTM      ;SEND MESSAGE
028D 2AE507          LHLD P3         ;GET ADDR OF END OF FILE1
0290 23              INX  H          ;P3=P3+1
0291 23              INX  H          ;P3=P3+2 TOTAL
0292 228907          SHLD BUFF2      ;SET START OF BUFFER 2=P3+2
0295 CDC805          CALL WRITE2     ;WRITE NEW FILE
0298 DC5D05          CC   SENDERR    ;SEND ERROR MESSAGE IF CARRY
                     ;
                     ;-- CLOSE BOTH FILES --
029B 11EA07          LXI  D,FCB1     ;ADDRESS OF READ FILE
029E CD4006          CALL CLOSE      ;CLOSE FILE
02A1 CC5D05          CZ   SENDERR    ;ERROR CLOSING FILE
02A4 11B808          LXI  D,FCB2     ;ADDRESS OF WRITE FILE FCB
02A7 CD4006          CALL CLOSE      ;CLOSE WRITE FILE
02AA CC5D05          CZ   SENDERR    ;GENERAL ERROR
02AD C3B002          JMP  EXIT1      ;GO TO EXIT
                     ;
                     ;
                     ;-- FINISHED --
02B0                 EXIT1:  DS   0  ;
02B0 011C07          LXI  B,MSG5     ;'FINISHED' MESSAGE
02B3 CD0105          CALL PRNTM      ;SEND MESSAGE
02B6 CD3603          CALL CRLF       ;
```

```
02B9                    EXIT2:  DS   0           ;EXIT TO CP/M
02B9 C31B01                     JMP  CPM         ;RETURN TO CPM
                        ;
                        ;
                        ;-------------------------------------------------
                        ;-  THE FOLLOWING ROUTINES ARE IN ALPHABETIC ORDER -
                        ;-------------------------------------------------
                        ;
                        ;----- BLANKSB -----
                        ; STORES BLANKS IN SECTOR BUFFER
                        ; BLANK: HL = ADDRESS TO START STORING BLANKS
                        ;        BC = LENGTH TO STORE
                        ;
02BC                    BLANKSB:DS   0           ;
02BC 218000                     LXI  H,SECBUF    ;GET SECTOR BUFFER ADDR
02BF 018000                     LXI  B,SIZES     ;GET LENGTH OF SECTOR BUFFR
02C2 3E20               BLANK:  MVI  A,' '       ;CHAR TO STORE=BLANK (SPACE)
02C4 CD5303                     CALL FILL        ;STORE/FILL FOR LENGTH IN BC
02C7 C9                         RET
                        ;
                        ;----- CHECKEND -----
                        ; CHECK FOR 2 SUCCESSIVE BYTES = 'ENDTEXT'
                        ; ENTER WITH ADDRESS IN HL
                        ; RETURNS WITH CARRY SET IF YES (END OF TEXT)
                        ;
02C8                    CHECKEND:DS  0           ;
02C8 7E                         MOV  A,M         ;GET BYTE FROM MEM
02C9 FEFF                       CPI  ENDTEXT     ;IS IT END SENTINEL?
02CB C2D802                     JNZ  CHKE50      ;NO, EXIT
02CE 23                         INX  H           ;YES, INCREMENT TO NEXT MEM
02CF 7E                         MOV  A,M         ;GET NEXT BYTE
02D0 2B                         DCX  H           ;RESTORE ADDRESS IN HL
02D1 FEFF                       CPI  ENDTEXT     ;IS IT ANOTHER END SENTINEL?
02D3 C2D802                     JNZ  CHKE50      ;NO, GO TO EXIT
02D6 37                         STC              ;YES, SET CARRY
02D7 C9                         RET              ;RETURN
02D8 B7                 CHKE50: ORA  A           ;CLEAR CARRY
02D9 C9                         RET              ;RETURN
                        ;
                        ;----- CHECKMEM ----- 9/28/78
                        ; CHECK FOR END OF MEMORY BUFFER
                        ; ENTER WITH:  HL = ADDRESS TO BE CHECKED
                        ; RETURNS WITH CARRY SET IF END OF MEMORY
                        ; USES POINTER 'MAXMEM' WHICH HAS ADDRESS OF LAST AVAIL
                        ;                                                  MEM
02DA                    CHECKMEM:DS  0           ;
02DA E5                         PUSH H           ;SAVE HL
02DB EB                         XCHG             ;PUT ADDRESS TO CHECK IN DE
02DC 2ADF07                     LHLD MAXMEM      ;GET LAST AVAILABLE MEMORY ADDR
02DF 7C                         MOV  A,H         ;PUT MSB OF MAX MEM IN A
02E0 BA                         CMP  D           ;COMPARE TO MSB OF ADDR TO CHK
02E1 C2EC02                     JNZ  CHKM50      ;NOT AT MAX, GO TO EXIT
02E4 7D                         MOV  A,L         ;GET LSB OF MAX MEM IN A
02E5 BB                         CMP  E           ;COMPARE TO LSB OF ADDR TO CHK
02E6 D2EC02                     JNC  CHKM50      ;NOT PAST MAX YET (E<=A)
02E9 C3ED02                     JMP  CHKM60      ;E > A  EXIT WITH CARRY SET
02EC B7                 CHKM50: ORA  A           ;CLEAR CARRY
02ED E1                 CHKM60: POP  H           ;RESTORE HL
02EE C9                         RET              ;RETURN
                        ;
                        ;END OF CHECKMEM.LIB
                        ;
                        ;----- CLEAR -----
                        ; CLEAR VIDEO SCREEN
                        ;
02EF                    CLEAR:  DS   0           ;
02EF                    CLEARV: DS   0           ;CLEAR USING VECTOR MONITOR
02EF 3E04                       MVI  A,CTRLD     ;VECTOR CLEAR SCREEN CHAR
02F1 CD0F01                     CALL OUT1        ;CLEAR SCREEN, SET VECTOR MON
02F4 C9                         RET
                        ;
                        ;----- CONVASCI ----- 9/78
                        ; CONVERT BINARY NUMBER IN BC TO ASCII
                        ; ENTBR WITH NUMBER IN BC (B=LSB, C=MSB)
                        ; RETURNS WITH 5 DIGIT NUMBER IN M1-M5 (LSB-MSB)
                        ;
02F5                    CONVASCI:DS  0           ;
02F5 C5                         PUSH B           ;SAVE ARGUMENTS
02F6 AF                         XRA  A           ;CLEAR A
02F7 323103                     STA  M1          ;CLEAR LSB STORAGE
02FA 323203                     STA  M2          ;CLEAR
02FD 323303                     STA  M3          ;CLEAR
0300 323403                     STA  M4          ;CLEAR
0303 323503                     STA  M5          ;CLEAR MSB STORAGE
0306 60                         MOV  H,B         ;MOVE NUMBER IN BC TO HL
0307 69                         MOV  L,C         ;DITTO
                        ;
0308 CD2803                     CALL CONVONE     ;CONVERT ONE ASCII DIGIT
030B 323103                     STA  M1          ;STORE LSB
030E CD2803                     CALL CONVONE     ;CONVERT ONE ASCII DIGIT
0311 323203                     STA  M2          ;STORE NEXT
0314 CD2803                     CALL CONVONE     ;CONVERT ONE ASCII DIGIT
0317 323303                     STA  M3          ;STORE NEXT
031A CD2803                     CALL CONVONE     ;CONVERT ONE ASCII DIGIT
031D 323403                     STA  M4          ;STORE 2ND MSB
0320 CD2803                     CALL CONVONE     ;CONVERT ONE ASCII DIGIT
0323 323503                     STA  M5          ;STORE MSB
                        ;
0326 C1                         POP  B           ;RESTORE BC
0327 C9                         RET              ;RETURN
                        ;
0328 00                 CONVONE:DB   0           ;CONVERT ONCE (DIVIDE BY 10)
0329 0E0A                       MVI  C,10        ;TO DIVIDE BY TEN
032B CD4003                     CALL DIVIDE      ;DIVIDE HL BY TEN
032E C630                       ADI  '0'         ;ADD ASCII BIAS
0330 C9                         RET              ;RETURN
                        ;
0331 00                 M1      DB   0           ;LSB OF CONVERTED ASCII NMBR
0332 00                 M2      DB   0           ;
0333 00                 M3      DB   0           ;
0334 00                 M4      DB   0           ;
0335 00                 M5      DB   0           ;MSB OF CONVERTED ASCII NMBR
                        ;
                        ;----- CRLF -----
0336                    CRLF:   DS   0           ;
0336 013D03                     LXI  B,MSGCR     ;GET ADDRESS OF CR, LF
0339 CD0105                     CALL PRNTM       ;PRINT IT
033C C9                         RET              ;RETURN
033D 0D0AFF             MSGCR:  DB   CR.LF,MEND
                        ;
                        ;----- DIVIDE -----
                        ; DIVIDE1.LIB 9/78
                        ;
                        ; UNSIGNED BINARY DIVIDE
                        ;
                        ; DIVIDES 16 BIT DIVIDEND BY 8 BIT
                        ; DIVISOR PRODUCING A 16 BIT QUOTIENT
                        ; AND AN 8 BIT REMAINDER.
                        ;
                        ; CALL WITH:
                        ;       HL = DIVIDEND
                        ;       C  = DIVISOR
                        ;
                        ; RETURN WITH:
                        ;       HL = QUOTIENT
                        ;       C  = DIVISOR
                        ;       A  = REMAINDER
                        ;       CARRY = SET ON DIVIDE BY ZERO
                        ;
0340 AF                 DIVIDE: XRA     A        ; CLEAR EXTENSION OF DIVIDEND REG
0341 B9                         CMP     C        ; CHECK IF DIVISOR = 0
0342 37                         STC              ; MARK DIVIDE ERROR
0343 C8                         RZ               ; RETURN ON / BY 0
0344 0610                       MVI     B,16     ; SET LOOP COUNTER
0346 29                 DIVID1: DAD     H        ; SHIFT DIVIDEND LEFT INTO
0347 17                         RAL              ; DIVIDEND EXTENSION REG.
0348 B9                         CMP     C        ; EXTENSION >= DIVISOR?
0349 DA4E03                     JC      DIVID2   ; QUOTIENT BIT = 0?
034C 91                         SUB     C        ; SUBTRACT DIVISOR FROM EXTENSION
034D 23                         INX     H        ; SET LEAST BIT OF QUOTIENT
034E 05                 DIVID2: DCR     B        ; TEST LOOP COUNT
034F C24603                     JNZ     DIVID1   ; DONE?
0352 C9                         RET
                        ;END OF DIVIDE1.LIB
                        ;
                        ;
                        ;----- FILL -----
                        ; STORES OF FILLS WITH CHARACTER FOR LENGTH
                        ; HL = STORE ADDRESS
                        ; BC = LENGTH TO STORE
                        ; A  = CHARACTER TO STORE
                        ;
0353                    FILL:   DS   0           ;
0353 77                 FILL10: MOV  M,A         ;STORE CHAR AT ADDRESS IN HL
0354 23                         INX  H           ;NEXT STORE ADDRESS
0355 0B                         DCX  B           ;LENGTH=LENGTH-1
0356 57                         MOV  D,A         ;SAVE CHAR IN D
0357 78                         MOV  A,B         ;GET LSB OF COUNTER IN A
0358 B1                         ORA  C           ;OR WITH MSB OF LENGTH
0359 7A                         MOV  A,D         ;RESTORE CHAR TO STORE
035A C25303                     JNZ  FILL10      ;LOOP UNTIL BC=0
035D C9                         RET              ;RETURN
                        ;
                        ;----- GETBYTE ----- VER 9/23/78
                        ; GET A BYTE FROM SECTOR BUFFER
                        ; RETURNS WITH BYTE IN 'A'
                        ; SETUP 'FCB1' BEFORE CALLING
                        ; IF SECTOR BUFFER IS EMPTY, READS FROM DISK
                        ; ALL REGISTERS MAY BE ALTERED ON RETURN
                        ; RETURNS WITH CARRY SET IF END OF FILE DETECTED.
                        ;
035E                    GETBYTE:DS   0           ;
035E 3A8C07                     LDA  GETFLAG     ;GET 'THRU ONCE' FLAG
0361 FE01                       CPI  TRUE        ;HAVE WE BEEN THRU ONCE?
0363 CA7803                     JZ   GETB10      ;YES, CONTINUE
0366 3E01                       MVI  A,TRUE      ;NO, GET VALUE TO SET FLAG
0368 328C07                     STA  GETFLAG     ;SET FLAG, BEEN THRU ONCE
036B 11EA07                     LXI  D,FCB1      ;GET FCB ADDRESS
036E CD6506                     CALL READ        ;READ SECTOR
0371 C27803                     JNZ  GETB10      ;CONTINUE IF NO ERROR
0374 F5                         PUSH PSW         ;ERROR, PUSH SO EXITS OK
0375 C39903                     JMP  GETB30      ;GO TO ERROR CHECK/PRINT
                        ;
0378 2A8807             GETB10: LHLD SECBP       ;GET SECTOR BUFFER POINTER
037B 7E                         MOV  A,M         ;GET BYTE FROM BUFFER IN A
037C 23                         INX  H           ;INCREMENT TO NEXT BYTE
037D 22E807                     SHLD SECBP       ;SET NEW SECTOR BFR POINTER
0380 47                         MOV  B,A         ;SAVE BYTE IN B
0381 3A8607                     LDA  BCNT        ;GET BYTE COUNTER
0384 3D                         DCR  A           ;DECREMENT BYTE COUNT
0385 328607                     STA  BCNT        ;RESTORE NEW BYTE COUNT
0388 78                         MOV  A,B         ;RESTORE A
0389 C2AD03                     JNZ  GETB100     ;EXIT IF BUFFER NOT EMPTY
                        ;
038C F5                         PUSH PSW         ;SAVE LAST BYTE
038D CDBC02                     CALL BLANKSB     ;BLANK SECTOR BUFFER
0390 11EA07                     LXI  D,FCB1      ;GET FCB ADDRESS
0393 CD6506                     CALL READ        ;READ NEXT SECTOR
0396 C2A903                     JNZ  GETB60      ;CONTINUE IF NO ERROR
0399 FE01               GETB30: CPI  1           ;IS IT END OF FILE?
039B CAA403                     JZ   GETB40      ;YES, GO TO SET EOF
039E 01B807                     LXI  B,MSG22     ;'READ ERROR..'
03A1 CD0105                     CALL PRNTM       ;
                        ;
03A4                    GETB40: DS   0           ;END OF FILE ENCOUNTERED
03A4 F1                         POP  PSW         ;FIX STACK
03A5 37                         STC              ;SET CARRY IF END OF FILE
03A6 C3AE03                     JMP  GETB110     ;GO TO EXIT
                        ;
03A9                    GETB60: DS   0           ;READ SECTOR AND NO ERROR
03A9 CDC203                     CALL INITIAL3    ;RESET SECTOR POINTER/COUNTER
03AC F1                         POP  PSW         ;RESTORE LAST BYTE
                        ;
03AD                    GETB100:DS   0           ;
03AD B7                         ORA  A           ;CLEAR CARRY, NORMAL EXIT
03AE C9                 GETB110:RET              ;RETURN
                        ;
                        ;
                        ;----- INITIAL ----- 6/78
                        ; ONE TIME INITIALIZATION
                        ;
03AF                    INITIAL: DS  0           ;
03AF 2A0600                     LHLD BDOSADR+1   ;GET ADDRESS OF CPM FBASE
03B2 11ECFF                     LXI  D,-20       ;ALLOW 20 SPARES SLOP
03B5 19                         DAD  D           ;SUBTRACT SPARES TO GET MAXMEM
03B6 22DF07                     SHLD MAXMEM      ;SET LAST AVAIL MEM ADDRESS
```

```
03B9 CDC003             CALL INITIAL2   ;CLEAR FLAGS
03BC CDC203             CALL INITIAL3   ;RESET SECTOR BUFFER POINTERS
03BF C9                 RET             ;RETURN
                ;
                ;----- INITIAL2 -----
                ; CLEARS FLAGS AND NORMAL CLEANUP
                ;
03C0            INITIAL2:DS  0          ;
03C0 AF                 XRA  A          ;CLEAR A
03C1 C9                 RET             ;RETURN
                ;
                ;----- INITIAL3 -----
                ; SETS BCNT AND SECTOR BUFFER POINTER,
                ; DONT ADD OR CHANGE,USED IN GETBYTE AND PUTBYTE
03C2            INITIAL3:DS  0          ;
03C2 3E80               MVI  A,SIZLS    ;GET SIZE OF SECTOR BUFFER
03C4 320007             STA  BCNT       ;RESET SECTOR BUFFER COUNTER
03C7 218000             LXI  H,SECBUF   ;GET SECTOR BUFFER ADDRESS
03CA 22E807             SHLD SECBP      ;RESET POINTER TO START OF BFR
03CD C9                 RET
                ;
                ;----- INPUT1 ----- 9/23/78
                ; INPUTS A STRING INTO KEYBD INPUT BUFFER
                ; ENDS INPUT WHEN A 'CR' OR 'CTRL-C' ENTERED
                ; CR CHARACTER IS STORED IN KINBUF (KEY INPUT BUFFER)
                ; RETURNS WITH LENGTH IN C
                ; LENGTH INCLUDES CR
                ;
03CE            INPUT1: DS   0          ;
03CE CD9505             CALL SETKIN     ;RESET KEYBD INPUT BUFFER
03D1 0E00               MVI  C,0        ;SET COUNT IN C=0
03D3            INP20:  DS   0          ;
03D3 CD0601             CALL KEYIN      ;GET KEYBOARD ENTRY
                        IF   ALONE      ;IF STAND ALONE VERSION
                        CALL OUT1       ;ECHO ON SCREEN
                        NOP             ;IF RUBOUT, INSERT CHECK1
                        ENDIF           ;END OF STAND ALONE EXTRA STUFF
03D6 FE03               CPI  CTRLC      ;IS IT CNTRL-C?
03D8 CAEF03             JZ   INP50      ;YES, GO TO EXIT
03DB CDBA05             CALL SUMCHAR1   ;INCREMENT COUNTER
03DE CD5105             CALL SAVEIN     ;SAVE INPUT STRING
03E1 FE0D               CPI  CR         ;IS IT RETURN?
03E3 CAE903             JZ   INP40      ;YES-GO TO CR EXIT
03E6 C3D303             JMP  INP20      ;LOOP UNTIL DONE
                ;
03E9            INP40:  DS   0          ;NORMAL CR EXIT
03E9 F5                 PUSH PSW        ;SAVE A AND FLAGS
03EA 3A8B07             LDA  CHARCNT    ;GET NUMBER OF CHARS ENTERED
03ED 4F                 MOV  C,A        ;SET C=LENGTH JUST INPUT
03EE F1                 POP  PSW        ;RESTORE A AND FLAGS
03EF C9         INP50:  RET             ;RETURN
                ;
                ;END OF INPUT1.LIB
                ;
                ;----- MAKEFCBA ----- 9/24/78
                ; MOVES TFCB TO FCB1 AND FCB2
                ; SETS FILE TYPE TO '.TBI' AND '.ASC'
                ;
03F0            MAKEFCBA:DS  0          ;
03F0 011000             LXI  B,16       ;GET LENGTH TO MOVE
03F3 115C06             LXI  D,TFCB     ;PICKUP ADDRESS
03F6 21EA07             LXI  H,FCB1     ;DESTINATION=READ FCB
03F9 CDE004             CALL MOVEIT     ;MOVE TFCB TO FCB1
                ;
                ;-- MOVE TFCB TO FCB2 --
03FC 21B806             LXI  H,FCB2     ;DESTINATION=WRITE FCB AREA
03FF CDE004             CALL MOVEIT     ;MOVE TFCB TO FCB2
                ;
0402            SETTYPE:DS   0          ;STORE READ AND WRITE FILE TYPE
0402 010300             LXI  B,3        ;LENGTH TO MOVE
0405 112C08             LXI  D,RFT      ;ADDR OF READ FILE TYPE CHARS
0408 21F307             LXI  H,FCB1+9   ;DESTINATION=FILE TYPE AREA
040B CDE004             CALL MOVEIT     ;MOVE READ FILE TYPE TO FCB
                ;
040E 112F08             LXI  D,WFT      ;ADDR OF WRITE FILE TYPE CHARS
0411 211408             LXI  H,FCB2+9   ;DESTINATION=FILE TYPE AREA
0414 CDE004             CALL MOVEIT     ;MOVE WRITE FILE TYPE TO FCB
0417 C9                 RET             ;RETURN
                ;
                ;MAKEFCB.LIB
                ;18SEP78
                ;----- MAKEFCB -----
                ; MAKE CP/M FILE CONTROL BLOCK
                ; VERSION 0.4,15SEP78
                ; ORIGINAL WAS MAKEFCB.LIB V0.2,28OCT77,CPM VOL-8
                ; CHANGED SKIPS1 FROM CPI M TO CMP M (ERROR 04SEP78)
                ; ADDED CHECK FOR 'CR' AT VALCHR
                ;
                ; CREATE A CP/M FILE CONTROL BLOCK  FROM
                ; A COMMAND STRING AT THE ADDRESS IN HL
                ; AND PLACE IT AT THE ADDRESS IN DE.  RETURN
                ; WITH THE CARRY SET IF AN ERROR OCCURS.
                ;
                ; HL = ADDRESS OF COMMAND STRING
                ; DE = ADDRESS OF FCB
                ; A  = 0 IF READ OPERATION, A=1 IF WRITE OPERATION
                ;
                ;----- LOCAL EQUATES -----
0021 =          FCBSIZ: EQU     33
000B =          FNMLEN: EQU     11      ; FILE NAME LENGTH

0418            MAKEFCB:DS      0       ; ENTRY POINT (CALL) LABEL 9/78
0418 E5         MTFCB:  PUSH    H       ; SAVE CMD STRING PTR
0419 D5                 PUSH    D       ; SAVE FCB ADDRESS
041A 32E707             STA     RWFLAG  ;SAVE READ OR WRITE FLAG
041D 012100             LXI     B,FCBSIZ; CLEAR ENTIRE FCB AREA
0420 3E00               MVI     A,0     ;
0422 CDD204             CALL    FILLB   ;
0425 D1                 POP     D       ; FILL FILE NAME WITH SPACES
0426 D5                 PUSH    D       ;
0427 13                 INX     D       ;
0428 010B00             LXI     B,FNMLEN;
042B 3E20               MVI     A,' '   ;
042D CDD204             CALL    FILLB   ;
                ;
0430 D1                 POP     D       ; RESTORE POINTERS
0431 E1                 POP     H       ;
                ;
0432 CDC804             CALL    SKIPS   ; SKIP LEADING SPACES
0435 23                 INX     H       ; CHECK FOR DISK CODE
0436 7E                 MOV     A,M     ;
0437 2B                 DCX     H       ;
0438 FE3A               CPI     ':'     ;
043A C24804             JNZ     MTFCB1  ; JUMP ON NO CODE
                ;
043D 7E                 MOV     A,M     ; TEST IF DISK CODE GOOD
043E 23                 INX     H       ;
043F 23                 INX     H       ;
0440 DE40               SBI     '@'     ;
0442 D8                 RC              ; MAKE ERROR RETURN IF BAD
0443 FE5B               CPI     'Z'+1   ;
0445 3F                 CMC             ;
0446 D8                 RC              ;
0447 12                 STAX    D       ; STORE DISK CODE AT FCB + 0
                ;
0448 13         MTFCB1: INX     D       ;
0449 0E08               MVI     C,8     ; PROCESS FILE NAME FIELD
044B CD6004             CALL    GETNAM  ;
044E 7E                 MOV     A,M     ; TEST FOR FILE TYPE SEPARATOR
044F 23                 INX     H       ;
0450 FE2E               CPI     '.'     ;
0452 C25C04             JNZ     MTFCB2  ;
                ;
0455 0E03               MVI     C,3     ; PROCESS FILE TYPE FIELD
0457 CD6004             CALL    GETNAM  ;
045A 7E                 MOV     A,M     ;
045B 23                 INX     H       ;
                ;
045C CDBA04     MTFCB2: CALL    TERMT   ; TEST FOR CORECT TERMINATOR
045F C9                 RET
                ;
                ; PROCESS NAME FIELD
                ;
0460 7E         GETNAM: MOV     A,M     ; GET CHAR FROM CMD STR
0461 23                 INX     H       ;
                ;
0462 FE3F               CPI     '?'     ; ALLOW AMBIG REFERENCE CHAR
0464 C27404             JNZ     GETN20  ;CONTINUE IF NOT
0467 3AE707             LDA     RWFLAG  ;GET WRITE FLAG
046A FE00               CPI     0       ;IS IT A READ OPERATION?
046C 3E3F               MVI     A,'?'   ;RESTORE A IN CASE ALLOWED
046E CA7F04             JZ      GETNA1  ;YES, GO STORE CHAR
0471 C39504             JMP     GETNA3  ;ERROR,MOVE POINTER TO END
                ;
0474            GETN20: DS      0       ;
0474 FE2A               CPI     '*'     ; FILL REST WITH ?
0476 CA8604             JZ      GETNA2  ;
                ;
0479 CD9C04             CALL    VALCHR  ; TEST FOR ALLOWED CHAR IN NAME
047C DA9504             JC      GETNA3  ;
                ;
047F 12         GETNA1: STAX    D       ; STORE CHAR IN TFCB
0480 13                 INX     D       ;
0481 0D                 DCR     C       ; CHECK NAME SIZE
0482 C26004             JNZ     GETNAM  ;
0485 C9                 RET             ;
                ;
0486            GETNA2: DS      0       ;FILL REST OF FIELD WITH '?'
0486 3AE707             LDA     RWFLAG  ;GET WRITE FLAG
0489 FE01               CPI     1       ;IS IT A WRITE?
048B CA9504             JZ      GETNA3  ;YES, '*' IS ILLEGAL, EXIT
048E 3E3F               MVI     A,'?'   ;FILL WITH '?'
0490 0600               MVI     B,0     ;
0492 C3D204             JMP     FILLB   ;
                ;
0495 13         GETNA3: INX     D       ; MOVE FCB PTR TO END OF FIELD
0496 0D                 DCR     C       ;
0497 C29504             JNZ     GETNA3  ;
049A 2B                 DCX     H       ;
049B C9                 RET             ;
                ;
                ; TEST FOR VALID CHAR IN  NAME FIELD
                ; RETURN WITH CARRY SET IF INVALID.
                ;
049C            VALCHR: DS      0       ;
                ;
049C FE0D               CPI     CR      ;IS IT 'RETURN'?
049E 3F                 CMC             ;COMPLEMENT CARRY
049F C8                 RZ              ;RETURN IF CR
04A0 FE2A               CPI     '*'
04A2 3F                 CMC
04A3 C8                 RZ              ;TRIED RC AND STILL NG 07SEP78
                ;
04A4 FE2C               CPI     ','
04A6 3F                 CMC
04A7 C8                 RZ              ;RETURN IF COMMA (15SEP78)
04A8 FE2E               CPI     '.'
04AA 3F                 CMC
04AB C8                 RZ
04AC FE20               CPI     ' '
04AE D8                 RC
04AF FE5F               CPI     '^'+1
04B1 3F                 CMC
04B2 D8                 RC
04B3 FE3A               CPI     ':'
04B5 3F                 CMC
04B6 D0                 RNC
04B7 FE40               CPI     '@'
04B9 C9                 RET

                ;
                ;
                ; TEST FOR VALID FILENAME TERMINATOR CHAR
                ; RETURN WITH CARRY SET IF INVALID.

04BA FE20       TERMT:  CPI     ' '
04BC C8                 RZ
04BD FE2C               CPI     ','
04BF C8                 RZ
04C0 FE0D               CPI     CR
04C2 C8                 RZ
04C3 FE3B               CPI     ';'
04C5 C8                 RZ
04C6 37                 STC
04C7 C9                 RET
                ; SKIP SPACES IN CMD STRING
```

```
04C8 3E20       SKIPS:  MVI     A,' '
04CA BE         SKIPS1: CMP     M       ;WAS CPI, FIXED 04SEP78,HOLLIDAY
04CB C0                 RNZ
04CC 23                 INX     H
04CD C3CA04             JMP     SKIPS1

                ; FILL BLOCK WITH VALUE
                ; ENTER WITH:
                ; A  = VALUE FOR FILL
                ; DE = START OF BLOCK
                ; BC = LENGTH OF BLOCK

04D0 3E00       CLRB:   MVI     A,0
04D2 04         FILLB:  INR     B
04D3 05                 DCR     B
04D4 C2DA04             JNZ     FILLB1
04D7 0C                 INR     C
04D8 0D                 DCR     C
04D9 C8                 RZ
04DA 12         FILLB1: STAX    D
04DB 13                 INX     D
04DC 0B                 DCX     B
04DD C3D204             JMP     FILLB
                ;
                ;END OF MAKEFCB.LIB
                ;
                ;
                ;----- MOVEIT ----- 9/24/78
                ; MOVES DATA FROM SOURCE TO DESTINATION
                ; ENTER WITH:
                ;       BC = LENGTH OR NUMBER OF BYTES TO MOVE
                ;       DE = SOURCE OR PICKUP ADDRESS
                ;       HL = TARGET OR DESTINATION ADDRESS
                ;
04E0            MOVEIT: DS   0          ;
04E0 C5D5E5             PUSH B! PUSH D! PUSH H  ;SAVE REGISTERS
04E3 1A         MOV10:  LDAX D          ;GET BYTE TO MOVE
04E4 77                 MOV  M,A        ;MOVE IT TO DESTINATION
04E5 13                 INX  D          ;NEXT SOURCE ADDRESS
04E6 23                 INX  H          ;NEXT DESTINATION ADDRESS
04E7 0B                 DCX  B          ;LENGTH=LENGTH-1
04E8 78                 MOV  A,B        ;PUT LENGTH IN A
04E9 B1                 ORA  C          ;TEST LENGTH FOR ZERO
04EA C2E304             JNZ  MOV10      ;LOOP UNTIL LENGTH=ZERO
04ED E1D1C1             POP H! POP D! POP B     ;RESTORE REGISTERS
04F0 C9                 RET             ;RETURN
                ;
                ;END OF MOVEIT
                ;
                ;----- PRNT -----
                ; PRINT STRING UNTIL BYTE IN 'A' ENCOUNTERED
                ; A  = BYTE TO STOP ON
                ; BC = ADDRESS TO PRINT FROM
                ;
04F1            PRNT:   DS   0          ;
04F1 E5                 PUSH H          ;SAVE HL
04F2 67                 MOV  H,A        ;SAVE BYTE TO FIND IN H
04F3 0A         PRNT10: LDAX B          ;GET DATA FROM ADDR IN BC
04F4 BC                 CMP  H          ;IS IT = TO BYTE REQUESTED ?
04F5 CAFF04             JZ   PRNT50     ;YES, GO TO EXIT
04F8 CD0F01             CALL OUT1       ;OUTPUT TO CONSOLE
04FB 03                 INX  B          ;INCREMENT TO GET NEXT BYTE
04FC C3F304             JMP  PRNT10     ;LOOP UNTIL BYTE FOUND
04FF E1         PRNT50: POP  H          ;RESTORE HL
0500 C9                 RET             ;RETURN
                ;
                ;----- PRNTM ----- 9/78
                ; PRINTS MESSAGE (OR STRING) UNTIL 'MEND' SENTINEL
                ; ENTER WITH MESSAGE ADDRESS IN BC
                ;
0501            PRNTM:  DS   0          ;
0501 3EFF               MVI  A,MEND     ;PUT BYTE TO STOP ON IN A
0503 CDF104             CALL PRNT       ;PRINT UNTIL 'MEND' FOUND
0506 C9                 RET             ;RETURN
                ;
                ;----- PRNTX ----- 9/78
                ; PRINT FOR LENGTH IN BC (MAX LNGTH=32767)
                ; BC = LENGTH OF STRING TO PRINT
                ; DE = ADDRESS TO START PRINTING FROM (BUFFER ADDR)
                ;
0507            PRNTX:  DS   0          ;
0507 78                 MOV  A,B        ;PUT LSB OF LENGTH IN A
0508 B1                 ORA  C          ;OR WITH MSB TO CHECK LENGTH
0509 CA1705             JZ   PRX50      ;GO TO EXIT IF LENGTH=0
050C 1A         PRX10:  LDAX D          ;GET BYTE AT ADDR IN DE
050D CD0F01             CALL OUT1       ;OUTPUT BYTE TO CONSOLE
0510 13                 INX  D          ;INCREMENT PICKUP ADDRESS
0511 0B                 DCX  B          ;DECREMENT LENGTH
0512 78                 MOV  A,B        ;PUT LSB OF LENGTH IN A
0513 B1                 ORA  C          ;IOR WITH MSB TO CHECK IF DONE
0514 C20C05             JNZ  PRX10      ;NOT DONE, LOOP UNTIL LENGTH=0
0517 C9         PRX50:  RET             ;RETURN
                ;
                ;END OF PRNTX.LIB
                ;
                ;----- PUSHM ----- 5/78
                ; SAVES ALL REGISTERS
                ;
0518 E3         PUSHM:  XTHL            ;SAVE HL, GET CALL LOC IN HL
0519 D5                 PUSH D          ;SAVE DE
051A C5                 PUSH B          ;SAVE BC
051B F5                 PUSH PSW        ;SAVE A AND FLAGS
051C E9                 PCHL            ;RETURN TO ADDR IN HL
                ;
                ;----- POPEM ----- 5/78
                ; RESTORES ALL REGISTERS
                ;
051D E1         POPEM:  POP  H          ;SAVE CALLING LOC IN HL
051E F1                 POP  PSW        ;RESTORE A AND FLAGS
051F C1                 POP  B          ;RESTORE BC
0520 D1                 POP  D          ;RESTORE DE
0521 E3                 XTHL            ;POP CALL LOC ON STACK,RESTORE HL
0522 C9                 RET             ;RETURN
                ;
                ;----- PUTBYTE -----
                ; PUTS A BYTE IN 'A' ONTO DISK (OR CPM BUFFER)
                ; SETUP 'FCB2' BEFORE ENTRY
                ;
0523            PUTBYTE:DS   0          ;
0523 CD1805             CALL PUSHM      ;SAVE REGISTERS
0526 2AE807             LHLD SECBP      ;GET SECTOR BUFFER POINTER
0529 77                 MOV  M,A        ;STORE BYTE IN BUFFER
052A 23                 INX  H          ;INCREMENT BUFFER POINTER
052B 22E807             SHLD SECBP      ;RESTORE NEW POINTER
052E 3A8607             LDA  BCNT       ;GET BYTE COUNTER
0531 3D                 DCR  A          ;COUNT=COUNT-1
0532 328607             STA  BCNT       ;RESTORE NEW COUNT
0535 C24D05             JNZ  PUT100     ;GO TO EXIT IF BUFFR NOT FULL
                ;
0538 110B08             LXI  D,FCB2     ;ADDRESS OF FILE CONTROL BLOCK
053B CD5606             CALL WRITE      ;WRITE NEXT RECORD
053E C24705             JNZ  PUT20      ;CONTINUE IF NO ERROR
0541 CD5D05             CALL SENDERR    ;WRITE ERROR
0544 C34D05             JMP  PUT100     ;GO TO EXIT AFTER WRITE ERR
                ;
0547            PUT20:  DS   0          ;
0547 CDC203             CALL INITIAL3   ;RESET SECTOR BUFFER POINTERS
054A CDBC02             CALL BLANKSB    ;STORE BLANKS IN SECTOR BUFFR
                ;
054D            PUT100: DS   0          ;
054D CD1D05             CALL POPEM      ;RESTORE REGISTERS
0550 C9                 RET             ;RETURN
                ;
                ;END OF PUTBYTE
                ;
                ;----- SAVEIN ----- 9/23/78
                ; SAVES KEYBOARD INPUT BYTE IN BUFFER.
                ;
0551            SAVEIN: DS   0          ;
0551 2ADD07             LHLD KINBUFP    ;GET CURRENT POINTER TO INPUT
0554 77                 MOV  M,A        ;SAVE/STORE A IN MEM
0555 23                 INX  H          ;NEXT ADDRESS
0556 22DD07             SHLD KINBUFP    ;SAVE NEW POINTER
0559 CDBA05             CALL SUMCHAR1   ;INCREMENT KEYBD INPUT COUNTER
055C C9                 RET             ;RETURN
                ;
                ;END OF SAVEIN.LIB
                ;
                ;----- SENDERR -----
                ; GENERAL ERROR MESSAGE
055D            SENDERR:DS   0          ;
055D 016807             LXI  B,MSGERR   ;GET 'ERROR..' MSG LOCATION
0560 CD0105             CALL PRNTM      ;SEND MESSAGE
0563 C9                 RET             ;RETURN
                ;
                ;----- SEND2 -----
0564            SEND2:  DS   0          ;
0564 0641               MVI  B,'A'      ;PRESET TO DRIVE 'A'
0566 3AEA07             LDA  FCB1       ;GET FIRST BYTE OF READ FCB
0569 FE00               CPI  0          ;IS IT ZERO?
056B CA7105             JZ   SEND2A     ;YES, GO STORE 'A'
056E C640               ADI  '@'        ;NO, ADD ALPHA BIAS
0570 47                 MOV  B,A        ;MOVE IT TO B FOR STORE
0571 78         SEND2A: MOV  A,B        ;PUT CHAR IN A
0572 32E906             STA  MSG2A      ;STORE DRIVE IN MESSAGE
0575 01D306             LXI  B,MSG2     ;GET MESSAGE LOC
0578 CD0105             CALL PRNTM      ;PRINT MESSAGE
057B C9                 RET             ;RETURN
                ;
                ;----- SETFCBS -----
                ; SETS UP FCB'S FROM 'INNAME' AND 'OUTNAME'
                ;
057C 00         SETFCBS:DB   0          ;
057D 213208             LXI  H,INNAME1  ;GET NAME OF FILE TO READ
0580 11EA07             LXI  D,FCB1     ;ADDRESS OF FCB TO STORE
0583 AF                 XRA  A          ;A=0 FOR READ OPERATION
0584 CD1804             CALL MAKEFCB    ;BUILD FCB FROM NAME
0587 D8                 RC              ;RETURN WITH CARRY SET IF ERROR
0588 213F08             LXI  H,OUTNAME1 ;GET FILE NAME TO WRITE
058B 110B08             LXI  D,FCB2     ;GET ADDRESS OF WRITE FCB
058E 3E01               MVI  A,1        ;SET A=1 FOR WRITE
0590 CD1804             CALL MAKEFCB    ;SETUP FCB2 FOR WRITE
0593 D8                 RC              ;CARRY SET IF ANY ERROR
0594 C9                 RET             ;NORMAL RETURN
                ;
                ;
                ;----- SETKIN ----- 9/23/78
                ; SET/RESET KEYBOARD INPUT BUFFER
                ;
0595            SETKIN: DS   0          ;
0595 218D07             LXI  H,KINBUF   ;GET ADDRESS OF KEYBD INPUT BFR
0598 22DD07             SHLD KINBUFP    ;SAVE/RESET INPUT POINTER
059B 015000             LXI  B,SIZEKIN  ;GET SIZE OF BUFFER
059E 3E20               MVI  A,' '      ;GET A BLANK FOR FILL
05A0 CD5303             CALL FILL       ;BLANK THE KEYBD INPUT BUFFR
05A3 AF                 XRA  A          ;GET ZERO IN A
05A4 328B07             STA  CHARCNT    ;RESET CHARACTER COUNTER
05A7 C9                 RET             ;RETURN
                ;
                ;END OF SETKIN.LIB
                ;
                ;----- STOREP2 -----
                ; STORE CHARACTER IN TEXT BUFFER POINTED TO BY
                ; P2, AND SET P3=ENDOFTEXT
                ;
05A8            STOREP2:DS   0          ;
05A8 2AE307             LHLD P2         ;GET CURSUR POSITION
05AB 77                 MOV  M,A        ;STORE BYTE IN BUFFER
05AC 23                 INX  H          ;INCREMENT TO NEXT POSITION
05AD 22E307             SHLD P2         ;SAVE NEW POINTER
05B0 2AE507             LHLD P3         ;GET END OF TEXT POINTER
05B3 23                 INX  H          ;INCREMENT TO NEXT ADDR FOR END
05B4 22E507             SHLD P3         ;SAVE END OF TEXT POINTER
05B7 36FF               MVI  M,ENDTEXT  ;STORE END SENTINEL AT END
05B9 C9                 RET             ;RETURN
                ;
                ;----- SUMCHAR1 -----
                ; INCREMENT KEYBD INPUT CHAR COUNTER (CHARCNT)
                ;
05BA            SUMCHAR1:DS  0  ;
05BA F5                 PUSH PSW        ;SAVE A AND FLAGS
05BB 3A8B07             LDA  CHARCNT    ;GET CURRENT COUNTER VALUE
05BE 3C                 INR  A          ;INCREMENT
05BF 328B07             STA  CHARCNT    ;RESTORE NEW COUNTER
05C2 F1                 POP  PSW        ;RESTORE A AND FLAGS
05C3 C9                 RET             ;RETURN
```

```
                ;
                ;----- WAITKEY -----
                ; WAIT FOR ANY KEY TO BE PRESSED
                ;
05C4            WAITKEY:DS  0           ;
05C4 CD0601             CALL KEYIN      ;WAIT FOR ANY KEY
05C7 C9                 RET             ;RETURN FROM WAITKEY
                ;
                ;----- WRITE2 -----
                ; OPEN FILE DEFINED BY FCB2 AND WRITE FILE
                ; SETUP FCB2 BEFORE ENTRY
                ; SETUP 'BUFF2' ADDRESS BEFORE ENTRY
                ; USER MUST CLOSE FCB2 AFTER RETURN FROM HERE
                ; RETURNS WITH CARRY SET IF ANY ERROR
                ; WRITE STOPS WHEN 2 SUCCESSIVE 'ENDTEXT' BYTES FOUND
                ;
                WRITE2:
05C8 110B08             LXI  D,FCB2     ;GET FCB ADDRESS
05CB CD5006             CALL DELETE     ;DELETE ANY FILE WITH SAME NAME
05CE 110B08             LXI  D,FCB2     ;GET FCB ADDRESS
05D1 CD4806             CALL CREATE     ;CREATE NEW FILE
05D4 CA0906             JZ   WR270      ;GO TO EXIT IF ANY ERROR
                ;
05D7 110B08             LXI  D,FCB2     ;GET FCB ADDRESS
05DA CD3806             CALL OPEN       ;OPEN FILE
05DD CA0906             JZ   WR270      ;EXIT IF ANY ERROR
                ;
05E0 AF                 XRA  A          ;GET A ZERO
05E1 322B08             STA  FCB2+32    ;SET NEXT RECORD=FIRST ONE
05E4 CDC203             CALL INITIAL3   ;RESET BYTE AND SECTOR COUNTERS
05E7 CDBC02             CALL BLANKSB    ;BLANK SECTOR BUFFER
                ;
05EA            WR240:  DS   0          ;
05EA 2A8907             LHLD BUFF2      ;GET BUFFER POINTER
05ED 7E                 MOV  A,M        ;GET BYTE FROM BUFFER
05EE 23                 INX  H          ;NEXT ADDRESS
05EF 228907             SHLD BUFF2      ;INCREMENT POINTER
05F2 CD2305             CALL PUTBYTE    ;PUT BYTE ON DISK
05F5 CDC802             CALL CHECKEND   ;SEE IF AT END OF TEXT (2 FF'S)
05F8 DAFE05             JC   WR250      ;AT END, GO TO EXIT
05FB C3EA05             JMP  WR240      ;LOOP UNTIL END SENTINEL
                ;
05FE            WR250:  DS   0          ;NORMAL EXIT, END OF FILE OR END
05FE 110B08             LXI  D,FCB2     ;GET ADDRESS OF FCB          SENT
0601 CD5606             CALL WRITE      ;WRITE LEFTOVER RECORD
0604 CA0906             JZ   WR270      ;EXIT IF ANY ERROR
0607 B7                 ORA  A          ;CLEAR CARRY FOR NORMAL EXIT
0608 C9                 RET             ;RETURN, EXIT 1
                ;
0609            WR270:  DS   0          ;ERROR EXIT
0609 37                 STC             ;SET CARRY FOR ERROR EXIT
060A C9                 RET             ;RETURN
                ;
                ;END OF WRITE2
                ;
                ;
                ;KEYPRIM.LIB
                ;18SEP78
                ;-------------------------------------------------;
                ;           I/O SECTION, PRIMITIVES               ;
                ;-------------------------------------------------;
                ; ALL ENTRIES SHOULD BE FROM JUMP TABLE ONLY
                ; USE JUMP TABLE LABELS TO CALL, NOT THESE LABELS
                ;
                ;----- KEYINX ----- 9/78
                ; INPUTS FROM KEYBOARD AND WAITS FOR KEY PRESSED
                ;
060B            KEYINX: DS   0          ;ENTRY FROM JUMP TABLE
                        IF   CPMKEY     ;IF CPM KEYBOARD
060B CD0C01             CALL INKD       ;GET STATUS,WAIT FOR KEY
060E C9                 RET             ;RETURN
                        ENDIF           ;END OF CPM KEYBD INPUT
                ;
                        IF   ALONE      ;IF STAND ALONE KEYBD
                KEY10:  DS   0          ;
                        INR  A          ;WAIT DELAY FOR POLY VTI PORT
                        JNZ  KEY10      ;WAIT
                        CALL INKS       ;GET KEYBOARD STATUS
                        JNZ  KEY10      ;LOOP UNTIL KEY PRESSED
                        CALL INKD       ;GET KEYBOARD DATA
                        RET             ;RETURN
                        ENDIF           ;END OF STAND ALONE KEYBD
                ;
                ;
                ;----- INKSX -----
                ; INPUT CONSOLE (KEYBOARD) STATUS
                ; RETURNS WITH ZERO BIT SET IF KEY PRESSED
                ;
                        IF   CPMKEY     ;IF USING CPM KEYBOARD
060F            INKSX:  DS   0          ;ENTRY FROM JUMP TABLE
060F 0E0B               MVI  C,CONSTAT  ;GET 'READ CONSOLE STATUS' CMD
0611 CD1E01             CALL BDOS       ;CALL CPM TO READ STATUS
0614 0F                 RRC             ;ROTATE RIGHT TO CHECK LSB
0615 D21D06             JNC  INKS30     ;EXIT IF NO INPUT
0618 3E01               MVI  A,1        ;PUT ANYTHING IN A,
061A FE01               CPI  1          ;SO CAN TEST AND SET ZERO BIT
061C C9                 RET     ;EXIT   ;EXIT WITH ZERO BIT SET
061D 3E00       INKS30: MVI  A,0        ;PUT ANYTHING IN A,
061F FE01               CPI  1          ;SO CAN TEST AND RESET ZERO BIT
0621 C9                 RET     ;EXIT   ;RETURN WITH ZERO BIT RESET
                        ENDIF           ;END OF CPM KEYBD VERSION
                ;
                ; STAND ALONE KEYBD STATUS INPUT (ONLY IF ALONE=TRUE)
                        IF   ALONE      ;IF STAND ALONE KEYBD VERSION
                INKSX:  DS   0          ;ENTRY FROM JUMP TABLE (INKS)
                        IN   KSTAT      ;GET KEYBOARD STATUS
                        DB   KRDAI      ;CMA OR NOP (INVERTS IT)
                        ANI  KRDA       ;MASK DATA AVAILABLE BIT
                        RET             ;RETURN
                        ENDIF           ;
                ;
                ;
                ;----- INKDX -----
                ; INPUT CONSOLE DATA
                ; IF CPM KEYBOARD VERSION, ALSO ECHOS INPUT
                ;
                        IF   CPMKEY     ;IF USING CPM KEYBOARD
0622            INKDX:  DS   0          ;ENTRY FROM JUMP TABLE (INKD)
0622 0E01               MVI  C,CONS     ;READ CONSOLE COMMAND
0624 CD1E01             CALL BDOS       ;CALL CPM TO READ AND ECHO
0627 E67F               ANI  7FH        ;STRIP OFF MSB IF NOT DONE
0629 C9                 RET             ;RETURN
                        ENDIF           ;END OF CPMKEY VERSION
                ;
                        IF   ALONE      ;IF STAND ALONE VERSION
                INKDX:  DS   0          ;ENTRY FROM JUMP TABLE
                        IN   KDATA      ;GET KEYBOARD DATA
                        ANI  7FH        ;STRIP OFF MSB
                        RET             ;
                        ENDIF           ;END OF STAND ALONE KEYBD
                ;
                ;
                ;----- OUT1X -----
                ; OUTPUT BYTE IN 'A' TO CONSOLE VIA CPM (BIOS)
                ;
062A            OUT1X:  DS   0          ;
062A CD1805             CALL PUSHM      ;SAVE ALL REGISTERS
062D 0E02               MVI  C,TYPEF    ;GET CPM COMMAND FOR CONS OUT
062F 5F                 MOV  E,A        ;PUT CHAR IN E FOR CPM
0630 CD1E01             CALL BDOS       ;CALL CPM TO OUTPUT BYTE
0633 CD1D05             CALL POPEM      ;RESTORE ALL REGISTERS
0636 C9                 RET
                ;
                ;----- OUT2X -----
                ; OUTPUT BYTE IN 'A' TO VIDEO SCREEN
                ; NOT IMPLEMENTED YET
                ;
0637 C9         OUT2X:  RET             ;NOT IMPLEMENTED
                ;
                ;END OF KEYPRIM.LIB
                ;
                ;FILEPRIM.LIB
                ;18SEP78
                ;-------------------------------------------;
                ;          FILE HANDLING PRIMITIVES         ;
                ;-------------------------------------------;
                ;
                ;----- OPEN -----
                ; OPEN FILE
                ; DE = FCB ADDRESS
                ; RETURNS WITH ZERO BIT SET IF FILE NOT FOUND
                ;
0638            OPEN:   DS   0          ;
0638 0E0F               MVI  C,OPENF    ;GET 'OPEN FILE' COMMAND IN C
063A CD1E01             CALL BDOS       ;CALL DISK OPERATING SYSTEM
063D FEFF               CPI  255        ;SEE IF NOT FOUND
063F C9                 RET
                ;
                ;----- CLOSE -----
                ; CLOSE FILE
                ; DE = FCB ADDRESS
                ; RETURNS WITH ZERO BIT SET IF FILE NOT PRESENT
                ;
0640            CLOSE:  DS   0          ;
0640 0E10               MVI  C,CLOSEF   ;CLOSE FILE COMMAND TO C
0642 CD1E01             CALL BDOS       ;CALL DOS
0645 FEFF               CPI  255        ;SEE IF NOT PRESENT
0647 C9                 RET
                ;
                ;----- CREATE -----
                ; CREATES (MAKES) A NEW FILE ON DISK
                ; DE = FCB ADDRESS
                ; RETURNS WITH ZERO BIT SET IF NO SPACE AVAILABLE
                ;
0648            CREATE: DS   0          ;
0648 0E16               MVI  C,CREATEF  ;CREATE FILE COMMAND TO 'C'
064A CD1E01             CALL BDOS       ;CALL DOS
064D FEFF               CPI  255        ;IF 255, NO SPACE AVAIL
064F C9                 RET
                ;
                ;----- DELETE -----
                ; DELETE FILE NAME FROM CPM DISK DIRECTORY
                ; DE = FCB ADDRESS
                ;
0650            DELETE: DS   0          ;
0650 0E13               MVI  C,DELETEF  ;DELETE COMMAND TO 'C'
0652 CD1E01             CALL BDOS       ;CALL DOS
0655 C9                 RET             ;RETURN, NO PARAMETERS FROM DOS
                ;
                ;----- WRITE -----
                ; WRITE SECTOR (NEXT RECORD)
                ; DE = FCB ADDRESS
                ; RETURNS WITH ZERO BIT SET IF ANY ERROR
                ;
0656            WRITE:  DS   0          ;
0656 0E15               MVI  C,WRITEF   ;WRITE COMMAND
0658 CD1E01             CALL BDOS       ;CALL DOS
065B FE01               CPI  1          ;TEST FOR ERROR
065D C8                 RZ              ;ERROR EXTENDING FILE
065E FE02               CPI  2          ;TEST FOR ERROR
0660 C8                 RZ              ;ERROR,END OF DISK DATA
0661 FEFF               CPI  255        ;TEST FOR ERROR
0663 C8                 RZ              ;ERROR, NO DIRECTORY SPACE
0664 C9                 RET             ;NORMAL RETURN
                ;
                ;----- READ -----
                ; READ SECTOR (NEXT RECORD)
                ; DE = FCB ADDRESS
                ; RETURNS WITH ZERO BIT SET IF ANY ERROR OR IF EOF
                ;
0665            READ:   DS   0          ;
0665 0E14               MVI  C,READF    ;GET READ FILE COMMAND
0667 CD1E01             CALL BDOS       ;READ USING DOS
066A FE01               CPI  1          ;IF 1 THEN END OF FILE
066C C8                 RZ              ;RETURN
066D FE02               CPI  2          ;IF 2 THEN UNWRITTEN DATA
066F C8                 RZ              ;RETURN
0670 C9                 RET             ;NORMAL RETURN
                ;
                ;END OF FILEPRIM.LIB
                ;
                ;-----------------------------------------------
                ;-              MESSAGES AREA
                ;-----------------------------------------------
                ;
                ;----- SIGN-ON MESSAGE -----
0671 0D0A       MSG1:   DB  CR,LF
```

```
0673 434F4E5650          DB   'CONVPATB.ASM '
0680 563032E372CVERSION:DB   'V0.7,28SEP78'
068C 2020562E48          DB   '  P.HOLLIDAY'
0698 0D0A0D0A            DB   CR,LF,CR,LF
069C 434F4E5645          DB   'CONVERTS TINY BASIC FILE TO ASCII (PATB OR TINYD
06D0 0D0A                DB   CR,LF
06D2 FF                  DB   MEND
                    ;
06D3 0D0A4D4F55MSG2:     DB   CR,LF,'MOUNT DISK IN DRIVE '
06E9 412C204849MSG2A:    DB   'A',', HIT <RETURN> ',MEND
                    ;
06FA 0D0A524541MSG3:     DB   CR,LF,'READING FILE..',MEND
070B 0D0A575249MSG4:     DB   CR,LF,'WRITING FILE..',MEND
071C 0D0A46494EMSG5:     DB   CR,LF,'FINISHED',MEND
0727 0D0A494E56MSG6:     DB   CR,LF,'INVALID NAME..',MEND
0738 0D6A434F4EMSG7:     DB   CR,LF,'CONVERTING FILE..',MEND
074C 0D0A46494CMSG8:     DB   CR,LF,'FILE NAME ? ',MEND
075B 0D0A524541MSG22:    DB   CR,LF,'READ ERROR',MEND
                    ;
0768 0D0A455252MSGERR:   DB   CR,LF,'ERROR.....',MEND
0775 46494C4520MSGNF:    DB   'FILE NOT FOUND, ',MEND
                    ;
                    ;
                    ;----- CONSTANTS AND STORAGE AREA -----
0786 00             BCNT:    DB   0          ;BYTES PER SECTOR COUNTER
0787 4C08           BUFF1:   DW   TEXTBUF    ;SOURCE ADDRESS POINTER
0789 4C08           BUFF2:   DW   TEXTBUF    ;DESTINATION ADDR POINTER
078B 00             CHARCNT:DB   0          ;INPUT1 CHARACTER COUNTER
078C 00             GETFLAG:DB   0          ;GETBYTE FLAG, 1=THRU ONCE
078D                KINBUF:  DS   SIZEKIN   ;ACTUAL KEYBOARD INPUT BUFFR
07DD 8D07           KINBUFP:DW   KINBUF     ;POINTER TO CURRENT INPUT
07DF 4C08           MAXMEM:  DW   TEXTBUF    ;LAST AVAILABLE MEM ADDRESS
07E1 4C08           P1:      DW   TEXTBUF    ;POINTER TO START OF TEXT BFR
07E3 4C08           P2:      DW   TEXTBUF    ;POINTER TO CURRENT POS IN TEXT
07E5 4C08           P3:      DW   TEXTBUF    ;POINTER TO END OF INPUT TEXT
07E7 00             RWFLAG:  DB   0          ;READ/WRITE FLAG =1 IF WRITE
07E8 6000           SECBP:   DW   SECBUF     ;POINTER TO SECTOR BUFFER
                    ;
                    ;----- FILE CONTROL BLOCKS (FCB) -----
07EA                FCB1:    DS   33         ;FIRST FCB (READ/INPUT)
060B                FCB2:    DS   33         ;SECOND FCB (WRITE/OUTPUT)
                    ;
082C 544249         RFT:     DB   'TBI'      ;READ FILE TYPE
082F 415343         WFT:     DB   'ASC'      ;WRITE FILE TYPE
0832 54494E5954INNAME1:DB    'TINYTREK.TBI '     ;NAME BUFFER
083F 54494E5954OUTNAME1:DB   'TINYTREK.ASC '     ;NAME BUFFER
                    ;
                    ;
084C                ZZEND:   DS   0          ;END OF PROGRAM/NEXT LOC
084C                TEXTBUF:DS   0          ;START OF INPUT BUFFER
084C                         END
```

A>

A>^Q

>?

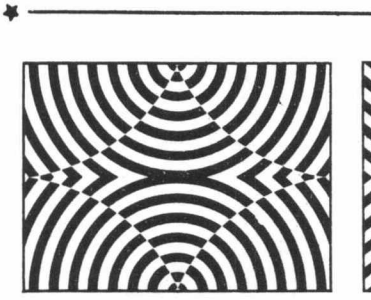
A>DUMP2

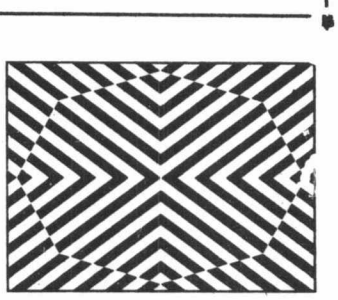
```
*CONVPATB.COM
0100  C3 2A 01 C3 7F 01 C3 DB  06 C3 0F 06 C3 22 06 C3  .*...... .....".
0110  2A 06 C3 2A 06 C3 00 CC  C3 00 C0 C3 00 00 C3 05  *..*.... ........
0120  00 C3 00 C0 C3 00 C0 C3  00 C0 CD AF 03 CD EF 02  ........ ........
0130  01 71 06 CD 01 05 3A 80  00 FE 02 DA 44 01 CD F0  .q....:. ....D...
0140  03 C3 7F 01 01 4C 07 CD  01 05 CD CE 03 FE 03 CA  .....L.. ........
0150  B9 02 CD 36 03 01 0D 00  11 8D 07 21 32 08 CD E0  ...6.... ...!2...
0160  04 21 3F 08 CD E0 04 CD  7C 05 D2 7C 01 CD 5D 05  .!?..... |..|..].
0170  01 27 07 CD 01 05 CD C4  05 C3 44 01 CD 02 04 CD  .'...... ..D.....
0180  64 05 CD 06 01 FE 0D C2  7F 01 01 FA 06 CD 01 05  d....... ........
0190  01 08 00 11 EB 07 CD 07  05 11 EA 07 CD 38 06 C2  ........ .....8..
01A0  B1 01 CD 5D 05 01 75 07  CD 01 05 CD C4 05 C3 B0  ...]..u. ........
01B0  02 CD C2 03 21 4C 08 22  E3 07 AF 32 0A 08 32 8C  ....!L." ...2..2.
01C0  07 CD BC 02 2A E5 07 CD  DA 02 DA EC 01 CD 5E 03  ....*... ......^.
01D0  DA EC 01 CD A8 05 FE FF  C2 C4 01 CD 5E 03 DA EC  ........ ....^...
01E0  01 CD A8 05 FE FF CA EC  01 C3 C4 01 2A E1 07 22  ........ ....*..*
01F0  87 07 2A E5 07 23 36 FF  23 22 89 07 01 38 07 CD  ..*..#6. #"...8..
0200  01 05 2A 87 07 23 23 22  87 07 2A 87 07 CD C8 02  ..*..##" ..*.....
0210  DA 73 02 4E 23 46 23 22  87 07 CD F5 02 2A 89 07  .s.N#F#" .....*..
0220  CD DA 02 DA 73 02 3A 34  03 77 23 3A 33 03 77 23  ....s.:4 .w#:3.w#
0230  3A 32 03 77 23 3A 31 03  77 23 36 20 23 CD DA 02  :2.w#:1. w#6 #...
0240  DA 73 02 22 89 07 2A 87  07 CD C8 02 DA 73 02 7E  .s."..*. .....s.~
0250  23 22 87 07 2A 89 07 77  23 22 89 07 FE 0D CA 6A  #"..*..w #".....j
0260  02 CD DA 02 DA 73 02 C3  46 02 36 0A 23 22 89 07  .....s.. F.6.#"..
0270  C3 0A 02 2A 89 07 36 20  23 36 45 23 36 4E 23 36  ...*..6  #6E#6N#6
0280  44 23 36 FF 23 36 FF 01  0B 07 CD 01 05 2A E5 07  D#6.#6.. .....*..
0290  23 23 22 89 07 CD C8 05  DC 5D 05 11 EA 07 CD 40  ##"..... .]......@
02A0  06 CC 5D 05 11 0B 08 CD  40 06 CC 5D 05 C3 B0 02  ..]..... @..]....
02B0  01 1C 07 CD 01 05 CD 36  03 C3 1B 01 21 80 00 01  .......6 ....!...
02C0  80 00 3E 20 CD 53 03 C9  7E FE FF C2 D8 02 23 7E  ..> .S.. ~.....#~
02D0  2B FE FF C2 D8 02 37 C9  B7 C9 E5 EB 2A DF 07 7C  +.....7. ....*..|
02E0  BA C2 EC 02 7D BB D2 EC  02 C3 ED 02 B7 E1 C9 3E  ....}... .......>
02F0  04 CD 0F 01 C9 C5 AF 32  31 03 32 32 03 32 33 03  .......2 1.22.23.
0300  32 34 03 32 35 03 60 69  CD 28 03 32 31 03 CD 28  24.25.`i .(.21..(
0310  03 32 32 03 CD 28 03 32  33 03 CD 28 03 32 34 03  .22..(.2 3..(.24.
0320  CD 28 03 32 35 03 C1 C9  00 0E 0A CD 40 03 C6 30  .(.25... ....@..0
0330  C9 00 00 00 00 00 01 3D  03 CD 01 05 C9 0D 0A FF  .......= ........
0340  AF B9 37 C8 06 10 29 17  30 DA 4E 03 91 23 05 C2  ..7...). ..N..#..
0350  46 03 C9 77 23 0B 57 78  B1 7A C2 53 03 C9 3A 8C  F..w#.Wx .z.S..:.
0360  07 FE 01 CA 78 03 3E 01  32 8C 07 11 EA 07 CD 65  ....x.>. 2......e
0370  06 C2 78 03 F5 C3 99 03  2A E8 07 7E 23 22 E8 07  ..x..... *..~#"..
0380  47 3A 86 07 3D 32 86 07  78 C2 AD 03 F5 CD BC 02  G:..=2.. x.......
0390  11 EA 07 CD 65 06 C2 A9  03 FE 01 CA A4 03 01 5B  ....e... .......[
03A0  07 CD 01 05 F1 37 C3 AE  03 CD C2 03 F1 B7 C9 2A  .....7.. .......*
03B0  06 00 11 EC FF 19 22 DF  07 CD C0 03 CD C2 03 C9  ......". ........
03C0  AF C9 3E 80 32 86 07 21  80 00 22 E8 07 C9 CD 95  ..>.2..! ..".....
03D0  05 0E 00 CD 06 01 FE 03  CA EF 03 CD BA 05 CD 51  ........ .......Q
03E0  05 FE 0D CA E9 03 C3 D3  03 F5 3A 8B 07 4F F1 C9  ........ ..:..O..
03F0  01 10 00 11 5C 00 21 EA  07 CD E0 04 21 0B 08 CD  ....\.!. ....!...
0400  E0 04 01 03 00 11 2C 08  21 F3 07 CD E0 04 11 2F  ........ !....../
0410  08 21 14 08 CD E0 04 C9  E5 D5 32 E7 07 01 21 00  .!...... ..2...!.
0420  3E 00 CD D2 04 D1 D5 13  01 0B 00 3E 20 CD D2 04  >....... ...> ...
0430  D1 E1 CD C6 04 23 7E 2B  FE 3A C2 48 04 7E 23 23  .....#~+ .:.H.~##
0440  DE 40 D6 FE 5B 3F D8 12  13 0E 08 CD 60 04 7E 23  .@..[?.. ....`.~#
0450  FE 2E C2 5C 04 0E 03 CD  60 04 7E 23 CD BA 04 C9  ...\.... `.~#....
0460  7E 23 FE 3F C2 74 04 3A  E7 07 FE 00 3E 3F CA 7F  ~#.?.t.: ....>?..
0470  04 C3 95 04 FE 2A CA 86  04 CD 9C 04 DA 95 04 12  .....*.. ........
0480  13 0D C2 60 04 C9 3A E7  07 FE 01 CA 95 04 3E 3F  ...`..:. ......>?
0490  06 00 C3 D2 04 13 0D C2  95 04 2B C9 FE 0D 3F C8  ........ ..+...?.
04A0  FE 2A 3F C8 FE 2C 3F C8  FE 2E 3F C8 FE 20 D8 FE  .*?...?. ..?.. ..
04B0  5F 3F D8 FE 3A 3F D0 FE  40 C9 FE 20 C8 FE 2C C8  _?..:?.. @.. ..,.
04C0  FE 0D C8 FE 3B C8 37 C9  3E 20 BE C0 23 C3 CA 04  ....;.7. > ..#...
04D0  3E 00 04 05 C2 DA 04 0C  0D C8 12 13 0B C3 D2 04  >....... ........
04E0  C5 D5 E5 1A 77 13 23 0B  78 B1 C2 E3 04 E1 D1 C1  ....w.#. x.......
04F0  C9 E5 67 0A BC CA FF 04  CD 0F 01 03 C3 F3 04 E1  ..g..... ........
0500  C9 3E FF CD F1 04 C9 78  B1 CA 17 05 1A CD 0F 01  .>.....x ........
0510  13 0B 78 B1 C2 0C 05 C9  E3 D5 C5 F5 E9 E1 F1 C1  ..x..... ........
0520  D1 E3 C9 CD 18 05 2A E8  07 77 23 22 E8 07 3A 86  ......*. .w#"..:.
0530  07 3D 32 86 07 C2 4D 05  11 0B 08 CD 56 06 C2 47  .=2...M. ....V..G
0540  05 CD 5D 05 C3 4D 05 CD  C2 03 CD BC 02 CD 1D 05  ..]..M.. ........
0550  C9 2A DD 07 77 23 22 DD  07 CD BA 05 C9 01 68 07  .*..w#". ......h.
0560  CD 01 05 C9 06 41 3A EA  07 FE 00 CA 71 05 C6 40  .....A:. ....q..@
0570  47 78 32 E9 06 01 D3 06  CD 01 05 C9 00 21 32 08  Gx2..... .....!2.
0580  11 EA 07 AF CD 18 04 D8  21 3F 08 11 0B 08 3E 01  ........ !?....>.
0590  CD 18 04 D8 C9 21 8D 07  22 DD 07 01 50 00 3E 20  .....!.. "...P.>
05A0  CD 53 03 AF 32 8B 07 C9  2A E3 07 77 23 22 E3 07  .S..2... *..w#"..
05B0  2A E5 07 23 22 E5 07 36  FF C9 F5 3A 8B 07 3C 32  *..#"..6 ...:..<2
05C0  8B 07 F1 C9 CD 06 01 C9  11 0B 08 CD 50 06 11 0B  ........ ....P...
05D0  08 CD 48 06 CA 09 06 11  0B 08 CD 38 06 CA 09 06  ..H..... ...8....
05E0  AF 32 2B 08 CD C2 03 CD  BC 02 2A 89 07 7E 23 22  .2+..... ..*..~#"
05F0  89 07 CD 23 05 CD C8 02  DA FE 05 C3 EA 05 11 0B  ...#.... ........
0600  08 CD 56 06 CA 09 06 E7  C9 37 C9 CD 0C 01 C9 0E  ..V..... .7......
0610  0B CD 1E 01 0F D2 1D 06  3E 01 FE 01 C9 3E 00 FE  ........ >....>..
0620  01 C9 0E 01 CD 1E 01 E6  7F C9 CD 18 05 0E 02 5F  ........ .......
0630  CD 1E 01 CD 1D 05 C9 C9  0E 0F CD 1E 01 FE FF C9  ........ ........
0640  0E 10 CD 1E 01 FE FF C9  0E 16 CD 1E 01 FE FF C9  ........ ........
0650  0E 1_ CD 1E 01 C9 0E 15  CD 1E 01 FE 01 C8 FE 02  ........ ........
0660  C8 FE FF C8 C9 0E 14 CD  1E 01 FE 01 C8 FE 02 C8  ........ ........
0670  C9 0D 0A 43 4F 4E 56 50  41 54 42 2E 41 53 4D 20  ...CONVP ATB.ASM
0680  56 30 2E 37 2C 32 38 53  45 50 37 38 20 20 50 2E  V0.7,28S EP78  P.
0690  48 4F 4C 4C 49 44 41 59  0D 0A 0D 0A 43 4F 4E 56  HOLLIDAY ....CONV
06A0  45 52 54 53 20 54 49 4E  59 20 42 41 53 49 43 20  ERTS TIN Y BASIC
06B0  46 49 4C 45 20 54 4F 20  41 53 43 49 49 20 28 50  FILE TO  ASCII (P
06C0  41 54 42 20 4F 52 20 54  49 4E 59 44 49 53 4B 29  ATB OR T INYDISK)
06D0  0D 0A FF 0D 0A 4D 4F 55  4E 54 20 44 49 53 4B 20  .....MOU NT DISK
06E0  49 4E 20 44 52 49 56 45  20 41 2C 20 48 49 54 20  IN DRIVE  A, HIT
06F0  3C 52 45 54 55 52 4E 3E  20 FF 0D 0A 52 45 41 44  <RETURN>  ...READ
0700  49 4E 47 20 46 49 4C 45  2E 2E FF 0D 0A 57 52 49  ING FILE .....WRI
0710  54 49 4E 47 20 46 49 4C  45 2E 2E FF 0D 0A 46 49  TING FIL E.....FI
0720  4E 49 53 48 45 44 FF 0D  0A 49 4E 56 41 4C 49 44  NISHED.. .INVALID
0730  20 4E 41 4D 45 2E 2E FF  0D 0A 43 4F 4E 56 45 52  NAME... ..CONVER
0740  54 49 4E 47 20 46 49 4C  45 2E 2E FF 0D 0A 46 49  TING FIL E.....FI
0750  4C 45 20 4E 41 4D 45 20  3F 20 FF 0D 0A 52 45 41  LE NAME  ? ...REA
0760  44 20 45 52 52 4F 52 FF  0D 0A 45 52 52 4F 52 2E  D ERROR. ..ERROR.
0770  2E 2E 2E 2E FF 46 49 4C  45 20 4E 4F 54 20 46 4F  .....FIL E NOT FO
0780  55 4E 44 2C 20 FF 00 4C  08 4C 08 00 00 20 50 2E  UND, ..L .L... P.
0790  48 4F 4C 4C 49 44 41 59  0D 0A 0D 0A 43 4F 4E 56  HOLLIDAY ....CONV
07A0  45 52 54 53 20 54 49 4E  59 20 42 41 53 49 43 20  ERTS TIN Y BASIC
07B0  46 49 4C 45 20 54 4F 20  41 53 43 49 49 20 28 50  FILE TO  ASCII (P
07C0  41 54 42 20 4F 52 20 54  49 4E 59 44 49 53 4B 29  ATB OR T INYDISK)
07D0  0D 0A FF 0D 0A 4D 4F 55  4E 54 20 44 49 8D 07 4C  .....MOU NT DI..L
07E0  08 4C 08 4C 08 4C 08 00  80 00 2C 20 48 49 54 20  .L.L.L.. ..,_HIT
07F0  3C 52 45 54 55 52 4E 3E  20 FF 0D 0A 52 45 41 44  <RETURN>  ...READ
0800  49 4E 47 20 46 49 4C 45  2E 2E FF 0D 0A 57 52 49  ING FILE .....WRI
0810  54 49 4E 47 20 46 49 4C  45 2E 2E FF 0D 0A 46 49  TING FIL E.....FI
0820  4E 49 53 48 45 44 FF 0D  0A 49 4E 56 54 42 49 41  NISHED.. .INVTBIA
0830  53 43 54 49 4E 59 54 52  45 4B 2E 41 53 43 20 00  SCTINYTR EK.TBI T
0840  49 4E 59 54 52 45 4B 2E  41 53 43 20 00 00 00 00  INYTREK. ASC ....
0850  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0860  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0870  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
```

Continued from pg. 14

```
0063000  LVL     SET     0
0064000          ENDIF
0065000          RET
0066000          ENDM
0067000 ;
0068000 ; SAVE (BC.L), TO BE RESTORED BY RELOAD BC.L LATER
0069000 SAVE     MACRO   REG
0070000 BC.L     EQU     B
0071000          IF      REG-B
0072000          ERROR   'R'
0073000          DB      0,0
0074000          ENDIF
0075000          IFZ     REG-B
0076000          PUSH    B
0077000          PUSH    H
0078000          ENDIF
0079000          ENDM
```

END of Part 1. Continued next issue.





QUOLST

BY GARY D. GAUGLER
2276 Beaver Valley Road
Fairborn, Ohio 45324

QUOLST is a transient program for the Smoke Signal BFD-68 disc system. This transient will report the sector availability for a diskette in the specified drive. QUOLST reports the number of sectors used on the diskette for non-system files and the number of sectors used for system files (those with a '$' extension). QUOLST will then report the total number of sectors available on the diskette.

QUOLST is invoked by typing

QUOLST c/r

This will report information for the default drive 0. To use QUOLST for another drive, type

QUOLST,n c/r

where n is the number of the desired drive.

```
                        NAM    QUOLST

                * VERSION 1.0

                * (C) COPYRIGHT 1978 BY GARY D. GAUGLER
                * ALL RIGHTS RESERVED.
                *
                * REPRODUCTION FOR NON-COMMERCIAL PURPOSES
                * IS PERMITTED.
                *

                * TRANSIENT TO REPORT SECTOR USEAGE ON BFD-68
                *
                * REPORTS USEAGE OF SYSTEM FILES ($ EXTENSIONS)
                *                   USER FILES (NON-$ EXTENSIONS)
                * AND TOTAL SECTORS AVAILABLE.

                * EXTERNALS

20 76D0         FCB     EQU     $76D0
21 72A9         ZTYPDF  EQU     $72A9
22 7283         ZWARMS  EQU     $7283
23 72AF         ZOUTHA  EQU     $72AF
24 7286         ZOUTEF  EQU     $7286
25 72A6         ZOUTST  EQU     $72A6
26 7297         ZGNCHR  EQU     $7297

28              * COMMON EQUATES

30 7080         TCA     EQU     $7080       ORG OF TRANSIENT COMMAND AREA

32              * DFM LINKAGES

34 7780         DFMORG  EQU     $7780       BASE ADDRESS OF DFM

36 7780         ODFM    EQU     DFMORG+0    OPEN DFM
37 7783         CDFM    EQU     DFMORG+3    CLOSE DFM
38 7786         DFM     EQU     DFMORG+6    I/O REQUEST ENTRY

40              * FILE CONTROL BLOCK (FCB) INDICES

42 0000         XFC     EQU     0           DFM FUNCTION CODE
43 0001         XFS     EQU     1           ERROR STATUS
44 0002         XUN     EQU     2           UNIT NUMBER
45 0003         XFN     EQU     3           FILE NAME
46 0009         XFE     EQU     9           FILE EXTENSION
47 000C         XFT     EQU     12          FILE TYPE
48 000D         XFS     EQU     13          FILE STATUS
49 000E         XSFU    EQU     14          FIRST SECTOR USED
50 0010         XLSU    EQU     16          LAST SECTOR USED
51 0012         XSUC    EQU     18          SECTORS USED COUNT
52 001E         XCT     EQU     30          CURRENT TRACK NUMBER
53 001F         XCS     EQU     31          CURRENT SECTOR NUMBER
54 0026         XDB     EQU     38          START OF BUFFER

55 0026         XNT     EQU     XDB+0       NEXT TRACK NUMBER
56 0027         XNS     EQU     XDB+1       NEXT SECTOR NUMBER
57 0028         XPT     EQU     XDB+2       PREVIOUS TRACK NUMBER
58 0029         XPS     EQU     XDB+3       PREVIOUS SECTOR NUMBER
59 002A         XSOD    EQU     XDB+4       START OF DATA

61              * DFM FUNCTION CODES

63 0000         OFREE   EQU     0           REPORT FREE SPACE ON DISK
64 0001         OSO4W   EQU     1           OPEN FOR WRITE
65 0002         OSWRIT  EQU     2           WRITE
66 0003         OSWC    EQU     3           WRITE CLOSE
67 0004         OSO4R   EQU     4           OPEN FOR READ
68 0005         OSREAD  EQU     5           READ
69 0006         OSRC    EQU     6           READ CLOSE
70 0007         ODEL    EQU     7           DELETE
71 0008         OREN    EQU     8           RENAME
72 0009         OAPP    EQU     9           APPEND
73 000A         ODIRI   EQU     10          DIRECTORY INITIALIZATION
74 000B         ODIRT   EQU     11          DIRECTORY TRANSFER
75 0012         OSSR    EQU     18          SINGLE SECTOR READ
76 0013         OSSW    EQU     19          SINGLE SECTOR WRITE

78              * DFM ERROR CODES

80 0001         EIFC    EQU     1           ILL. FUNCTION CODE
81 0002         EFE     EQU     2           FILE EXISTS
82 0003         EFIO    EQU     3           MFD ERROR
83 0004         EFB     EQU     4           FILE BUSY
84 0005         ENSF    EQU     5           NO SUCH FILE
85 0006         EOF     EQU     6           END OF FILE
86 0007         EDF     EQU     7           DISK FULL
87 0008         EIF     EQU     8           ILLEGAL FCB
88 0009         EIFN    EQU     9           ILLEGAL FILE NAME
89 000A         EFS     EQU     10          FILE STATUS ERROR
90 000B         EITS    EQU     11          ILL. TRACK/SECTOR # (SSIO)
91 000C         EIUN    EQU     12          ILL. UNIT #
92 000E         EDR     EQU     14          DISK READ ERROR
93 000F         EDW     EQU     15          DISK WRITE ERROR

95 70A0                 ORG     TCA
96                      OPT     NOG,NOP

98 70A0 CE 0000 START   LDX     #0
99 70A3 FF 7149         STX     USRCNT      CLR SECTOR COUNTERS
100 70A6 FF 714B        STX     SYSCNT
101 70A9 FF 714D        STX     AVAIL
102 70AC CE 76D0        LDX     #FCB
103 70AF BD 7297        JSR     ZGNCHR      WHICH DRIVE?
104 70B2 81 30          CMP A   #'0
105 70B4 2F 0A          BLE     ZERO
106 70B6 81 33          CMP A   #'3
107 70B8 2C 06          BGE     ZERO
108 70BA A0 30          SUB A   #'0
109 70BC A7 02          STA A   XUN,X
110 70BE 20 02          BRA     CON
111 70C0 6F 02  ZERO    CLR     XUN,X
112 70C2 86 0A  CON     LDA A   #ODIRI
113 70C4 A7 00          STA A   XFC,X
114 70C6 BD 7786        JSR     DFM
115 70C9 26 34          BNE     ERR
116 70CB 86 0B  M0      LDA A   #ODIRT      READ AN FIB
117 70CD CE 76D0        LDX     #FCB
118 70D0 A7 00          STA A   XFC,X
119 70D2 BD 7786        JSR     DFM
120 70D5 26 28          BNE     ERR
121 70D7 A6 09          LDA A   XFE,X       CHECK EXTN
122 70D9 81 24          CMP A   #'$
123 70DB 27 09          BEQ     ADDSYS      ADD TO SYSTEM COUNT
124 70DD A6 12          LDA A   XSUC,X
125 70DF E6 13          LDA B   XSUC+1,X
126 70E1 CE 7149        LDX     #USRCNT
127 70E4 20 07          BRA     M1

129 70E6 A6 12  ADDSYS  LDA A   XSUC,X
130 70E8 E6 13          LDA B   XSUC+1,X
131 70EA CE 714B        LDX     #SYSCNT
132 70ED EB 01  M1      ADD B   1,X
133 70EF 25 08          BCS     UPCNT
134 70F1 E7 01          STA B   1,X
135 70F3 AB 00  M2      ADD A   0,X
136 70F5 A7 00          STA A   0,X
137 70F7 20 02          BRA     M0
138 70F9 E7 01  UPCNT   STA B   1,X
139 70FB 6C 00          INC     0,X
140 70FD 20 F4          BRA     M2

142 70FF A6 01  ERR     LDA A   XES,X       GET ERR CODE
143 7101 81 06          CMP A   #EOF
144 7103 27 06          BEQ     REPORT      GIVE DSK INFO
145 7105 BD 72A9        JSR     ZTYPDF
146 7108 7E 7283 EXIT   JMP     ZWARMS
```

Continued on pg. 29

SORTING SEQUENTIAL FILES IN PLACE

BY DAVID S. BURRIS
Computer Science Department
Sam Houston State University
Huntsville, Texas

WAYNE BYRD
United States Army
Woodbridge, Virginia

Mini and micro computers are performing many jobs today formerly relegated to larger computer systems with extensive auxiliary storage. Sorting of sequential files on compact systems is frequently a challenge when the number of records to be sorted exceeds main memory and auxiliary storage space is limited. Many efficient sorting techniques, such as polyphase and a balanced merge (2,3), are available when two or more temporary sequential tape/cassette or direct access files may be allocated to the sorting process. The problem is greatly aggravated, however, when there is insufficient direct access storage available to allocate one or more temporary files or if a single sequential auxiliary storage device (such as a cassette drive) must hold both the sorted and unsorted file. The following algorithm sorts a sequential file under the circumstances stated with no movement of records. The general strategy is to count the number of occurrences of each KEY in the first pass through the file and record this number in an array as COUNT(KEY). Hence, after the first pass, the number of occurrences of each key are known. In subsequent passes, all records with the lowest key value are processed before keys with higher values by utilizing the information contained in the array COUNT.

Algorithm Sort-In-Place

The N records to be sorted are on FILE 1 where N is larger than the amount of available main memory, M, and the number of unique keys K is less than M. An integer array, COUNT, of K counters is required and each of the K unique key values are associated by an index with an entry of COUNT.

```
STEP 1:  [initialize]
         For I:=1 step 1 to K set COUNT(I):=0.
STEP 2:  [count key occurrences]
         For I:=1 step 1 to N
              BEGIN Read KEY; COUNT(KEY):=COUNT(KEY)+1 END;

STEP 3:  [perform desired action]
         For I:=1 step 1 to K perform step 5.

STEP 4:  STOP, the sort is complete.

STEP 5:  Rewind FILE 1.  (For cassette and tape drives
         rewind means to rewind the tape, for direct
         access devices rewind means to reposition
         the file pointer to the first record.)
         While COUNT(I)>0
             BEGIN Read NEWKEY; if NEWKEY = KEY of COUNT(I) then set
             COUNT(I):=COUNT(I)-1
             and process the record in the desired
             manner END.
```

Figure 1 shows the contents of the array count after Step 2 for a set of alphabetic keys. Sequential files in which the number of unique keys K is small compared to the total number of records N to be sorted occur frequently. Consider a personnel file maintained in ascending order by social security number.

Keys and associated index			File-to-sort	Contents of COUNT(I)
Keys		Index		
Joe	—	1	Joe	COUNT(1)=3
Max	—	2	Max	COUNT(2)=2
Sam	—	3	Sam	COUNT(3)=1
(a)			Joe	(c)
			Max	
			Joe	
			(b)	

(a) The keys in the input file and their associated index in the array COUNT. (b) The file of records to be sorted. (c) The contents of the array COUNT after Step 2 of the algorithm.

Figure 1

For printing the payroll it may be desirable to partition the file based upon the record field that indicates the employee's department. Thus, all paychecks for a given department could be printed together to facilitate distribution. Hence, in Step 5 of the algorithm, the action taken would be to print the paycheck and the order of the original file is not destroyed. A copy of the sorted file may be made in Step 5 if desired and space for a second file in sequential or direct access storage is available.

Sort-In-Place can also be useful when a set of data is to be grouped into ranges and then all data within a particular range processed before proceeding to the next range. This occurs frequently in scientific data processing. For example, the output file may consist of real numbers. By breaking the file into ascending numeric ranges and then adding by ranges, numeric roundoff errors can be minimized or a file sorted by ranges may be generated in Step 5 as described above. If the ranges are judiciously chosen such that the number of records in each range as determined in Step 2 will fit in main memory, a completely sorted file can be generated. This may be accomplished by storing all records within a given range in main memory, sorting them, and then writing the sorted records to a second file. The number of sorts required would be equal to the number of original file partitions (ranges).

Efficiency

The reader may recognize that SORT-IN-PLACE is an adaptation of a well known main memory sorting technique known as sort by counting (1,2). The time required by the algorithm is dependent upon the number of unique keys in the file. In general K+1 passes must be made over the original file. Fewer passes may be required if processing of KEY I+1 is started in Step 5 when COUNT(I)=0 without rewinding the file. At most K passes are required if the smallest key is known and processed during the counting phase. SORT-IN-PLACE is efficient only if the number of unique keys or categories to be grouped is small. The algorithm as stated has the advantage of being "stable". A sorting algorithm is stable if records with the same key value are in the same relative order in both the sorted and unsorted files. When two or more temporary files can be allocated to the sorting process, other techniques (that are much faster) can and should be used (2,3). If the file exists on magnetic tape, the time required for the sort may be excessive as the tape must be rewound after every pass. The sort can be particularly efficient if the number of unique keys K is much less than the number of records to be sorted and the file exists in sequential direct access storage. No auxiliary storage is required.

References

1. Feurzeig, W. *Mathsort*, Comm. ACM 3 (1960), 601.
2. Knuth, Donald E. *Sorting and Searching*, Addison-Wesley, 1973.
3. Martin, William *Sorting*, Computing Surveys 3 (1971), 147.

VICE VERSA

Pencil to CPM and CPM to Pencil

BY JIM GAGNE, M.D.
515 Dalehurst Avenue
Los Angeles, CA 90024

Any CP/M user knows what a drag it is to write source code with the CP/M editor, and how nice it would be to have a good screen-oriented editor like Michael Shrayer's Electric Pencil. Unfortunately, Pencil files are not accepted by the CP/M assembler, BASIC, or other programs, primarily because of a problem with the line termination characters. I was outraged that Michael Shrayer would charge $35 for a simple little program to convert .PCL files from the Electric Pencil II to CP/M files and vice versa, since the only differences are line termination characters (Pencil files omit the line feed), tabs (there is no HTAB character in Pencil files, only the proper number of spaces), and end-of-file characters (1AH in CP/M, 00 in Pencil). So PENCPM is my answer to Shrayer, and I think it works well. Since I wrote my program before he released his, any similarity between our productions is obviously due to his intelligence. Mine features nice error messages, you have your choice of CP/M filetype if the file to be converted is a .PCL, and the thing has no known bugs (as of right now, anyway). When possible, blanks are compressed to horizontal tabs, saving some disk space. You will note that records are read and written one at a time, which slows the program some. I'm working on a version that reads and writes 512-byte blocks at one time, to speed it up, but so far the program crashes for reasons unknown. I'll send in a patch if I get it working.

About the only aspect of the program that might cause some trouble is how I handle the ends of lines. One of the nice things about the Pencil is that it automatically handles line feeds within a paragraph, and you don't have to hit a CR/LF unless you want to be certain a line ends at a certain point, e.g., an end of a paragraph. So the Pencil stores all text not separated by user-typed line feeds as one long line, which is then split up again at the time of being put on the screen or printed page. Obviously, this won't work for an unaltered CP/M text file, as many printers and programs will rebel at an unlimited line length. I have handled this situation very simply, just by chopping off any text that extends more than a fixed line length (currently 80 characters), forcing a carriage return/line feed sequence, and then resuming typing whatever text was interrupted. I chose the easy way out because I plan to convert only source code files, which need to be less than 80 characters per line anyway to fit on my printer. However, this way of handling ends of lines will frequently cause paragraph-oriented text to be chopped off in the middle of a word.

If you need to fix it, there are two ways of handling this situation. First, write a program that intercepts Pencil output to the printer (at the BIOS lister jump vector) and diverts it to a file with filetype ".TXT". This will allow you right justification if you wish it, plus any margins you want. Similar to the write portion of PENCPM, the diverting program would open a new file, write to it via a copy of the SEND routine, fill any extra space with end-of-file characters (1A hex) and close the file when done (perhaps signaled by a special keyboard character caught by your console input routine). This program would by itself translate text to CP/M compatible form, unless you have a problem with a reversed carriage return/line feed sequence (Pencil prints line feed/carriage return), which is very easy to fix if needed.

Alternatively, one could modify the present SEND routine to first write to a line buffer and then to the write buffer, forcing a CR/LF at the last blank character or hyphen before the end of the new line.

One note for writing successful source code with the Pencil: you must be *sure* to end each line with a line feed. If not, the text will look very similar on the screen, but the conversion routine will have no way of telling that there's a new line. Also, lines longer than LINLNGTH (currently 80 characters) will be split in the middle, producing an assembler error. To avoid a problem, the user may change LINLNGTH as desired.

I am donating PENCPM to the computing community, with the stipulation that no one ever charge for it more than the cost of the underlying media. And if you ask me, as overpriced as the Pencil is already, Shrayer should incorporate a patch into the body of the main program and then give it away free for nothing to bona fide users.

```
TYPE PENCPM.PRN

                ;FILE CONVERSION ROUTINE, PENCIL TO CP/M AND CP/M TO PENCIL
                ;
                ;       BY JIM GAGNE, M.D.
                ;       DATAMED RESEARCH
                ;       515 DALEHURST AVENUE
                ;       LOS ANGELES, CA 90024

                ;       WRITTEN SEPT 20, 1978
                ;       COPYRIGHT (C) 1978, JAMES GAGNE
                ;THIS PROGRAM MAY BE USED OR GIVEN AWAY BY ANYONE.  IT MAY NOT
                ;BE SOLD.  ANY SIMILARITY TO ANY OTHER PROGRAM IS PURELY
                ;COINCIDENTAL.
                ;
                ;OBJECT OF PROGRAM:
                ;
                ;1.     ACCEPTS ANY FILE (ANY FILETYPE BUT '.PCL' ASSUMED TO BE
                ;       CP/M) WHEN PROGRAM CALLED.  CREATES NEW FILE OF SAME FILE-
                ;       NAME BUT DIFFERENT FILETYPE ('.PCL' IF OLD FILETYPE NOT
                ;       '.PCL').  QUERIES USER FOR NEW FILETYPE IF IT'S TO BE CP/M.
                ;
                ;2.     ERASES ANY OLD FILES OF SAME <FILENAME>.<FILETYPE> AS
                ;       NEW FILE, THEN CREATES NEW FILE AND PERFORMS CONVERSION.
                ;
                ;3.     AT END OF FILE, FILLS REST OF FINAL RECORD WITH APPRO-
                ;       RIATE EOF CHARACTER AND CLOSES FILE, THEN BACK TO CCP.
                ;
                ;
0005 =          BDOS            EQU     05      ;JUMP ADDR FOR I/O ROUTINES.
                ;                               INPUT PARAM IN DE; PROCEDURE CODE
                ;                               IN C; RESULT IN A REG'S
                ;
                ;****** << BDOS PROCEDURE CODES >> ******
0009 =          PRBFCODE        EQU     9       ;PRINT BUFFER, STOPPING AT '$'
000A =          RDBFCODE        EQU     10      ;READ CONSOLE, PUT IN BUFFER (DE)
000B =          CSTSCODE        EQU     11      ;A=FF IF CNSL CHAR WAITING, ELSE 0
000F =          OPENCODE        EQU     15      ;OPEN EXISTING FILE
0010 =          CLOSECOD        EQU     16      ;CLOSE NEW FILE
0011 =          SRCHCODE        EQU     17      ;SEARCH FOR <FILENAME><TYPE>
0013 =          DELFCODE        EQU     19      ;DELETE FILE
0014 =          READCODE        EQU     20      ;READ NEXT RECORD
0015 =          WRITCODE        EQU     21      ;WRITE NEXT RECORD
0016 =          MAKECODE        EQU     22      ;CREATE NEW FILE
001A =          DMACODE         EQU     26      ;SET DMA
                ;
005C =          READFCB         EQU     5CH     ;STANDARD FILE CONTROL BLOCK
005D =          READNAME        EQU     READFCB+1       ;START OF FILENAME
0065 =          READTYPE        EQU     READFCB+9       ;FILETYPE
                ;
                ;WRITEFCB DEFINED AT END OF PROGRAM IN STORAGE AREA
                ;
                ;
0080 =          READBUFF                EQU     80H     ;DEFAULT BUFFER AREA
                ;WRITBUFF IS AT THE END.
                ;
0050 =          LINLNGTH        EQU     80      ;MAX LENGTH CP/M LINE
                ;<< WARNING >>  WHILE PENCIL PUTS ALL TEXT WITHIN A PARAGRAPH
                ;IN ONE LINE ON THE DISK (NO CR OR LF), WE CANNOT DO THAT WITH
                ;CP/M.  PROBLEM IS THAT WORDS MAY BE CHOPPED OFF IN THE MIDDLE
                ;OR SOURCE CODE LINES MAY HAVE EXTRA, INSERTED CR/LF IF TOO LONG.
                ;FEEL FREE TO CHANGE LINLNGTH IF NEEDED; HOPE IT FITS ONYOUR PRNTR
                ;
                ;CONTROL CHARACTERS
0000 =          NUL             EQU     00
0009 =          HTAB            EQU     09
000A =          LF              EQU     0AH
000D =          CR              EQU     0DH
0020 =          SPACE           EQU     20H
0020 =          BL              EQU     SPACE
                ;
0100                    ORG     100H
                ;
                ;
                ;INITIALIZE, STORING OLD STACK POINTER.
0100 3A5D00             LDA     READNAME        ;GET 1ST CHAR OF FILENAME
0103 FE20               CPI     SPACE   ;BLANK = NO FILENAME ENTERED
0105 CA4401             JZ      NONAME          ;ERROR MSG & RETURN
0108 115C00             LXI     D,READFCB
010B 0E0F               MVI     C,OPENCODE      ;OPEN SOURCE FILE
010D CD0500             CALL    BDOS
0110 3C                 INR     A       ;IF A = 0FFH, ERROR
0111 CA9A01             JZ      CANTFIND        ;ERROR MSG & RETURN
0114 210000             LXI     H,0
0117 39                 DAD     SP
0118 222005             SHLD    OLDSP
011B 316205             LXI     SP,NEWSP
                        NEXT, COPY OLD <FILENAME> INTO NEW FCB.
011E 215D00             LXI     H,READNAME
0121 117B04             LXI     D,WRITNAME
0124 010800             LXI     B,8
0127 CDEF02             CALL    MOVEHTOD
012A 216500             LXI     H,READTYPE      ;GET FILETYPE
012D 7E                 MOV     A,M
012E FE50               CPI     'P'     ;IF <> '.PCL', THEN MUST BE CP/M FILE
0130 C28C02             JNZ     CONVTPCL        ;AND MUST BE CONVERTED TO .PCL
0133 23                 INX     H
0134 7E                 MOV     A,M
0135 FE43               CPI     'C'
0137 C28C02             JNZ     CONVTPCL
013A 23                 INX     H
013B 7E                 MOV     A,M
013C FE4C               CPI     'L'
013E C28C02             JNZ     CONVTPCL
0141 C3C701             JMP     ASKTYPE         ;ELSE CONVERT TO CP/M
0144 114D01     NONAME: LXI     D,NONMMSG
0147 0E09       NONM1:  MVI     C,PRBFCODE
0149 CD0500             CALL    BDOS
014C C9                 RET     ;BACK TO CCP W/O REBOOT
                NONMMSG.
014D 0D0A594F55         DB      CR,LF,'YOU MUST TYPE NAME OF FILE TO BE CONVERTED'
0179 2C20494E43         DB      ', INCLUDING FILETYPE.  TRY AGAIN.$'
                CANTFIND:
019A 11A001             LXI     D,CFMSG
019D C34701             JMP     NONM1
                CFMSG:
01A0 0D0A534F55         DB      CR,LF,'SOURCE FILE NOT PRESENT.  TRY AGAIN.$' ;
                ;THIS IS THE START OF THE CONVERSION TO CP/M FILE.
01C7 11D201     ASKTYPE: LXI    D,FILEMSG       ;ASK USER FOR CP/M FILETYPE
01CA 0E09               MVI     C,PRBFCODE      ;'PRINT BUFFER' CODE
01CC CD0500             CALL    BDOS    ;SEND THE MESSAGE
01CF C3F601             JMP     INTYPE
01D2 5748415420FILEMSG: DB      'WHAT IS THE CP/M FILETYPE?'
01EC 092E24             DB      HTAB,'.$'       ;HTAB THEN PRINT PERIOD,
                        SO ONLY ALPHANUMERIC CHAR ENTERED.
01EF 0600000000CIBUFF:  DB      6,0,0,0,0,0,0,0 ;CONSOLE INPUT BUFFER
                ;
                INTYPE:
01F6 11EF01             LXI     D,CIBUFF
01F9 0E0A               MVI     C,RDBFCODE      ;'READ BUFFER' CODE
01FB CD0500             CALL    BDOS    ;GET SOME CHARACTERS
01FE 0603               MVI     B,3
0200 21F101             LXI     H,CIBUFF+2      ;1ST BYTE =LNGTH BUFF; 2ND =# USED
0203 118304             LXI     D,WRITTYPE      ;WHERE NEW FILETYPE GOES
0206 7E         TYP0:   MOV     A,M
0207 FE30               CPI     30H     ;MAKE SURE ALPHANUMERIC
0209 DAC701             JC      ASKTYPE
020C FE60               CPI     60H     ;UPPER CASE?
020E D2C701             JNC     ASKTYPE ;IF NOT, TRY AGAIN.
0211 12                 STAX    D
0212 23                 INX     H
0213 13                 INX     D
0214 05                 DCR     B
0215 C20602             JNZ     TYP0
                ;
                ;
                ;NOW MAKE SURE NO PREVIOUS ENTRIES IN DIR LIKE NEW <FILENAME.TYP>
                ;AND ERASE ANY THAT ARE THERE.  THEN CREATE NEW FILE AND SEND
                ;MESSAGE.
0218 CDFA02             CALL    CHECKDIR        ;MESSAGE CONFIRMING NAME.
                ;
                ;ALL SET; BEGIN MAIN WORK LOOP, ONE CHAR AT A TIME
                NXTCPMCH:
021B CD5703             CALL    GETCHAR ;GET NEXT CHAR, READING IN SOURCE FILE
                HKCPMCH:                ;AS NEEDED.
021E FE0D               CPI     CR      ;PENCIL FILES OMIT THE LINE FEED.
0220 CA3D02             JZ      OUTCRLF
0223 FE20               CPI     SPACE   ;REPLACE BLANKS WITH HTAB IF APPROPRIATE
0225 CA4E02             JZ      SPACERTN
0228 DA1B02             JC      NXTCPMCH        ;OMIT OTHER CONTROL CHAR
022B CDA203             CALL    SEND    ;MUST BE PRINTING CHAR - SEND AS IS.
022E 3A1E05             LDA     COLUMNNO ;MAKE SURE NOT OFF PAPER
0231 3C                 INR     A       ;AND INC TO UPDATE
0232 FE51               CPI     LINLNGTH+1
0234 CA3D02             JZ      OUTCRLF ;IF SO, FORCE CRLF
0237 321E05             STA     COLUMNNO
023A C31B02             JMP     NXTCPMCH ;GET MORE CHARACTERS
                ;
                ;       SENDS CRLF
                OUTCRLF:
023D 3E0D               MVI     A,CR
023F CDA203             CALL    SEND
0242 3E0A               MVI     A,LF
0244 CDA203             CALL    SEND
```

```
0247 97             SUB     A
0248 321E05         STA     COLUMNNO
0248 C31B02         JMP     NXTCPMCH
                    ;       SEND HTAB IF POSSIBLE, &/OR SPACES.
                    SPACERTN:
024E CDD703         CALL    CHECKTAB
0251 C31B02         JMP     NXTCPMCH
                    ;
                    ;       JOB COMPLETE, FILL REST OF WRITE FILE W/ <EOF> & QUIT.
0254 3E1A           OUTEOF: MVI     A,1AH   ;CTL Z = CP/M EOF
0256 CDA203         CALL    SEND
0259 05             DCR     B       ;B HAS 1 IF LAST SEND PRODUCED RECORD
                            ;WRITE, ELSE = 0 IF NO WRITE.
025A C25402         JNZ     OUTEOF ;IF 0, MORE EOF'S NEEDED UNTIL REC FILLED
                    CLOSEFIL:
025D 117A04         LXI     D,WRITEFCB
0260 0E10           MVI     C,CLOSECOD      ;CLOSE NEW FILE
0262 CD0500         CALL    BDOS
0265 3C             INR     A
0266 CA5904         JZ      WRITEERR        ;IF A = 0FFH, ERROR
0269 117802         LXI     D,DONEMSSG
026C 0E09           MVI     C,PRBFCODE
026E CD0500         CALL    BDOS
                    ;
                    ALLDONE:
0271 2A2005         LHLD    OLDSP
0274 F9             SPHL
0275 C30000         JMP     0       ;REBOOT.
                    ;
                    DONEMSSG:
0278 0D0A46554E     DB      CR,LF,'FUNCTION COMPLETE$'
                    ;
                    ;HERE IS THE CP/M TO PENCIL CONVERSION PART.  MUCH SIMPLER!
                    ;
                    CONVTPCL:
028C 218304         LXI     H,WRITTYPE
028F 3650           MVI     M,'P'   ;FILL <FILETYPE> W/ .PCL
0291 23             INX     H
0292 3643           MVI     M,'C'
0294 23             INX     H
0295 364C           MVI     M,'L'
0297 CDFA02         CALL    CHECKDIR        ;ESTABLISH NEW FILE, ETC.
                    NXTPCLCH:
029A CD5703         CALL    GETCHAR
029D FE0D           CPI     CR      ;SINCE PENCIL WRITES CR WHEN IT MEANS
029F CA9A02         JZ      NXTPCLCH ;CRLF, DUMP ALL CR AND CONVERT LF TO CR
02A2 FE0A           CPI     LF
02A4 CAC302         JZ      OUTCRPCL ;AND ZERO COLUMNNO
02A7 FE1A           CPI     1AH     ;=CP/M EOF, MEANING ALL DONE.
02A9 CAE302         JZ      PCLEOF
02AC FE09           CPI     HTAB    ;PENCIL EXPANDS ALL TABS
02AE CACF02         JZ      EXPNDTAB
02B1 FE20           CPI     20H     ;DUMP OTHER CTL CHARACTERS
02B3 DA9A02         JC      NXTPCLCH
02B6 CDA203         CALL    SEND    ;MUST BE PRINTING ASCII
02B9 3A1E05         LDA     COLUMNNO        ;ONE COLUMN OVER
02BC 3C             INR     A
02BD 321E05         STA     COLUMNNO
02C0 C39A02         JMP     NXTPCLCH
                    OUTCRPCL:
02C3 3E0D           MVI     A,CR
02C5 CDA203         CALL    SEND
02C8 97             SUB     A
02C9 321E05         STA     COLUMNNO
02CC C39A02         JMP     NXTPCLCH
                    EXPNDTAB:       ;REPLACE HTABS WITH SPACES TO NXT 8TH COLUMN.
02CF 3E20           MVI     A,SPACE
02D1 CDA203         CALL    SEND
02D4 3A1E05         LDA     COLUMNNO
02D7 3C             INR     A
02D8 321E05         STA     COLUMNNO
02DB E607           ANI     07      ;AT NEXT 8TH COLUMN?
02DD CA9A02         JZ      NXTPCLCH
02E0 C3CF02         JMP     EXPNDTAB
                    PCLEOF: ;FILL PCL FILE W/ NULS (EOF CHAR IN PCL) AND EXIT.
02E3 3E00           MVI     A,NUL
02E5 CDA203         CALL    SEND
02E8 05             DCR     B       ;B = 1 IF FILE JUST WRITTEN
02E9 C2E302         JNZ     PCLEOF
02EC C35D02         JMP     CLOSEFIL
                    ;
                    ;
                    ;
                    ;***** << SUBROUTINES >> *****
                    ;
                    ;**     MOVE CONTENTS OF HL TO AREA POINTED BY DE, BC TIMES.
                    ;       SAME AS LDIR COMMAND, Z-80
                    MOVEHTOD:
02EF 7E             MOV     A,M
02F0 12             STAX    D
02F1 23             INX     H
02F2 13             INX     D
02F3 0B             DCX     B
02F4 78             MOV     A,B
02F5 B1             ORA     C
02F6 C2EF02         JNZ     MOVEHTOD
02F9 C9             RET
                    ;
                    ;**     CHECKDIR DELETES ANY EXISTING FILE(S) OF SAME <FILENAME>
                    ;       & <TYPE> AS NEW FILE TO BE WRITTEN, CREATES NEW FILE,
                    CHECKDIR:       ;AND THEN TYPES MESSAGE.
02FA 117A04         LXI     D,WRITEFCB
02FD 0E11           MVI     C,SRCHCODE      ;SEARCH FOR FILE CODE
02FF CD0500         CALL BDOS
0302 3C             INR     A       ;0FFH = NO FILE FOUND
0303 CA1103         JZ      CREATFIL        ;GREAT, NOW MAKE NEW FILE.
0306 117A04         LXI     D,WRITEFCB
0309 0E13           MVI     C,DELFCODE      ;ELSE DELETE FILE
030B CD0500         CALL    BDOS
030E C3FA02         JMP     CHECKDIR        ;DO IT UNTIL NO MORE DUPL FILES.
                    CREATFIL:
0311 117A04         LXI     D,WRITEFCB
0314 0E16           MVI     C,MAKECODE
0316 CD0500         CALL    BDOS
0319 3C             INR     A       ;IF A = FF, NO ROOM IN DIR
031A CA5904         JZ      WRITEERR
                    ;
                    ;       NOW TELL THEM WHAT YOU'RE DOING.
031D 217B04         LXI     H,WRITNAME ;FIRST, MOVE FILENAME TO MESSAGE
0320 114903         LXI     D,NEWNAME  ;AREA BELOW.
0323 010800         LXI     B,8
0326 CDEF02         CALL    MOVEHTOD
0329 13             INX     D       ;SKIP A SPACE
032A 010300         LXI     B,3
032D CDEF02         CALL    MOVEHTOD
0330 113803         LXI     D,TELLMSG       ;ALL SET.  SEND THE MESSAGE.
0333 0E09           MVI     C,PRBFCODE
0335 C30500         JMP     BDOS
                    TELLMSG:
0338 0D0A434F4E     DB      CR,LF,'CONVERTING TO:
                    NEWNAME:
0349 46494C454E     DB      'FILENAME TYP   $'  ;DUMMY FILENAME
                    ;
                    ;**     GET NEXT SOURCE CHAR ROUTINE.  READS NEW RECORD IF NEEDED
                    GETCHAR:
0357 3A1B05         LDA     READPTR ;THIS = DISPLACEMENT FROM READBUFF
035A 3C             INR     A
035B FE80           CPI     80H
035D CA6C03         JZ      READFILE
0360 4F             GC0:    MOV     C,A
0361 321B05         STA     READPTR
0364 0600           MVI     B,0
0366 218000         LXI     H,READBUFF
0369 09             DAD     B
036A 7E             MOV     A,M
036B C9             RET
                    READFILE:
036C 0E0B           MVI     C,CSTSCODE
036E CD0500         CALL    BDOS    ;CHECK IF CHAR WAITING
0371 B7             ORA     A
0372 C27102         JNZ     ALLDONE ;IF SO, ABORT.
0375 3A1D05         LDA     EOF     ;IF END OF FILE, ALL DONE!
0378 B7             ORA     A
0379 C25402         JNZ     OUTEOF  ;JUST FILL REST OF WRITE FILE WITH <EOF>
037C 118000         LXI     D,READBUFF      ;SET CORRECT DMA
037F 0E1A           MVI     C,DMACODE       ;SETDMA CODE
0381 CD0500         CALL    BDOS
0384 3EFF           MVI     A,0FFH
0386 321B05         STA     READPTR ;RESET POINTER
0389 115C00         LXI     D,READFCB
038C 0E14           MVI     C,READCODE      ;READ RECORD CODE
038E CD0500         CALL    BDOS
0391 B7             ORA     A
0392 CA6003         JZ      GC0     ;A = 0 IF GOOD READ
0395 3D             DCR     A       ;A = 1 IF END OF FILE
0396 C23404         JNZ     READERR ;ELSE HARD READ ERROR, TYPE MESSAGE & QUIT
0399 3EFF           MVI     A,0FFH
039B 321D05         STA     EOF     ;FF = END OF FILE TRUE, SO NXT READ EXITS.
039E 97             SUB     A       ;READ 1ST BYTE
039F C36003         JMP     GC0
                    ;
                    ;**     THIS SUBROUTINE WRITES A CHARACTER IN REG A
                    ;       INTO A WRITE BUFFER, AND
                    ;       WRITES THE RECORD WHEN FULL.  OUTPUT: B = 1 IF FILE
                    ;       WRITE PERFORMED, ELSE 0.
```

```
                SEND:
03A2 F5         PUSH    PSW     ;SAVE BYTE
03A3 3A1C05     LDA     WRITEPTR
03A6 4F         MOV     C,A     ;CALCULATE BUFFER ADDR
03A7 0600       MVI     B,0
03A9 219B04     LXI     H,WRITBUFF
03AC 09         DAD     B
03AD F1         POP     PSW     ;RESTORE CHAR
03AE 77         MOV     M,A
03AF 79         MOV     A,C
03B0 3C         INR     A       ;UPDATE POINTER
03B1 FE80       CPI     80H     ;END OF BUFFER?
03B3 D2BC03     JNC     WRITFILE ;THEN PUT IT ON DISK.
03B6 321C05     STA     WRITEPTR ;ELSE STORE UPDATED POINTER
03B9 0600       MVI     B,0     ;RESET B
03BB C9         RET
                WRITFILE:
03BC 97         SUB     A
03BD 321C05     STA     WRITEPTR
03C0 119B04     WRFL1:  LXI     D,WRITBUFF
03C3 0E1A       MVI     C,DMACODE       ;SET NEW DMA
03C5 CD0500     CALL    BDOS
03C8 117A04     LXI     D,WRITEFCB
03CB 0E15       MVI     C,WRITCODE      ;WRITE NEXT FILE CODE
03CD CD0500     CALL    BDOS
03D0 B7         ORA     A       ;A = 0 IF GOOD WRITE
03D1 C25904     JNZ     WRITEERR ;ERROR MESSAGE
03D4 0601       MVI     B,1     ;SIGNAL THAT WRITE DONE.
03D6 C9         RET
                ;
                ;
                ;**     SPACE TO HTAB ROUTINES.
                CHECKTAB:
03D7 0601       MVI     B,1
03D9 C5         CKSPNO: PUSH    B
03DA CD5703             CALL    GETCHAR
03DD C1                 POP     B
03DE FE20               CPI     SPACE
03E0 C2E703             JNZ     NOTSP
03E3 04                 INR     B
03E4 C3D903             JMP     CKSPNO
03E7 3A1B05     NOTSP:  LDA     READPTR
03EA 3D                 DCR     A       ;RESTORE POINTR TO NON-SP CHAR
03EB 321B05             STA     READPTR
03EE 05                 DCR     B
03EF CA2004             JZ      OUTSP
03F2 04                 INR     B
                MORETH1:
03F3 48                 MOV     C,B     ;C = CORRECT # SPACES
03F4 3A1E05             LDA     COLUMNNO
03F7 3C         MT0:    INR     A
03F8 F5                 PUSH    PSW
03F9 E607               ANI     7       ;ARE WE AT NEXT 8TH COLUMN?
03FB CA0804             JZ      GOTTAB  ;THEN OUTPUT TAB
03FE F1                 POP     PSW     ;RESTORE COLUMNNO
03FF 05                 DCR     B
0400 C2F703             JNZ     MT0     ;JUMP IF MORE SPACES TO COUNT
0403 CD2004     OUTALLSP: CALL  OUTSP   ;SEND SPACES UNTIL ALL GONE
0406 0D                 DCR     C       ;C = SPACE COUNT
0407 C8                 RZ      ;WHEN 0, ALL DONE
0408 C30304             JMP     OUTALLSP
040B F1         GOTTAB: POP     PSW     ;RESTORE COLUMNNO (UPDATED)
040C FE51               CPI     LINLNGTH+1      ;OFF PAGE?
040E D23D02             JNC     OUTCRLF ;THEN NEXT LINE
0411 321E05             STA     COLUMNNO        ;ELSE STORE COLUMNNO
0414 3E09               MVI     A,HTAB
0416 C5                 PUSH    B
0417 CDA203             CALL    SEND
041A C1                 POP     B
041B 05                 DCR     B       ;B NOW HAS CORRECT # SPACES
041C C8                 RZ      ;ALL DONE IF NONE
041D C3F303             JMP     MORETH1
                ;
0420 3E20       OUTSP:  MVI     A,SPACE
0422 C5                 PUSH    B
0423 CDA203             CALL    SEND
0426 C1                 POP     B
0427 3A1E05             LDA     COLUMNNO
042A 3C                 INR     A
042B 321E05             STA     COLUMNNO
042E FE51               CPI     LINLNGTH+1
0430 D23D02             JNC     OUTCRLF
0433 C9                 RET
                ;
                ;**     ERROR ROUTINES
                READERR:
0434 113F04             LXI     D,RDERRMSG
0437 0E09       RE0:    MVI     C,9
0439 CD0500             CALL    BDOS
043C C35D02             JMP     CLOSEFIL
                ;
                RDERRMSG:
043F 0D0A524541         DB      CR,LF,'READ ERROR - TRY AGAIN. $'
                ;
                WRITEERR:
0459 115F04             LXI     D,WRERRMSG
045C C33704             JMP     RE0
                ;
                WRERRMSG:
045F 0D0A575249         DB      CR,LF,'WRITE ERROR - TRY AGAIN. $'
                ;
                ;
                ;***** << BUFFER AREA >> *****
                ;
                ;       WRITE FILE CONTROL BLOCK.  INITIALIZED WITH 0'S & SPACES.

047A 0020202020WRITEFCB         DB      0,BL,BL,BL,BL,BL,BL,BL,BL,BL,BL,BL,0,0
0488 0000000000                 DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0483 =          WRITTYPE        EQU     WRITEFCB+9      ;FILETYPE
049A =          WRITNREC        EQU     WRITEFCB+32     ;NEXT RECORD TO READ/WRITE
                ;
049B            WRITBUFF:       DS      128     ;OUTPUT BUFFER
                ;
                ;***** << VARIABLES >> *****
                ;
051B 7F         READPTR         DB      7FH     ;INITIALIZED TO FORCE 1ST READ
051C 00         WRITEPTR        DB      00
051D 00         EOF:            DB      0       ;INITIALIZED TO 0; 1 = END OF FILE
051E 00         COLUMNNO:       DB      0
051F 00         VFYCOUNT:       DB      00      ;# OF TIMES WRITE VERIFIED.
0520            OLDSP:          DS      2
0522            STACK:          DS      40H     ;ROOM FOR STACK
0562 =          NEWSP:          EQU     $
0562                    END     100H    ;START AT 100
```

Continued from pg. 24

```
148 70FB A6 00  REPORT LDA A  #0FREE
149 70FD A7 00         STA A  XFC+X
150 70FF BD 7786       JSR    DFM
151 7102 26 ER         BNE    ERR
152 7104 A7 714D       STA A  AVAIL
153 7107 F7 714E       STA B  AVAIL+1
154 710A CE 711A       LDX    #HEADER
155 710D BD 72A6       JSR    ZOUTST
156 7110 CE 7148       LDX    #USRCNT-1
157 7113 C6 03         LDA B  #3
158 7115 37       M3   PSH B
159 7116 08            INX
160 7117 BD 72AF       JSR    ZOUTHA
161 711A C6 04         LDA B  #4
162 711C A6 20    M4   LDA A  #$20
163 711E BD 7286       JSR    ZOUTEE
164 7111 5A            DEC B
165 7112 26 FR         BNE    M4
166 7114 33            PUL B
167 7115 5A            DEC B
168 7116 26 ED         BNE    M3
169 7118 20 CE         BRA    EXIT

171 711A 00       HEADER FCB   $D,$A
172 711C 53            FCC    'SECTOR UTILIZATION-'
173 712F 0D            FCB    $D,$A
174 7131 55            FCC    'USER    SYSTEM   AVAIL'
175 7146 0D            FCB    $D,$A,0

177 7149          USRCNT RMB   2
178 714B          SYSCNT RMB   2
179 714D          AVAIL  RMB   2

1A1                      END

                SYMBOL TABLE:
ADDSYS  70C6  AVAIL  7140  CDFM   7783  DFM    7786
DFMORG  7780  EOF    0007  CON    70A2  EFB    0004
EFF     0002  EFIB   0003  EFS    000A  EIF    0001
EFN     0009  EITS   000B  EIUN   000C  ENSF   0005
EPR     000F  EXIT   70EA  FCB    76D0  HEADER 711A
M1      70CD  M2     70D3  M3     7105  M4     710C
OAPP    0009  ODEL   0007  ODIRI  0004  ODIRT  0008
OREN    0009  OSDAP  0004  OSDAW  0001  OSEC   0006
OSSR    0012  OSSW   0013  OSWC   0003  OSWRIT 0002
START   7080  SYSCNT 714A  TCA    70A0  UPCNT  7009
XCS     001F  XCT    001E  XDR    0026  XES    0001
XFF     0009  XFN    0003  XFS    000D  XFT    000C
XNS     0027  XNT    0026  XPS    0029  XPT    0028
XSOD    002A  XSUC   0012  XUN    0002  ZEBO   70A0
ZOUTFE  72A6  ZOUTHA 72AF  ZOUTST 72A6  ZTYPDE 72A9
                                        ZWARMS 7283
```

Build a Self-learning No-programming Computer with your Microprocessor

BY KLAUS HOLTZ
455 Eddy Apt. 1204
San Francisco, Ca. 94109

Author's Note: The algorithms described in this paper have been simulated in a computer and are sure to work as described. The implementation depends on your microprocessor hardware.

How would you like to build a futuristic self-learning computer which is educated almost like a human child and lets you get rid of all that messy programming? A computer with unlimited, reliable memory and total recall which, once set up, never needs programming or supervision of its internal functioning.

Science fiction? Not at all. These computers are already on the drawing board. Moreover, you can build such a computer in just a few days. All you need is a microprocessor, a cheap mass memory such as a tape cassette and some kind of keyboard/display. The resulting system will be truly astonishing, functioning in a way totally different from conventional computers. It is not just another clever architecture or programming trick, but a completely different way of building computers.

Here is what you can do with this new computer:

Educate it to give information similar to a dictionary, almanac or encyclopedia. Get information by simply asking for it through the keyboard. There is no messy format or computer language. Educate it in any subject without worry about programming or its internal functioning. Feed it any kind of knowledge such as sports records, inventories, stock prices or whatever needs to be remembered.

Use it as a diary for personal records, or as a reminder of future appointments. A calendar clock will retrieve the information and remind you of your appointments at the proper time.

Build a typewriter which refuses to type incorrectly spelled words.

Use it as language translator to translate one language into another.

Send a confidential message to your friends in a highly compressed form and unbreakable code.

Invent new games. Anticipate questions asked of the computer at an exhibit.

Exchange and merge the accumulated knowledge from your computer with other computers.

All these things could be accomplished in the same computer without re-programming it. Once you have set up your system you can forget all about programming.

The theories behind this new computer knowledge are Knowledge Processing and Infinite Dimensional Networks. The theory is rather scary: "Every concept may be specified like a point in multidimensional space." The point is you do not need to understand the whole theory to build a practical computer. Just follow the instructions in this paper. Once you have built your first primitive computer you'll have an idea how it works; no doubt you will later build a better one. This new computer works with "knowledge" instead of mere data, and the understanding of the nature of knowledge and how it may be stored in digital memories is the real breakthrough. The mathematical structure which ends up in your mass memory is called an infinite dimensional network. Its main feature is that it assembles itself from the input data without programming or human supervision. There are many different networks doing strange things such as electronic eyes which can truly see. You only need the most simple serial and associative networks. It isn't really so difficult once you get the hang of it.

System Hardware

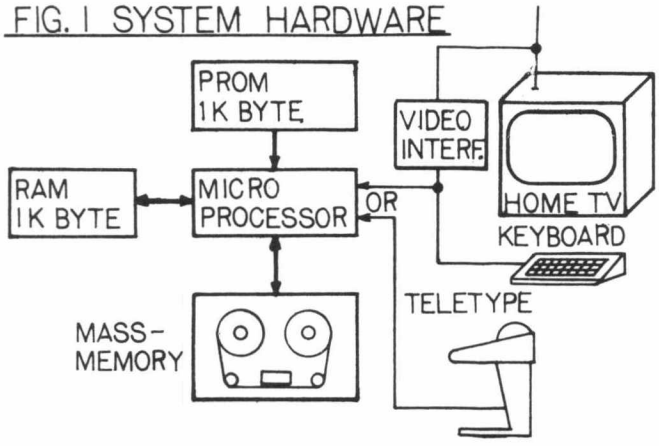
To build this new computer you need:

A Microprocessor. This is not really the most efficient way, but simulating the new computer in a microprocessor is quick and easy. Any commercial microprocessor will do. It should have about 1K byte of PROM and few K byte of RAM. The RAM can be eliminated later, depending on the mass memory (or if you have a buffered display).

A Mass Memory Device. This device should provide for lots of cheap and non-volatile memory space. Any kind of random or serially accessible memory will do. The memory field may be divided into blocks where each block is first read into the buffer RAM before it is treated by the microprocessor. A special feature of this memory is that once a word of information is stored it will never be changed. An EPROM will work but is very expensive. Another strange feature of this new computer is that the more information stored in the memory, the less additional storage required to store additional information. This will almost saturate memory requirement after some time and lead to large data compression. The best memory devices for your system would be an audio cassette, a floppy disk or a tape cassette. It does help if these devices could be read backwards. Later computer will use bubble memories when they become less expensive.

A Keyboard Display. Some way of entering and retrieving alphanumeric information must be provided. The easiest way would be through a teletype or a packaged display unit but this is expensive. A keyboard connected via a TV video interface to your home TV set may be more convenient and less expensive. You may also build your own keyboard and alphanumeric display. Very soon these computers will have voice input and output at which time you can simply talk to them.

FIG. 1 SYSTEM HARDWARE



Self Assembling Infinite Dimensional Networks

These networks are really the very heart of this new computer. Only the most simple serial and associative networks are required for your system. If you follow the flow charts and compare the results with the sample memory contents, you should get the idea of how these networks assemble themselves from the input data. This is not just a funny way to store data, but is genuine learning, analogous in principle to our own brain. Each of the three basic routines are here explained in detail. They will be called out later in the system flow chart. A few basic terms must be introduced here without resorting to complex theory.

GATE: This in the input/output data from your keyboard. It will most likely be ASCII character but any other code may by used. The GATE can also be ADDRESS numbers. It should be 16 bits wide.

ADDRESS: This is the ADDRESS of the memory locations in your mass memory device. In serial accessible memory deivces, this ADDRESS must be generated externally. The term ADDRESS is used in its conventional way in which each data word in the memory device is stored in a unique address location.

POINTER: This is the ADDRESS of a previous data word which is stored together with the GATE in each memory word. It points at a previous ADDRESS in the operation sequence. The pointer should be 16 bits wide.

MATRIX: This is the combination of a GATE and a POINTER or 2 POINTERS which are stored as data word in the mass memory device. It is equivalent to the data word or memory content. The memory word should be at least 32 bits wide.

MATRIX REGISTER: This is a special register inside your microprocessor. Any internal or external register may be used. This register will be very active and important and should be atlleast 4 bytes (32 bits) wide.

Serial Network Routine

The purpose of this network is to convert any input word or sentence into a single address number. The network will assemble itself from the input data and will expand automatically to absorb (learn) new information. The resulting memory content in Fig. 5 is called a stored infinite dimensional network even though its network character is hard to visualize.

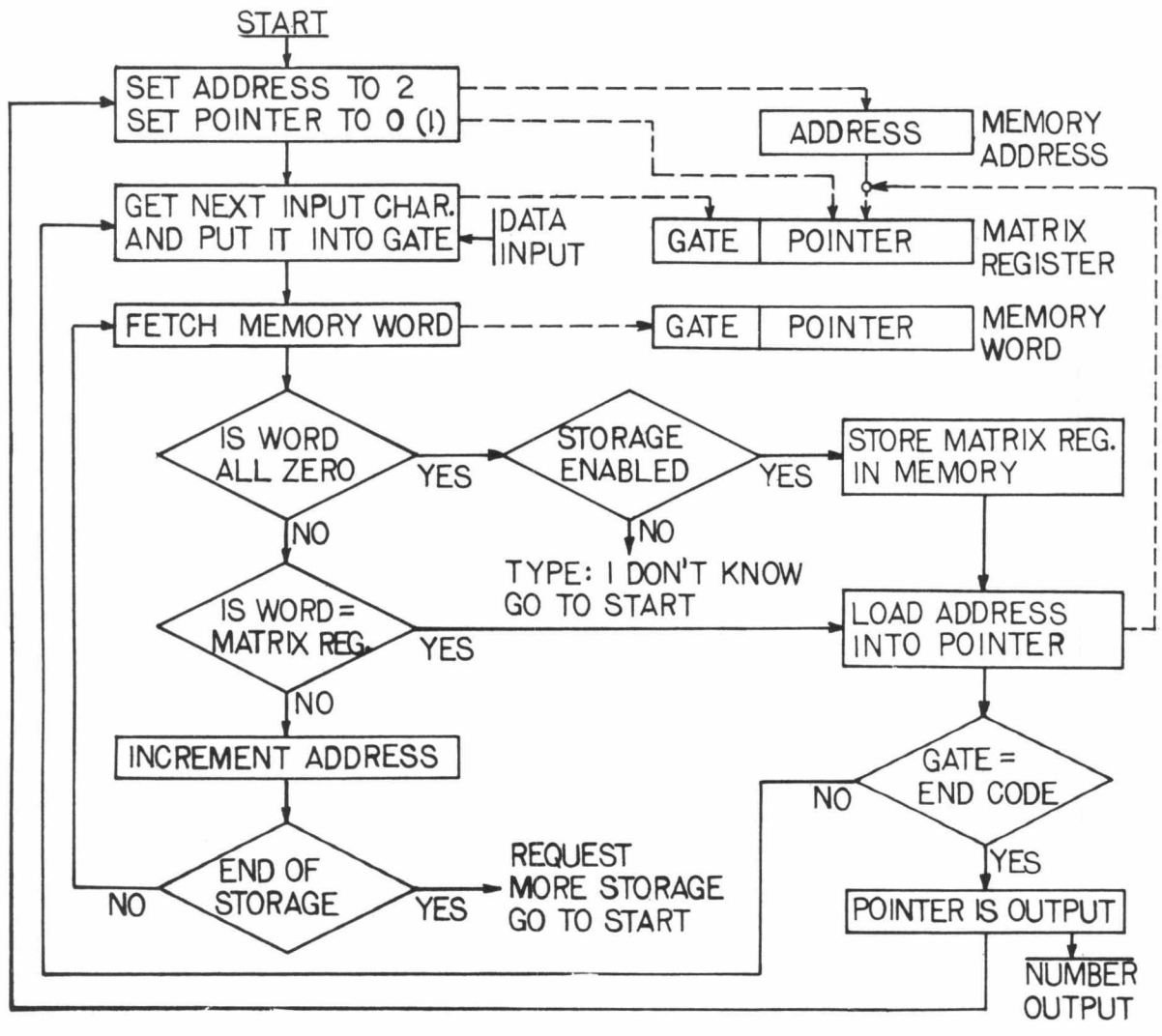
Fig. 2 shows the simple routine which must be converted into micro-code and stored in the program PROM. The particular code will depend on your microprocessor.

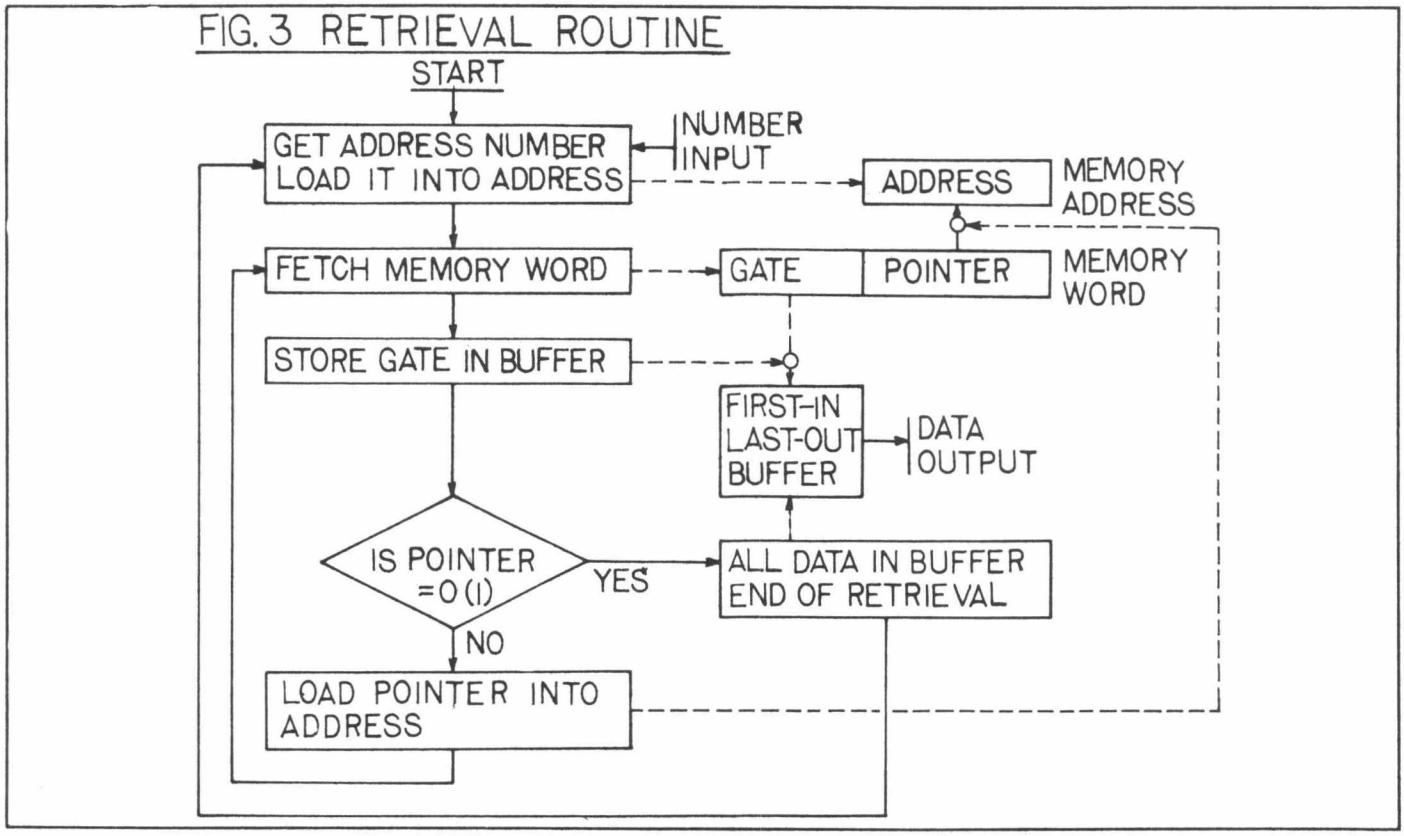
The first two address locations in the mass memory are reserved as starting address for each of two networks. These networks are interleaved automatically by choosing separate starting addresses. We start the memory scan at address 2. The data character from the buffer RAM or keyboard is loaded into the gate portion of the matrix register inside our microprocessor. The pointer portion is loaded with the start address 0 (or 1). The microprocessor will scan the mass memory to find a memory word which is the same as the present content of the matrix register. Every memory word is fetched in sequence and compared with the matrix register. If we reach the end of the presently used memory field we find that the memory word is all zero. If storage is enabled we store the matrix register in the memory, thereby adding a new word at the end of memory field. If storage is not enabled we type out a message such as I DON'T KNOW. After storing the matrix register we load the present memory address (the address where the new word was stored) into the pointer portion of the matrix register. We now examine the data character (Gate) in the matrix register. If it is a code indicating the end of a written word such as space, comma, full stop, question mark etc., we use the present pointer number in the matrix register as output from this routine and go back to start.

If during the memory scan we encounter a word in the memory which is identical to the content of the matrix register, we load the address at which this word was found into the pointer portion of the matrix register and continue the scan with the next input character. If we reach the end of the storage, we should add more memory space since all the internal memory is used up. The network will expand and fill the memory until it runs out of storage space.

If all this sounds like black magic, compare how the word IDEA would be stored in the sample memory. It is assumed that the words ROSE, ROBOT and IDN were previously stored and the word IDEA entered for the first time. The starting address is 0, which is used as the first pointer. The first input character I will lead to the first matrix Gate I and Pointer 0. We scan the memory in forward direction to detect a matrix I-0. We find such a word in address 11. Address 11 becomes the next pointer which is combined with the next

FIG.2 SERIAL NETWORK ROUTINE





input charcter D to yield the next matrix D-11. Such a matrix is already stored in address 12. Address 12 becomes pointer 12 which is combined with the next input character E to yield the next matrix E-12. We reach the end of the used memory field without finding such a matrix. We therefore store matrix E-12 in the next free memory location, which is address 15. Address 15 becomes pointer 15 and together with the last character A, forms the last matrix A-15, which is stored in the next free memory location adddress 16. Once storage has been initiated there is no need to search the memory for the following matrixes since none will be found. The next input character is an end of word code (space) which is stored in address 17 and terminates the routine. The current pointer 17 is the output for the word IDEA.

This output address 17 totally identifies the word IDEA. If we follow the same input sequence with the word IDEA again we would again obtain address 17 but no new storage would be required.

As said earlier, the purpose of this routine is to convert any input word into a unique address number which represents that word. If a word was not previously stored (learned) it will be added automatically. Once it is stored it will be recognized but never stored again. Also note that memory locations are shared between similar words so that memory requirement will saturate after a while. The whole English vocabulary could be stored this way by simply typing the words into the network. Every word will be stored (learned) only once and the words may be entered in any random repetitive way. All you have to do is type the input text. You do not have to worry about pointers and address and whether or where data is stored. All this is taken care of by the micro routine once it is set up. If you are too lazy to type the text you may connect it to some telegraph line for it to automatically learn the vocabulary.

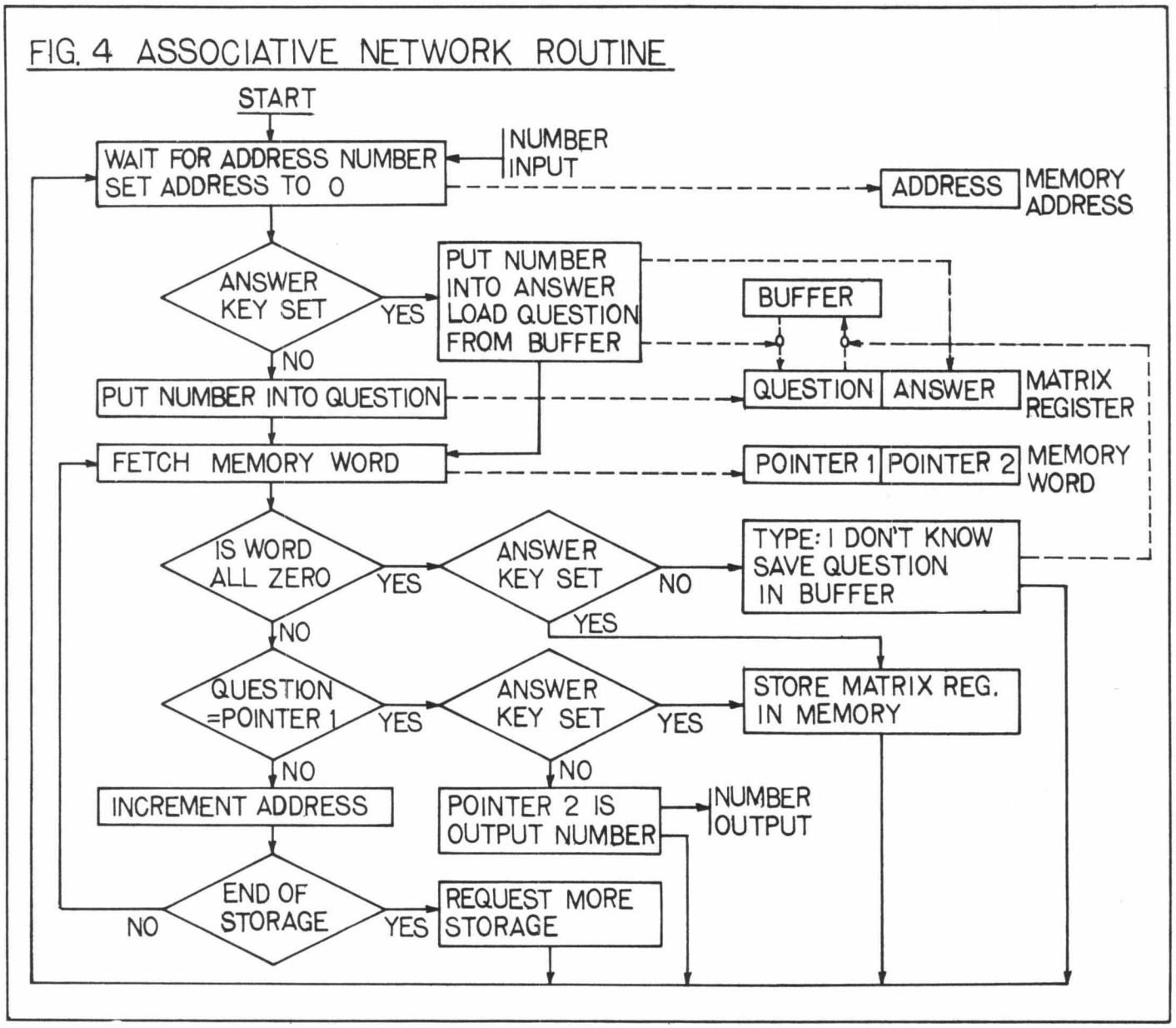
Retrieval Routine

If we teach a vocabulary to this network, each English word would be identified by a unique address number. In the example in Fig. 3 it is ROSE-6, ROBOT-10, IDN-14, and IDEA-17. These numbers are of course unique to each network and depend on the sequence of the original education. Each word is identified by a single address number which may be used in turn to retrieve the original input word. For example, the word IDEA may be retrieved from the network if we know its unique address number 17.

The retrieval routine is shown in Fig. 3.

Let's follow the steps of retrieving the word IDEA from its address 17. The input to this routine is address 17. We fetch the matrix stored in memory address 17 and load it into the matrix register. In the gate portion we find an output character (space) which is the last character after the word IDEA. The retrieval of a word is always in reverse order. IDEA becomes AEDI but this will be corrected later by a first-in last-out buffer. The letter SP is an output data character which is sent to the output buffer or output device. The matrix SP-16 also supplies pointer 16 which points to the next letter in the sequence. Fetching the next matrix from memory location 16 we will find another data character plus another pointer. Following this sequence (address 17, 16, 15, 12, 11) we will finally reach pointer 0 by which time we will have retrieved the entire word IDEA. Address 0 was the starting address of this network. If we recognize this pointer 0 we have completed the retrieval sequence.

Every word in the English language may therefore be converted into a unique address number which may later by used to retrieve the original word.

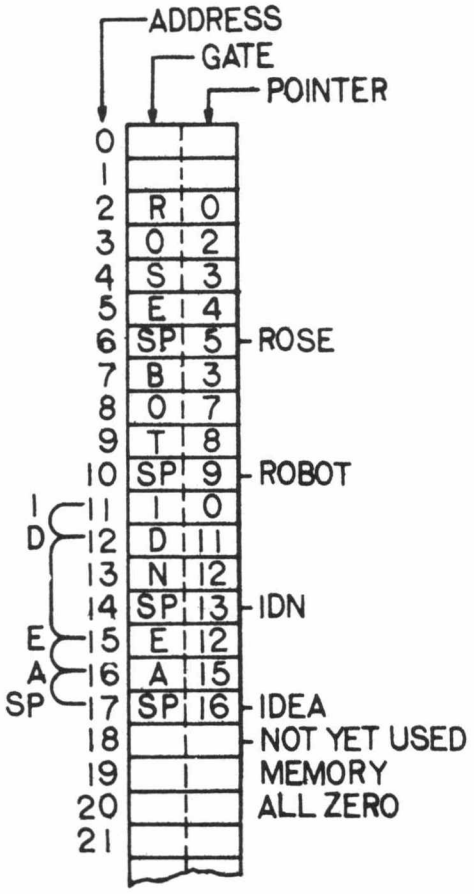


Associative Network

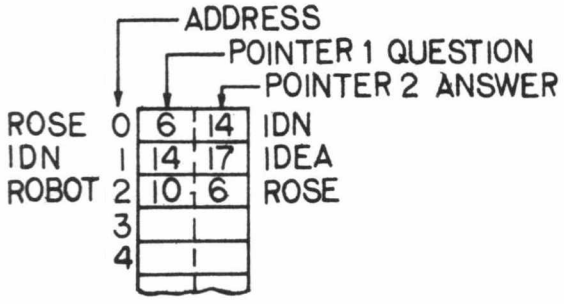
This network is of a different class from the serial network explained so far. Its purpose is to connect separate action sequences such as question-to-answer. This network should be physically separate from the serial networks and stored in a different section of the mass memory. The matrix register in the microprocessor will now hold an input pointer and an output pointer each at least 2 bytes wide. The input to this network is the address number from the serial networks. It will be a single address number representing an entire question or answer sentence. An external key will tell this routine whether the typed-in-sentence is meant to be a question or answer. If it is a question to the system we load the address number into the question portion of the matrix register. The microprocessor will scan the memory and examine each word in sequence. It tries to find a word stored in memory in which the input pointer 1 is the same as the "question" portion of the matrix register. Only half of the memory word needs to be compared. If the end of the used portion of the memory field (memory word all zero) is reached without finding such a word, the question address number is stored in a buffer and the system will type-out a message such as I DON'T KNOW. This means that the answer to this particular question is not yet known and a human teacher must supply it. The following sentence will most probably be an answer, which will then be learned by the system.

If, during the memory scan, a memory word is located in which the input pointer is identical to the "question" in the matrix register, an answer to this particular question is available. It is stored in the adjacent output pointer 2 in that same memory word. This output pointer 2 will be used to retrieve and type-out the answer sentence. If we reach the end of the storage device we need to add more storage before the system is able to learn any more information.

FIG.5 STORED NETWORK EXAMPLE



SERIAL NETWORK



ASSOCIATIVE NETWORK

A particular input question is therefore either already stored (learned), in which case the routine will supply the answer address number, or if the answer is not yet learned, the answer address number is stored in a buffer. The human teacher may then activate the answer key and type-in the answer sentence. This answer sentence is converted like the question into a single address number by the serial network and supplied to the associative network routine. The answer address number is loaded into the answer portion of the matrix register. The question portion is loaded from the question buffer (previous question).

The memory field is then scanned to find a memory word containing an input pointer which is identical to the question portion of the matrix register. If we find such a memory word we would store the matrix register into that memory word. We have therefore updated the system's knowledge by replacing the old answer with the new answer.

If we reach the end of the present memory field (memory word is all zero), we store the matrix register into that memory word. The system has learned another question-answer by storing it in its memory.

System Integration

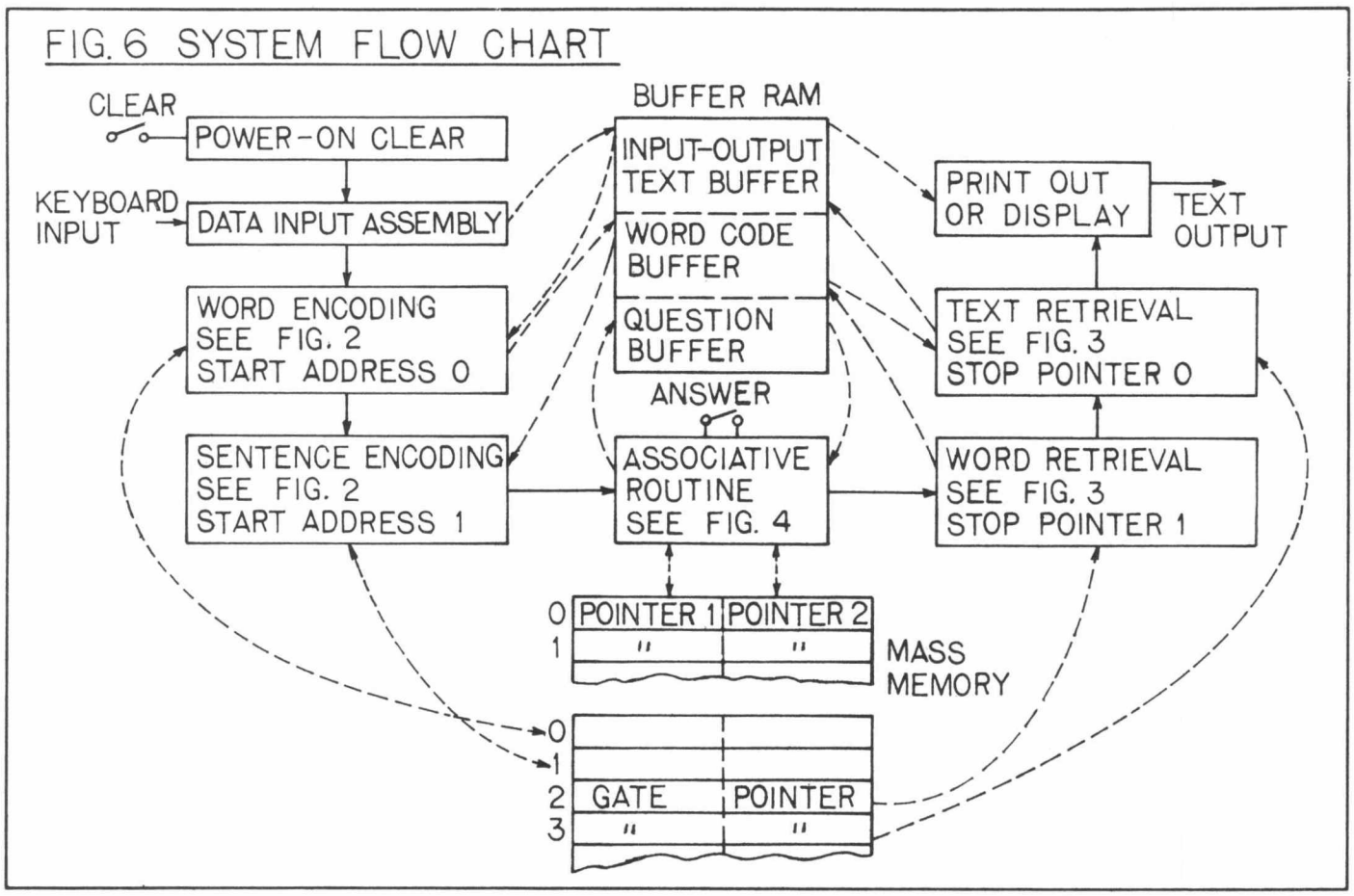
We are now ready to put the system together. Some input/output control routines are also required. Figure 6 shows the system flow chart. The 1K RAM in the system could be eliminated if each input charcter is treated immediately and you have a buffered ouput display.

A power-on routine is required in every computer system to set-up registers and clear the buffer storage. This routine should also be accessible by depressing a CLEAR key (interrupt) in case something goes wrong in the process.

The input assembly routine should accept the input from the keyboard and store the character in the RAM buffer in forward direction. The routing should recognize a special end-of-sentence signal such as carriage return, line feed, GO key or question mark in order to exit this routine. The program should be smart enough to recognize superfluous carriage returns, line feeds or space used solely to advance the paper and handle backspace and correction.

The word encoding routine is a serial network routine as shown in Figure 2. It fetches the data character from the buffer RAM and converts each word into its unique address number which is then stored in the word address buffer. This routine may use starting address 0 in the mass memory.

The sentence encoding routine is another serial network which converts each input sentence, consisting of several individual words, into a unique address number. Instead of using characters from the keyboard as input, this routine uses the word address numbers from the previous routine. These word address numbers are now stored in the word address buffer. It may look strange at first glance that address numbers may be used as input data, but this routine operates otherwise just like the one in Figure 2. For automatic interleaving of this network with the previous word encoding network, starting address 1 should be used. Note also that address numbers are at least 2 bytes wide as compared with 1 byte ASCII character.



The associative routine as described in Fig. 4 will accept the address numbers from the previous routine and process it according to the answer keys mentioned earlier. It will either find an associated answer address and hand it to the sentence retrieval routine or otherwise indicate a lack of knowledge by typing out I DON'T KNOW. The question address number should be preserved in the RAM buffer.

The sentence retrieval routine will retrieve the word address numbers from the sentence address number as described in Fig. 3. The retrieved word address numbers should be stored in the word address buffer RAM, but in reverse order. The first address number retrieved should be stored in the last location.

The word retrieval routine will take the word address numbers from the RAM buffer starting with the last word. It will convert each word address number back into its original text and store the resulting character in the data input buffer starting from the last location.

The last printout routine will take the accumulated text in the data input buffer and print it on the output display. Note that the text does not start at address 0 and the empty slot in front must be ignored.

Operation

After completing the control routines and debugging you are now ready to enter the new age of knowledge processing and forget programming once and for all. From now on you will only work with keyboard and display, and the internal operation of the computer no longer concerns you.

Let's see how such a computer would operate. As an example let's assume that the computer is used as an English-German language translator. This would be only a very crude device which must be much refined before it becomes useful.

A human teacher would type an English sentence into the computer. If a word or sentence is not yet contained in the network it would be added automatically. If the translation of that English sentence is already stored the computer would typeout the German translation. If it is not yet known the human teacher would type the German translation or answer into the computer, which would automatically absorb it. Once a translation is learned by the computer it will be remembered but never learned again. The translations may be entered in any random order and it does not matter if the teacher repeats himself frequently. This is indeed how a human child would be educated, but this computer has a mathematically precise, unlimited and reliable memory and total recall. It never forgets.

A human teacher setting out to teach language translation to the computer would just type in English and German sentences. At first the computer would absorb knowledge very rapidly but would slow down after a while because it remembers most words and translations and the teacher would repeat himself more and more frequently. After weeks or months most of the teacher's knowledge in this subject would be absorbed by the computer.

In order to share his knowledge, or make money, the teacher could make a memory dump of this network on tape for sale. By loading that tape into a second computer we obtain a system containing the identical knowledge.

A much more interesting copy is through a knowledge dump. In this dump the input pointer (question) and output point (answer) in the associative network would be translated backwards through the serial networks to obtain a copy (tape) containing all the teacher's knowledge in clear text. Each memory word in the associative memory is translated into a clear text question followed by a clear text answer. Such a tape would contain more data but would also be more useful.

Imagine that a number of teachers would teach language translation to their individual computers. After some time, each teacher would make a knowledge dump of his tape and go to a conference. By feeding each tape as if it were keyboard input into a common computer, this computer would absorb all the new knowledge from each teacher in a matter of minutes. Its memory requirement would expand only moderately since most of the knowledge would be redundant. This computer would then contain the combined knowledge from all the teachers and would therefore be more knowledgeable than any human.

The lesson is this: the computer can become smarter than yourself and any knowledge stored in inifinite dimensional networks is universal. All accumulated human knowledge will be merged in all knowing super computers and the stored knowledge can be constantly updated and expanded. With its super reliable, unlimited capacity memory which never forgets, these computers may quickly surpass human capabilities.

Instead of language translation you could use your computer for the following tasks:

Question	Answer
Word from an encyclopedia	Information in that subject
Record subject	Sports records, dates
Dictionary word	No answer if misspelled
	Description of term
Question, Problem	Guessing game
Calendar date	Personal diary information
Calendar clock automatic input	Appointment, reminder, birthdays
Article name	Inventory, stock, price, delivery
Anticipate question in an exhibit	Give funny answers. Pretend computer to be self aware

A secret text message could be compressed and put into an unbreakable code by sending address numbers instead of text words. These address numbers are easily retrieved using an identical copied system.

Outlook

This simple system is just the befinning of a comprehensive computer revolution. Other network classes result in self-organizing sense organs such as eves which can really see, and robots which are educated, by simply guiding their hands through the motion. The basic theory of knowledge processing and infinite dimensional networks is very extensive and may actually explain human brain functioning. By building such a computer now you can join this revolution and contribute to it. Before you start mass producing your system however check with the author about patent rights.

Reference

"Here comes the brain-like self-learning no programming computer of the future" The first West Coast Computer Faire *Conference Proceedings* Computer Faire Box 1579 Palo Alto CA 94302

A CLOSER LOOK AT THE MC6800

BY CURTIS ROADS, EDITOR
Computer Music Journal
Box E
Menlo Park, CA 94025

(A similar article appeared in Vol. II, No. 3 of Computer Music Journal*)*

In the next few paragraphs, I'm going to take a look at what appears to be one of the more important new microprocessors—the Motorola MC68000—from a critical vantage, since, in the past, I've been one who has accepted the assets (high-performance, mature and elegant software tools) and liabilities (cost) of traditional minicomputers as against microcomputers for applications requiring a disk operating system, multiprogramming, and a complement of software utilities and high-level languages.

In approaching the literature on the MC68000, a most encouraging sign is the statement: "The MC68000 hardware design was heavily influenced by advances in software technology"(1).

The MC68000 is the latest stage in a series of microprocessors that began with Motorola's 6800 chip. An intermediate stage has been the 6809 chip, a kind of pseudo 16-bit machine, with 8-bit registers and 16-bit arithmetic and addressing. Given the unwieldy 6800-line instruction set, one is inclined to believe the Motorola salespeople when they say that the design of the more rational MC68000 involved outside consultants from a couple of leading (here unnamed) university computer science departments. In (2), an example program is presented, coded in 6800 code, 6809 code, and 68000 code. The 6800 program takes 45 instructions, the 6809 program takes 13 instructions, while the 68000 takes only 10. While the 6809 is five times as fast as the 6800, the 68000 is twenty-five times faster. The raw machine speed of the MC68000 appears to be favorably compared to a fast minicomputer such as the PDP-11/45. This looks good on paper at least, but there are many ways to measure computer throughput, and without any software to run on it, it is a rather abstract measure.

Starting with the sound premise that "Total software costs will dwarf hardware costs in the 1980's" (2), the MC68000 appears to have been designed to provide a more hospitable implementation environment for the programmer.

The instruction set features 63 *instruction types* (operands) which are transformed into executable instructions via the specification of *data types* and *addressing modes*. For example, there are many variants of the MOV (move) instruction; the move-word operand is written MOV.W, while the move longword operand is written MOV.L. The instruction set permits register-to-register, register-to-memory, memory-to-register, and memory-to-memory transfers.

The six data types handled by MC68000 operands include: bits, binary-coded-decimal digits (nibbles), ASCII characters, bytes (8-bit), words (16-bit), and longwords (32-bit).

Five basic addressing modes (register-direct, register-indirect, absolute, immediate, program-counter relative) can be combined with post-incrementing, pre-decrementing, and indexing to form fourteen different addressing modes.

Thus, by combining instruction operands with various data types and addressing modes, the MC68000 effectively executes over 1000 instructions.

The architecture of the MC68000 is mixed; while only 16 bits may go in and out of the machine, all internal data paths are 32 bits wide.

The MC68000 is characterized by eight 32-bit *data* registers and eight 32-bit *address* registers. All sixteen registers may be used as data registers for word or longword operations or as index registers. The first eight registers may also be used for byte operations, while only the second eight registers may be used as base address registers or stack pointers.

The program counter is 24 bits wide, permitting direct addressing to 16 M-bytes without paging. The program status register is 16 bits wide.

Like the PDP-11 (which the machine resembles in many ways) the MC68000 has eight levels of vectored interrupts, as well as both hardware and software vectored traps. It is said to have full BCD arithmetic as well as hardware multiply and divide instructions.

The MC68000 literature states: "Special emphasis has been given to the instructions set's support of structured, high-level languages." PASCAL is singled out in (1) and (4).

The STM (store multiple registers) and LCM (load multiple registers) instructions should reduce processor overhead for procedure calls. Other instructions that may simplify modern programming include PEA (push effective address onto stack) and LEA (load effective address from stack) as well as the transfer instructions RTR (return and restore), JSR (jump to subroutine), BSR (branch to subroutine), and RTS (return from subroutine). Yet another instruction CHK may be used to check illegal array indices.

Certain sections of the MC68000 preliminary literature are unclear. Two ambiguous instructions called LINK and UNLINK are said to "allow linked lists to be manipulated on the stack." Another highly ambiguous statement assures us that the MC68000 has "a full complement of block/string operations," without any further detail.

Like DEC's PDP-11/34 (and 11/40, 11/45, etc.) the MC68000 is capable of running in a privileged mode called *Supervisor* mode, in which certain privileged instructions may be executed. For example, the operating system would run in *Supervisor* mode, and could manipulate the Memory Management registers, which are unavailable to the non-privileged *User* mode.

An area of ambiguity lies in the fact that there is no mention of a separate set of registers for *Supervisor* mode vis-a-vis *User* mode. It appears that it would be necessary then to save the registers before a context-switch, (i.e., a switch from *User* to *Supervisor* mode, or vice versa) and restore them afterwards.

However, one of the vaguest areas in the MC68000 literature is the bus structure. The literature does say that communication between the cpu and peripherals works only after an interlocking process over the bus (like the PDP-11). An external Bus Arbitrator is said to resolve conflicts over bus use for multiprocessor systems involving a shared bus and memory. There is no detail as to how this arbitration is implemented.

The Memory Management unit is provided *not* as a device for virtual memory extension (as it is on the PDP-11, with its more restricted memory addressing) but rather, it is provided primarily for multi-user protection, i.e., so that various users do not tread upon one another's address space. It is said to manage variable-length pages, and provides for dynamic relocation of active pages (as the operating system permits). It also appears to provide for pure, reentrant code segments, through a distinction between code and data memory spaces (cf. I and D space on the PDP-11). However, once again, there is no discussion of the implementation mechanism.

An external DMA (direct-memory-access) controller is also available; this is necessary for digital-to-analog conversion ports.

From the standpoint of its hardware organization (fast, low-cost, rational architecture) the MC68000 would appear to make a very interesting microcomputer network machine. For example, it is becoming more and more feasible to offload certain software functions to dedicated microprocessors. A most obvious application is the elimination of purely software device drivers. Already, three intelligent device drivers are becoming common: the intelligent terminal, the self-testing operator's console/diagnostic controller, and the network controller. In this latter case, the primary computer treats the network as another device; output from the primary computer is re-structured according to a network protocol and is then passed along to other network nodes. At the other end, a dedicated microprocessor unscrambles the message from the protocol, and passes the message onward to another primary computer. The primary computer could be another MC68000.

What about software? In the present literature, this is another very vague domain of discourse. On the one hand, we are told that the MC68000 was developed in order to ease the coming "software impasse," and on the other hand, we are told that "with the MC68000, you have available a complete, compatible system of software and hardware . . . Software support includes a disk operating system, debug aids, assemblers, and high-level languages—PASCAL, MPL . . ."(4) No dates or details are given—could this be an impasse? Perhaps we can get a clue from their 6809 chip, which is presently available. Its 1979 software release will feature (get ready)—an 8K BASIC interpreter. Ah, well, maybe next year. Hopefully, the MC68000 group will set their sights a little higher than this. If so, they should be working hard right now, as, according to all of the preliminary literature, the MC68000 is due out in the second quarter of 1979. However, sources close to the MC68000 developers in Austin say that the system will more likely not be released until early 1980.

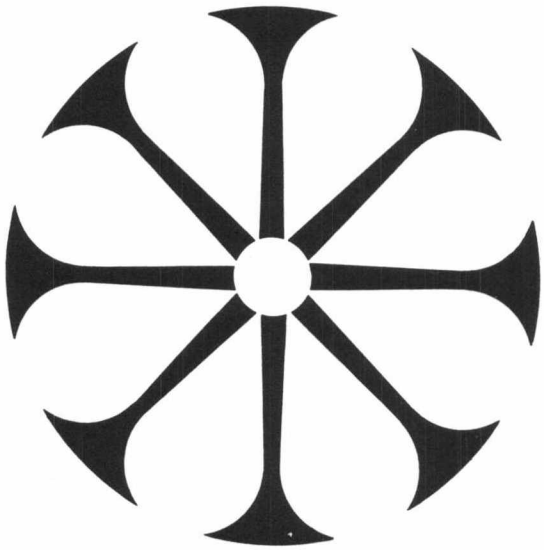
REFERENCES

(1) "Introducing the MC68000," (1978) Motorola, Phoenix.

(2) "16-bit MPU's: Past, Present, and Future—the 6800 Approach," R.G. Daniels, G.J. Summer—Motorola, 1978 WESCON Session 20, Los Angeles.

(3) "16-bit MPU's to Offer Performance of Mini's," D. Bursky, (1978), *Electronic Design*, Vol. 26, No. 18, Rochelle Park.

(4) "Complex Systems are Simple to Design," I. Lemair, R. Nobis—Motorola, (1978) *Electronic Design*, Vol. 26, No. 18, Rochelle Park.



Label BASIC
A BASIC Preprocessor

BY SYLVAN RUBIN
James S. Mills & Associates
P. O. Box 3074
Santa Clara, CA 95051

In this article, a preprocessor for BASIC will be described that improves the writeability and readability of BASIC so much that it's almost like a new language.

Nearly all versions of BASIC share two features that obstruct their effective use in writing fairly large or complex programs. These are the sequential line numbers required for all program statements, and the very restricted variable names allowed. These holdovers from the original Dartmouth BASIC were once useful, because the line numbers provided an editing capability on systems that lacked real editing software, and the short variable names used less memory, and could be preallocated to a fixed data space. While some very elementary present-day microcomputers may still require such drastic simplicity, most systems are provided with a competent editor (which is typically much smaller than the BASIC interpreter), and most programmers can write programs with an editor far more easily than by typing statements directly into a BASIC interpreter.

However, the line numbers seriously interfere with using any editor. Major rearrangements of existing programs are impractical, because so many line numbers and line number references have to be changed. Assembling programs from tested subroutine library files with the editor is impractical, again, because all the line numbers have to be updated.

The line numbering is also annoying while composing a new program on paper, because the programmer has to keep track of the numbers for lines not yet written. The effort spent on this "cross-referencing" of line numbers while simultaneously trying to keep the program logic and flow of control clearly in mind is wastefully distracting. The inevitable result is more programming errors. The short variable names are less of a problem, but they too, lead to unnecessary errors, because of their lack of self-documentation. Program maintenance and revision are especially handicapped by dealing with variables whose purposes are often forgotten even by the program author. Most programmers in BASIC can recite harrowing experiences with bugs resulting from reusing a variable that should have been left alone.

Most users of BASIC think they have to live with BASIC as it is. Not true. It isn't necessary to write new BASIC interpreters to eliminate these two problem areas. LABEL-BASIC, with labels where needed for GOTO and GOSUB statements, or where desired for "labelling" code, and meaningful variable names, can be translated into a standard BASIC that runs under existing interpreters (or compilers). The LABEL-BASIC preprocessor works with any BASIC. It does not change anything in the BASIC programming rules, or the BASIC syntax or punctuation. It simply translates labels into correct line numbers, and long variable names into short ones. The results of this simple preprocessing are impressive. Suddenly, it becomes easy to write independent BASIC program modules. The program editor is turned into a useful program construction and assembling tool. The programs actually describe, in simple English, what they are doing.

It may be hard to believe that computer programs can be so transformed by such simple means. The short example in this paper shows part of what can be done. A complex subroutine, such as this one, is difficult to describe clearly in any programming language. The labels and variables in this example are a meaningful explanation of how this sort of algorithm works. Large programs in LABEL-BASIC, with deep subroutine structure, become far more comprehensible than can be achieved by liberal use of comments alone. The labels are guideposts to program flow; and the variable names are explicit data descriptors.

The sample program in this article is a version of QUICKSORT, based on the paper by R. Sedgewick in *CACM*, v. 21, p. 847, Oct. 1978, on various efficient implementations. This version is useful for sorting large tables of data in memory, and includes taking the pivot value from the center of each sublist to avoid the deleterious effect of pre-ordering on the efficiency of QUICKSORT. It was chosen as an example because of its complexity, and because it may be a useful addition to any programmer's library. Fig. 1 shows the LABEL-BASIC version, and Fig. 2 shows the result of translating it with our translator.

Subroutines such as this one can be filed as separate program text files and appended to a main program with a text editor. It is only necessary to use the correct parameter names in the calling routine to properly pass data. The LABEL-BASIC translator will link all the subroutines together with correct line numbers and variable names.

LABEL-BASIC uses a few special characters to identify the labels, label-references, and variables. Labels are always at the beginning of a line, and are terminated by back-slash, \. Label references in statements, such as GOSUB [KEY SELECTION] or GOTO [THAT'S ALL], use the two square brackets as delimiters. These three characters are not used by most BASICs. Also, this allows the labels to include blanks or other punctuation for greater expressiveness. Variable names are distinguished from everything else by prefix characters. Numeric variable names begin with %, as in %TOTAL, or %SIZE(%INDEX). String variables begin with $, as in $TEXT. These two prefix characters do not usually interfere with other uses of these characters in BASIC, such as ordinary string variable names, because the standard BASICs usually use them as suffix characters, like A$. As a matter of fact, both LABEL-BASIC variable names and ordinary BASIC variables can both be used in the same LABEL-BASIC program. To take care of special cases where % or $ can be ambiguous, such as hexadecimal numbers ($A10), two instructions to the translator are available to redefine either the numeric or string prefix, PREFIX% and PREFIX$. The variable names must be entirely alphanumeric characters; any non-alphanumeric character or blank or end-of-line terminates the name.

Continued on pg. 42

PIAs for KIM

BY JOEL SWANK
4655 SW 142nd #186
Beaverton, OR. 97005

Dear DDJ:

*Enclosed is an article about a small project I just completed. My last two articles generated some inquiries about how to hook PIAs to KIM. Since this normally would require expansion of KIM, I hesitated to do more than offer suggestions. Then this idea came to me and I liked it so much that I implemented it and wrote the article. No sample program is included. The hardware can be tested by using either of my previous programs (*DDJ *#28 & #30) and changing the PIA addresses; $DA to $04 in the IC tester, and $D8 to $04 in Tiny Grafix.*

When I first began expanding my KIM, I soon ran out of I/O ports. KIM's 15 bits of user I/O can drive a fairly sophisticated device, but what if you have two such devices? I chose to add PIAs (Peripheral Interface Adaptor), specifically, the Motorola 6820 PIA (identical to the MOS Techonology 6520). The 6820 contains two eight bit ports with two control lines each. This device is designed to operate connected to the system bus and occupy part of the CPU's address space. Each eight bit port takes up two memory addresses, so you can get a lot of I/O without using much address space. It is a MOS device and therefore uses little power, and is compatible with most microprocessor busses. The cost is relatively low– I've seen them advertised for under $4. With programmable features like automatic handshake, pulse mode, and interrupt capability, they are hard to beat.

Hardware

Since it is an MOS device, it is ideal for connection to the KIM expansion connector. Logically, the PIA looks to the CPU like a four byte memory. Like a memory, it must be connected to the data and address busses. The data bus connections have tri-state buffers and can be connected directly to KIM's dual direction data bus. The 6820 has only two address lines, so most of the address decoding must be external. To save external hardware, some address lines can be left undecoded. This will cause it to occupy more than its minimum four bytes. KIM.s onboard 74LS145 provides decoding down to the 1K level. K1 through K4 are available on the application connector. If you haven't added memory there, and you don't mind using up 1K of memory for the PIA, it can be connected to KIM with a 1 resistor interface. Figure 1 is a list of connections to do this. Connecting the PIA in this way will make it occupy addresses $400 – $403. It will also occupy addresses $404 – $407, $408 – $40B, etc. all the way to $7FC – $7FF, because address lines A2 through A9 are not used in the selection. The resistor is connected between pin 23 and +5VDC to pull up the open collector output of the 74LS145 to logical false when K1 is not selected. My KIM board uses 560 ohm resistors for this, but any value close to that will work. The connections to IRQ (pins 37 and 38) are not required and can be disconnected when not in use. This will prevent accidental interrupts from occurring should an errant program activate the PIA. The wires on the data line should be kept as short as possible to prevent noise. Noise on the data lines can cause problems all over the system.

Programming

The 6820 is divided into 2 eight bit ports (called A and B), that operate independently. Each bit of each port is individually programmable to be either input or output. Each port has 3 addressable registers. Port A occupies addresses $400 and $401. Port B occupies addresses $402 and $403. Since ports operate almost the same, I will look at port A; all said is true for port B, except where noted.

Address $400 is the control register (CRA) for port A. All modes of operation are controlled by this register. Figure 2 is a summary of the use of the control register. Address $401 is used to access both the data direction register (DDRA) and the data output register (ORA). It is also used to read the data present on the external lines (PA0–PA7) when they are programmed as inputs. The first step in using the PIA is to set the data direction register. This register has eight bits, each corresponding to 1 of the external lines. To set PA0 to input bit 0 of DDRA is set to 0. To make it an output bit 0 is set to 1. Bit 2 of the control register is used to select whether $401 is connected to the direction register or to the I/O register. Usually a program will set the direction register during initialization and never access it again, so control bit 2 is left set to 1 (I/O register access) most of the time.

When using the port as output, data to be output is stored in the I/O register and it appears on the external lines for all lines programmed as outputs. Lines programmed as inputs are not affected. When used as input the CPU can read the I/O register and each bit of the retrieved byte wil reflect the state of the external lines for all lines programmed as inputs. The lines programmed as outputs will reflect the last value written there by the CPU.

In addition to the eight external data lines, each port has two control lines used for data strobe and handshake signals. These functions are controlled by the control register. Line CA1 is an input only line and line CA2 is programmable as input or output. CRA bits 6 and 7 can be read to detect an active transition of CA2 and CA1 respectively. Bits 3 – 5 are used to control CA2 and bits 0 – 1 are used to control CA1. The options that can be selected for CA1 are the active transition for that line (that is whether it responds to a negative going change in voltage or a positive going change in voltage), and whether this change results in an interrupt to the CPU being generated. CA2 has 3 bits to control it because it can be either input or output. When in input mode, it acts the same as CA1. When in output, it has 3 possible modes of use. Direct mode is when this line is under direct control of the CPU. Its value is set by altering bit 3 of the control

register. Pulse mode is where an automatic 1 microsecond pulse is generated each time the I/O register is read. Here is where port A and B differ. Port B's pulse is generated only on a write to the I/O register. The third mode is handshake mode. Using this mode CA2 is set high on an active transition of CA1 and low by a read of the I/O register. Here again port B is sensitive to a write instead of a read. Two PIAs could be connected in handshake mode, A port to B port, and transfer data very rapidly. For a more detailed description of the workings of this device I recommend the MOS Technology Hardware Manual.

With the above notes you should have no trouble getting one or more PIAs working on KIM. Then it will be possible to have several I/O devices connected at once. Seeing the cost effectiveness and ease of interface of the PIA, I wonder why there are no S-100 or SS-50 PIA boards available. It seems a very inexpensive way to get a lot of I/O capability. Also if more manufacturers would make their peripheral devices compatible to 1 or more parallel eight bit ports, instead of a specific buss design, there would be a lot more interchangability of hardware. Each bus would have its PIA interface and a multitude of devices would be compatible to the PIA.

TO CONNECT A PIA AT ADDRESS $4ØØ

PIA PIN#	FUNCTION	KIM CONNECTION	FUNCTION
21	R/W	E-V	R/W
22	CS1	E-21	+5V
23	CS3	A-C	K1
24	CS2	E-21	+5V
25	ENABLE	E-U	Ø2
26	D7	E-8	D7
27	D6	E-9	D6
28	D5	E-1Ø	D5
29	D4	E-11	D4
3Ø	D3	E-12	D3
31	D2	E-13	D2
32	D1	E-14	D1
33	DØ	E-15	DØ
34	RES	E-7	RST
35	RS1	E-B	A1
36	RSØ	E-A	AØ
37	IRQB	E-4	IRQ
38	IRQA	E-4	IRQ
1	VSS	E-22	GND
2Ø	VCC	E-21	+5V

FIGURE 1

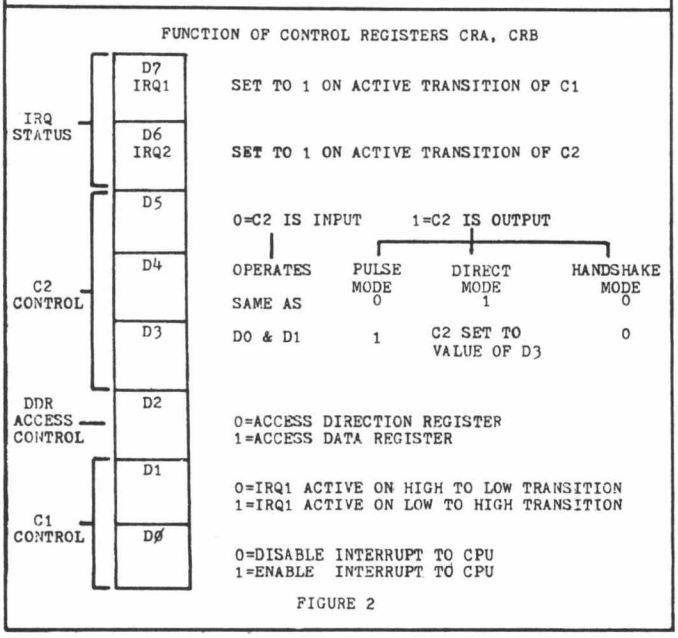


FIGURE 2

Continued from pg. 40

We have implemented two versions of the LABEL-BASIC translator. Both are written in LABEL-BASIC; one on an interpreted BASIC, the other on a compiled BASIC. Both use a disk-based system, and translate disk file to disk file, so that only the translator program and its symbol and address tables need be in memory. Both versions are for the Motorola 6800 CPU, running under the Smoke Signal DOS. Since the source programs are BASIC, they are easily convertible to any other computer.

For information on the availability of LABEL-BASIC for various popular operating systems, write to Unbounded Computing, P. O. Box 60725, Sunnyvale, CA 94088. For a copy of the manual for LABEL-BASIC, please enclose $1.00 for mailing costs. The LABEL-BASIC name, translator programs, and manual are copyrighted.

```
!  QUICKSORT SUBROUTINE   SYLVAN RUBIN   1 DEC 1978
!  DATA TO BE SORTED IN ARRAY: %TABLE(%FIRST:%LAST)
!  SENTINEL: %TABLE(%FIRST-1)=SMALLER THAN ANY DATA ELEMENT
!  STACKS: %LOSTACK & %HISTACK EACH NEED LOG2(%LAST-%FIRST) ELEMENTS

QUICKSORT\ %LOWEND=%FIRST: %HIGHEND=%LAST: %STACKPTR=0
PARTITION\ IF %LOWEND>=%HIGHEND THEN [POP STACK]
       %CENTER=(%LOWEND+%HIGHEND+1)/2  ! USE CENTER FOR PIVOT
       %TEMP=%TABLE(%CENTER): %TABLE(%CENTER)=%TABLE(%HIGHEND)
       %TABLE(%HIGHEND)=%TEMP
       %LOW=%LOWEND-1: %HIGH=%HIGHEND
       %PIVOT=%TABLE(%HIGHEND): GOTO [SCAN-L]
EXCHANGE\ %TEMP=%TABLE(%LOW): %TABLE(%LOW)=%TABLE(%HIGH)
       %TABLE(%HIGH)=%TEMP
SCAN-L\ %LOW=%LOW+1: IF %TABLE(%LOW)<%PIVOT THEN [SCAN-L]
SCAN-H\ %HIGH=%HIGH-1: IF %TABLE(%HIGH)>%PIVOT THEN [SCAN-H]
       IF %LOW<=%HIGH THEN [EXCHANGE]
SWAP PIVOT\ %TEMP=%TABLE(%LOW): %TABLE(%LOW)=%TABLE(%HIGHEND)
       %TABLE(%HIGHEND)=%TEMP
PUSH STACK\ IF (%HIGH+1-%LOWEND)>(%HIGHEND-%LOW) THEN [STACK LOW]
STACK HIGH\ IF %LOW+2>%HIGHEND THEN [SHIFT HIGHEND]
       %STACKPTR=%STACKPTR+1
       %LOSTACK(%STACKPTR)=%LOW+1: %HISTACK(%STACKPTR)=%HIGHEND
SHIFT HIGHEND\ %HIGHEND=%HIGH: GOTO [PARTITION]
STACK LOW\ IF %LOWEND+1>%HIGH THEN [SHIFT LOWEND]
       %STACKPTR=%STACKPTR+1
       %LOSTACK(%STACKPTR)=%LOWEND: %HISTACK(%STACKPTR)=%HIGH
SHIFT LOWEND\ %LOWEND=%LOW+1: GOTO [PARTITION]

POP STACK\ IF %STACKPTR=0 THEN RETURN
       %LOWEND=%LOSTACK(%STACKPTR): %HIGHEND=%HISTACK(%STACKPTR)
       %STACKPTR=%STACKPTR-1: GOTO [PARTITION]
```

Fig. 1. QUICKSORT in LABEL-BASIC.

```
50  A=A0: A1=A2: A3=0
60  IF A>=A1 THEN 280
70 A4=(A+A1+1)/2
80 A5=A6(A4): A6(A4)=A6(A1)
90 A6(A1)=A5
100 A7=A-1: A8=A1
110 A9=A6(A1): GOTO 140
120  A5=A6(A7): A6(A7)=A6(A8)
130 A6(A8)=A5
140  A7=A7+1: IF A6(A7)<A9 THEN 140
150  A8=A8-1: IF A6(A8)>A9 THEN 150
160 IF A7<=A8 THEN 120
170  A5=A6(A7): A6(A7)=A6(A1)
180 A6(A1)=A5
190  IF (A8+1-A)>(A1-A7) THEN 240
200  IF A7+2>A1 THEN 230
210 A3=A3+1
220 B(A3)=A7+1: B0(A3)=A1
230  A1=A8: GOTO 60
240  IF A+1>A8 THEN 270
250 A3=A3+1
260 B(A3)=A: B0(A3)=A8
270  A=A7+1: GOTO 60
280  IF A3=0 THEN RETURN
290 A=B(A3): A1=B0(A3)
300 A3=A3-1: GOTO 60
```

Fig. 2. QUICKSORT, after translation into BASIC.

Letters

RESPONSES TO THE UBIQUITOUS MR. GABRIELSON:

Dear *Doc*:

This is in response to the letter from Mike Gabrielson in *Dr. Dobb's* #29, p.3, in which he inquires about efficient reentrant routines to PUSH and PULL all 8080 registers.

The routines listed below seem to me to be as short as is possible:

```
000.000                              ORG     0

000.000  042 014 000  PUSHER  SHLD    HSAVE
000.003  343                   XTHL
000.004  365                   PUSH    PSW
000.005  325                   PUSH    D
000.006  305                   PUSH    B
000.007  345                   PUSH    H
000.010  052 014 000           LHLD    HSAVE
000.013  311                   RET
000.014  000 000       HSAVE   DW      0

000.016  341           PULLER  POP     H
000.017  301                   POP     B
000.020  321                   POP     D
000.021  361                   POP     PSW
000.022  343                   XTHL
000.023  311                   RET
```

COMPARISON OF THESE ROUTINES WITH THE ORIGINAL ONES:

			BYTES	CYCLES
PUSHER	–	OLD	16	132
	–	NEW	14	104
PULLER	–	OLD	7	86
	–	NEW	6	68

Sincerely,
William W. Moss

1507 Riverview Lane
Bradenton, FL 33505

Dear *Dr. Dobb's*,

In the October 1978 issue of *Dr. Dobb's* you published a letter from Mr. Mike Gabrielson showing three routines for loading either a hexadecimal 00, 01, or FF in each location of 64K of RAM in a 6800 system.

As advertised, the routines will work assuming that the M6800 system has 64K of RAM. This is almost impossible since the 6800 needs some memory locations reserved for memory mapped I/O operations. Certainly no one would construct a computer that neither accepts nor outputs any information, although I am sure that some systems have had such properties during construction of same.

I am enclosing a routine that will load all low order RAM with a desired constant and may be used with the Southwest Technical Products Co. MC6800 system. I assume that the user has either MIKBUG, SWTBUG, SMARTBUG, RT/68MR, or HUMBUG type operating systems which begin execution at location $E0D0. The program is loaded into memory as indicated in the scratch area starting at location $A010. This allows the program to execute and return control back to the MONITOR, allowing the user to immediately start loading desired program or programs into memory.

The accompanying memory dump of the routine is from another program I will submit for publication in *Dr. Dobb's* next month. These are examples of small routines that are in my book to be published by *dilithium Press* (note: the d is lower case).

P.S. The output is from a resident 6800 editor and two pass assembler that resides in less than 4K of memory.

Sincerely,
Dr. Chuck Adams

North Texas State University
Department of Computer Sciences
Denton, Texas 76203

```
A
   1                *
   2                *     THIS ROUTINE LOADS ALL OF LOW MEMORY WITH A
   3                *
   4                *     CONSTANT LOADED INTO ACCUMULATOR A, ACCA.
   5                *
   6                *     WRITTEN FOR SOUTHWEST TECHNICAL PRODUCTS CO.
   7                *
   8                *     M6800 SYSTEM OR SIMILAR SYSTEM WITH SCRATCH
   9                *
  10                *     RAM LOCATED ABOVE LOW ORDER MEMORY.
  11                *
  12                *     DR. CHUCK ADAMS
  13                *     NORTH TEXAS STATE UNIVERSITY
  14                *     DEPT. OF COMPUTER SCIENCES
  15                *     DENTON, TEXAS  76203
  16                *
  17          E0D0 MNTR   EQU    $E0D0      STARTING ADDRESS OF MONITOR
  18                *
  19 A010           *     ORG    $A010
  20                *
  21                *     (ACCA) := $00 FOR CLEARING MEMORY
  22                *
  23 A010 86 00     START LDAA   #$00       LOAD ACCA WITH CONSTANT
  24 A012 CE 0000         LDX    #$0000     INIT IX TO START OF MEMORY
  25 A015 A7 00     LOOP  STAA   0,X        STORE CONSTANT INTO M POINTED TO BY (IX)
  26 A017 08              INX                MOVE POINTER TO NEXT LOCATION
  27 A018 8C 8000         CPX    #$8000     LAST BYTE + 1 TO BE INITIALIZED
  28 A01B 26 F8           BNE    LOOP       GOTO STORE AGAIN
  29 A01D 7E E0D0         JMP    MNTR       RETURN TO MONITOR WHEN DONE
  30                *
  31                *     STARTING ADDRESS OF ROUTINE INTO $A048 AND $A049
  32                *
  33 A048           *     ORG    $A048
  34                *
  35 A048     A010        FDB    START
  36                      END

SYMBOL TABLE

MNTR   E0D0     START  A010     LOOP   A015

   0  ERRORS
```

Mr. Gabrielson:

As you observed, evaluating A(m,n) for increasing n is quite simple when m≤3. For example, $A(3,n) = 2^{n+3} - 3$. By definition A(4,0) = A(3,1) = A(2,5) = A(1,11) = A(0,12) = 13, and A(4,1) = A(3,13) = . . . = $2^{16} - 3 = 65563$ and so on. The calculations quickly become unrewarding. A(4,2) = A(3,65563) ≈ 2×10^{19737}, for example. But the function is a useful tool.

Best regards,
J. A. McLaughlin

Folly Fraction Farm
Clarksville, MD 21029

Dear Doctors:

As Mike Gabrielson points out in *DDJ* #31, you can learn something about computer languages from the way they code Ackermann's function. It's a kind of software touchstone that helps tell gold from brass. (Touch it to BASIC or FORTRAN and see what you get.)

Here are two more implementations, one in Tom Gibson's tiny-c (running on a Digital Group Z-80) and one in Xerox APL. Supporting functions are listed for general interest.

Both languages allow recursion and both handle Ackermann well, though APL's clumsy control structures look bad next to C's easy if's and else's. Of course recursion gobbles up the stack, and both examples end in errors. The tiny-c manual defines error 17 as "too many functions," which is about the same thing as "WS FULL."

Why did the big-system APL poop out after 3 ACKERMANN 3? I used a very small workspace to avoid wasting CPU's, and 3 ACKERMANN 4 ran out of room after calling itself 68 times.

Sincerely,
Les Hancock

305 Lexington Avenue
New York, NY 10016

```
     ∇ACKERMANN[□]∇
    ∇ R←M ACKERMANN N
[1]   →(0=M,N)/MZERO,NZERO
[2]   →0;R←(M-1)ACKERMANN M ACKERMANN N-1
[3]   NZERO:→0;R←(M-1)ACKERMANN 1
[4]   MZERO:R←N+1
    ∇
     ∇DEMO[□]∇
    ∇ MAT←DEMO
[1]   MAT←10 10⍴0
[2]   M←¯1
[3]   LOOP1:→0×⍳9<M←M+1
[4]   N←¯1
[5]   LOOP2:→LOOP1×⍳9<N←N+1
[6]   MAT[M+1;N+1]←M ACKERMANN N
[7]   →LOOP2
    ∇
     I22          ⍝ HOW MANY BYTES LEFT IN WORKSPACE?
5676
     DEMO
WS FULL
ACKERMANN[1]  →(0=M,N)/MZERO,NZERO
                    ^
     MAT
 1  2  3  4  5  6  7  8  9 10
 2  3  4  5  6  7  8  9 10 11
 3  5  7  9 11 13 15 17 19 21
 5 13 29 61  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
```

```
 0  0  0  0  0  0  0  0  0  0
     pI27  ⍝ HOW MANY ENTRIES IN STATE INDICATOR?
69

>.p 99

/* Ackermann's function: recursion in tiny-c.
/*      (Les Hancock, 1/12/79)
ackermann int m, n [
   if (m == 0) return n + 1
   if (n == 0) return ackermann ((m - 1), 1)
   return ackermann ((m - 1), ackermann (m, (n - 1)))
]
demo [
   int m, n
   m = -1
   pl ""
   while (1) [
      m = m + 1
      pl ""
      n = -1
      while (10 > (n = n + 1)) [
         pn formatted ackermann (m, n)
      ]
   ]
]
formatted int n [
   int spaces
   if (n < 10) spaces = 5
   else if (n < 100) spaces = 4
   else if (n < 1000) spaces = 3
   else if (n < 10000) spaces = 2
   else spaces = 1
   while (spaces = spaces - 1) ps " "
   return n
]
>.demo

    1   2   3   4   5   6   7   8   9  10
    2   3   4   5   6   7   8   9  10  11
    3   5   7   9  11  13

 3 --- err  17

DONE 26 14448   Note: rows of output correspond to values of
>>>             m from 0 through 2; columns are n = 0 through 9.
```

COMPUTERS AND HOME STEREOS

Dear *Doctor Dobbs*:

In response to Jim Warren's "Computer Control of Music Tapes for Your Home Stereo:"

I am much intrigued by your fantasy ("Tattoo!"), but a few items bother me. Firstest and mostest, el cheapo cassette decks, and even some SONY's, seem to pack the tape on the 'reels' in an extremely loose fashion. Next, if the directory is to be dynamic in size, as well as content, it should be the last record on the tape and moved as new items are added.

The application I had in mind was to use the computer controlled tape deck as a mass storage peripheral. There are two approaches to the organization of data sets on the tape. The first and most common is to use variable length records, one to a block. This provides the most efficient storage of formatted data, with two major disadvantages. The machine-written inter-block gaps must be read 'on the fly' during fast wind operations. The second bummer is that, if blocks are to be updated, they must be rewritten at the end of the tape, unless the block to be updated is at the end before the update occurs.

The other approach is to use fixed length records, delimiting them with markers that are not necessarily machine written (more on that in a bit). This still carries similar disadvantages to the above-mentioned method (and primitive micro DOS's). There may be much unusable space on the

media after a file has been updated, necessitating a packing operation which may be more than just a little bit clumsy. To avoid this, a bit map may be employed to distinguish between full and empty records. When a file is killed, it's catalog entry is deleted, so that it cannot be referenced, and it's positions in the bitmap are set to empty, so that it's space may be reused. To save a file, a catalog entry is made, pointing to the first record of the file. Space is found for subsequent records by a search of the bitmap, and each subsequent record is pointed to by the record before it. To illustrate:

```
WITH BITMAP      WITHOUT          OPERATION

AAAAA            AAAAA            file 'A' is saved
AAAAABBB         AAAAABBB         file 'B' is saved
AAAAABBBA        AAAAABBBAAAAAA   file a lengthened

space wasted
```

Now for some implementation ramblings. One side of a C-90 cassette is about 5062 inches long, space enough for 81000 unformatted bytes (assuming mits ACR format). In this format, a 512 byte record can be written in 17.07 seconds, on 32 inches of tape. If you were to make records 34 inches long, 148 of them would fit on one side. A piece of aluminum foil (found in stereo stores) can serve to mark the records to be sensed 'on the fly' by a pair of metallic feelers near the tape heads. Every time a bit of foil passed the feelers, an up/down counter would be incremented/decremented according to the direction of tape motion. When a Volume is mounted, it would be rewound to an optical BOT mark, the record counter set to zero, and the first record read to a special buffer area in memory. This record could contain such goodies as volume, name, a pointer to the directory, amount of free storage, etc.

To save a file, a check would be made to see if the file already exists on the volume. If and when it does not, a search of the bitmap would be made to find the first free record. The number of the first record will be added to the catalog entry, and from there the process of finding a record, filling it, and marking it as full can continue until the file is saved. This hardware would allow access by the sequential, direct, indexed, or any other access method you might be into.

Again, we seem to have two drawbacks. Unless you are a watchmaker or have similar skills, it is going to take much doing to mount the record feelers and the EOT/BOT sensors in a cassette deck. The other is that 74 kilobytes is not very much to have on-line, even if the hardware costs only $40. There are, however, bigger and better and reel-to-reel tape decks, too. An 1800 foot spool of tape will hold 317 kilobytes. A stereo or quad tape deck can double or quadruple the amount of storage with no increase in access time by switching the proper inputs and outputs of the drive and modem, either with reed relays, or, for those that are more into analog, transistor switching.

The electrically controlled actuation of the tape deck is not quite as bad as it sounds either. Many cheapo decks can be actuated by two linear motors (something like pygmy garage door openers). If you can braze, you will have no trouble here: all you need is a rotary motor, a threaded stud, and a nut to match. One word of caution is in order here: beware of motors and solenoids around the tape. These can magnetize the heads and/or erase the tape.

For those with more cash, there are a variety of solenoid-driven tape units available. Try also your favorite free speech radio/TV station.

I am currently working on a FORTRAN compiler/interpreter for the 8080 and would appreciate some feedback on the following points.

1. Is it really worthwhile to maintain the distinction between integer and real data types? My floating point is pretty fast, and I feel that it would unnecessarily complicate the compiler and possibly slow down the interpreter.

2. How about doing all I/O (except console) by indexed sequential method? This would yield both direct and sequential access, and it shouldn't be too hard to simulate many 'units' on one physical device.

3. How useful would a macro capability be?

Sincerely,
Charles De Craene

4741 Laurel Cyn. Suite 110
No. Hollywood, Calif. 91607

A LINE-FILE EDITOR

Dear Suzanne Rodriguez,

Enclosed is a line-file editor originally written as a splice in BASIC so the LOCAL terminal functions would be available in REMOTE. Some problems with a serial terminal occurred, however. For example, inserting a letter would cause the cursor to bounce to the end of the line and back. This was improved somewhat by monitoring the line size and only moving the cursor as needed. Later some of the functions were removed and written into a text editor. What is enclosed are some of those functions and enough of a program added on so that the program will function with a call from BASIC. BDC line number prompting and a few other features have been tacked on.

This letter was prepared from the program listing.

Yours truly,
L. Barker

Chicago, Illinois

```
OK
LOAD
@1T
A*VA
A*H9000
A*L65
A*RA


INIZ?N

10 INPUT "Allowed line length";X
20 POKE 11860,0 :POKE 11861,96
30 POKE 574,0 :POKE 575,112
40 X=USR(X)
50 FOR I=24576 TO X-1
60 PRINT CHR$(PEEK(I));
70 NEXT I
80 END
OK

.A

  10 0000        ; Terminal driver - File program for an ACT IV
  20 0000        ; to be called by a program in OSI 65-D BASIC
  30 0000        ;     Written by  L. Barker
  40 0000        ;
  50 0000        ; defined control codes are -
  60 0000        ;   #4  delete chr.
  70 0000        ;   #9  tab
  80 0000        ;   #13 return
  90 0000        ;   #19 insert chr.
 100 0000        ;   #23 delete line (@)
 110 0000        ;   #24 cursor forward
 120 0000        ;   #25 cursor backspace
 130 0000        ;   #30 erase to end of row
 140 0000        ;   #124 #13 returns to basic with x as the last loc
```

```
0000          ;
0000          ; 65-D Locations
0000          ;   11860  write to file pointer low
0000          ;   11861  pointer high
0000          ;   574  Call USR pointer low
0000          ;   575  pointer high
0000          ;    6    indirect pointer to pick up value X
0000          ;    175  low byte of 15 bit signed number X
0000          ;
0000            FILE=78 ; indirect pointer to file
0000           LENGTH=80 ; max line length plus one
0000           LINENU=81 ; bdc line number
0000             FLAG=82 ; insert mode - off is zero
0000             SIZE=83 ; line size entered
0000           TEMPRY=84 ; temp storage
0000          ;
0000            ACIA=64512
0000            PRINT=65035
0000          ;
7000           *=28672
7000          ;
7000 A900      BEGIN LDA #0     ;clear line number
7002 8551           STA LINENU
7004 8554           STA TEMPRY ;clear storage area
7006 AD542E         LDA 11860  ;write to file pointer, low
7009 854E           STA 78
700B AD552E         LDA 11861  ;file pointer, high
700E 854F           STA 79
7010 201F70         JSR GETLEN ;get line len. passed from basic
7013 A5AF           LDA 175    ;low byte of length passed by x
7015 F061           BEQ EXIT   ;if len=0 then invalid, exit
7017 C948           CMP #72
7019 B05D           BCS EXIT   ;basic's file input limit is 71
701B 8550           STA LENGTH ;store max allowed line length
701D D00D           BNE NUMBER ;set up line number
701F 6C0600  GETLEN JMP (6)
7022 98       AGAIN TYA      ;add y to file pointer
7023 18             CLC      ;and store pointer to next line
7024 654E           ADC 78
7026 854E           STA 78
7028 9002           BCC NUMBER
702A E64F           INC 79
702C A901    NUMBER LDA #1    ;bdc 0-99 line numbers
702E F8             SED       ;printed on screen for ref.
702F 18             CLC
7030 6551           ADC LINENU
7032 8551           STA LINENU
7034 D8             CLD
7035 A90D     CLEAR LDA #13   ;return
7037 200BFE         JSR PRINT
703A A91E           LDA #30
703C 200BFE         JSR PRINT
703F A450           LDY LENGTH
7041 A920           LDA #32   ;clear to end of line
7043 88       CLOOP DEY       ;clear file area with spaces
7044 914E           STA (78),Y
7046 D0FB           BNE CLOOP
7048 A551           LDA LINENU
704A 29F0           AND #240
704C 4A             LSR A
704D 4A             LSR A
704E 4A             LSR A
704F 4A             LSR A
7050 18             CLC
7051 6930           ADC #48
7053 200BFE         JSR PRINT  ;print hi byte of line number
7056 A551           LDA LINENU
7058 290F           AND #15
705A 18             CLC
705B 6930           ADC #48
705D 200BFE         JSR PRINT  ;print lo byte of line no.
7060 A92E           LDA #46    ;output period
7062 8D3F02         STA 575    ;restore usr call pointer, hi
7065 200BFE         JSR PRINT
7068 A920           LDA #32    ;output space
706A 8D3E02         STA 574    ;restore usr call pointer, lo
706D 200BFE         JSR PRINT  ;over doing restore,but works
7070 A900           LDA #0
7072 8552           STA FLAG   ;clear insert mode
7074 8553           STA SIZE   ;set size equal to zero
7076 F028           BEQ INPUT
7078 60       EXIT  RTS
7079 200BFE  RETURN JSR PRINT  ;chr$(13) entered
707C A453           LDY SIZE
707E C8             INY
707F 914E           STA (78),Y
7081 C8             INY
7082 A90A           LDA #10
7084 200BFE         JSR PRINT  ;line feed
7087 914E           STA (78),Y
7089 C8             INY
708A A554           LDA TEMPRY
708C C97C           CMP #124   ;check for end of input chr
708E D092           BNE AGAIN
7090 C003           CPY #3     ;check #124 in first position
7092 D08E           BNE AGAIN
7094 A44E           LDY 78     ;returning to basic
7096 A54F           LDA 79     ;pass back last position
7098 6C0800         JMP (8)    ;lo byte stored in y
709B A907     WARN  LDA #7     ;sound bell if line to long
709D 200BFE         JSR PRINT
70A0 AD00FC  INPUT  LDA ACIA
70A3 4A             LSR A
70A4 90FA           BCC INPUT
70A6 AD01FC         LDA ACIA+1
70A9 297F           AND #127  ;knock off parity bit
70AB C920           CMP #32   ;check if control code
70AD 901D           BCC CONTRL
70AF C450           CPY LENGTH
70B1 B0E8           BCS WARN  ;sound bell if line to long
70B3 C453           CPY SIZE  ;store max line len. so far
70B5 9002           BCC SHORT
70B7 8453           STY SIZE
70B9 8554    SHORT  STA TEMPRY
70BB A552           LDA FLAG  ;check if in insert mode
70BD D00A           BNE INSERT
70BF A554           LDA TEMPRY
70C1 914E           STA (78),Y
70C3 C8             INY
70C4 200BFE  OUTPUT JSR PRINT
70C7 D0D7           BNE INPUT
70C9 4C7371  INSERT JMP BRANCH
70CC 48      CONTRL PHA
70CD A552           LDA FLAG  ;clear insert mode and sound bell
70CF F009           BEQ CODES ;if flag is set
70D1 A900           LDA #0
70D3 8552           STA FLAG
70D5 A907           LDA #7
70D7 200BFE         JSR PRINT
70DA 68      CODES  PLA
70DB C90D           CMP #13
70DD F09A           BEQ RETURN
70DF C917           CMP #23
70E1 D003           BNE RESTC
70E3 4C3570         JMP CLEAR
70E6 C450    RESTC  CPY LENGTH
70E8 B035           BCS BCKWRD  ;codes that back up
70EA C918           CMP #24
70EC F0D5           BEQ OUTPUT-1
70EE C913           CMP #19
70F0 D006           BNE TAB
70F2 A907           LDA #7
70F4 8552           STA FLAG
70F6 D0CC    OTLINK BNE OUTPUT
70F8 C909       TAB CMP #9
70FA D023           BNE BCKWRD
70FC 98             TYA      ;check tab will be less than
70FD 29F8           AND #248  ;the max length allowed
70FF 18             CLC
7100 6908           ADC #8
7102 C550           CMP LENGTH
7104 B016           BCS INLINK  ;if not go back to input
7106 98             TYA
7107 2907           AND #7
7109 8554           STA TEMPRY
710B 38             SEC
710C A908           LDA #8
710E E554           SBC TEMPRY
7110 AA             TAX      ;set up positions to the
7111 A918           LDA #24 ;next tab stop and advance there
7113 200BFE  TBLOOP JSR PRINT
7116 C8             INY
7117 CA             DEX
7118 D0F9           BNE TBLOOP
711A 8453           STY SIZE
711C 4CA070  INLINK JMP INPUT
711F C904    BCKWRD CMP #4 ; delete chr
7121 F011           BEQ DELETE
7123 C91E           CMP #30
7125 F03B           BEQ ERASE
7127 C000           CPY #0
7129 F0F1           BEQ INLINK
712B C919           CMP #25
712D D0ED           BNE INLINK
712F 88             DEY
7130 A908           LDA #8
7132 D0C2           BNE OTLINK
7134 8454    DELETE STY TEMPRY
7136 A201           LDX #1
7138 C453    DELOOP CPY SIZE
713A B00D           BCS SKIP  ;if cursor at end of line, skip
713C C8             INY
713D B14E           LDA (78),Y
713F 88             DEY
7140 914E           STA (78),Y
7142 200BFE         JSR PRINT
7145 E8             INX
7146 C8             INY
7147 D0EF           BNE DELOOP
7149 E002      SKIP CPX #2
714B 9002           BCC SKIPPY  ;if at beginning size =0
714D C653           DEC SIZE
714F A920    SKIPPY LDA #32
7151 914E           STA (78),Y
7153 200BFE         JSR PRINT
7156 A454           LDY TEMPRY
7158 A908           LDA #8
715A 200BFE  BDLOOP JSR PRINT
715D CA             DEX
715E D0FA           BNE BDLOOP
7160 F0BA           BEQ INLINK
7162 8453     ERASE STY SIZE
7164 A920           LDA #32
7166 914E    ERLOOP STA (78),Y
7168 C8             INY
7169 C450           CPY LENGTH
716B 90F9           BCC ERLOOP
716D A453           LDY SIZE
716F A91E           LDA #30
7171 D083           BNE OTLINK
7173 C8      BRANCH INY
7174 A654           LDX TEMPRY
7176 8454           STY TEMPRY
7178 C450           CPY LENGTH
717A B00F           BCS DONE
717C A450           LDY LENGTH
717E 88      BRLOOP DEY
717F 88             DEY
7180 B14E           LDA (78),Y
7182 C8             INY
7183 914E           STA (78),Y
7185 C454           CPY TEMPRY
7187 F002           BEQ DONE
7189 B0F3           BCS BRLOOP
718B 88       DONE  DEY
718C 8A             TXA
718D 914E           STA (78),Y
718F E653           INC SIZE
```

```
2430 7191 A553           LDA SIZE
2440 7193 C550           CMP LENGTH
2450 7195 9002           BCC SHORTR
2460 7197 C653           DEC SIZE
2470 7199 B14E    SHORTR LDA (78),Y
2480 719B 20BBFE         JSR PRINT
2490 719E C8             INY
2500 719F C453           CPY SIZE
2510 71A1 F0F6           BEQ SHORTR
2520 71A3 90F4           BCC SHORTR
2530 71A5 A920           LDA #32
2540 71A7 20BBFE         JSR PRINT
2550 71AA A908           LDA #8
2560 71AC 20BBFE  B2LOOP JSR PRINT
2570 71AF 88             DEY
2580 71B0 C454           CPY TEMPRY
2590 71B2 B0F8           BCS B2LOOP
2600 71B4 C8             INY
2610 71B5 4CA070         JMP INPUT
2620 71B8                 END
```

```
START ADDRESS IS AT $013F.

H,A,M?A              H=HEX, A=ASCII, M=MONITOR (ADDRESS AT $014E).
ADDR?                REQUEST FOR LOWER AND UPPER SEARCH ADDRESS LIMITS.
Y<                   "Y" ENTERED FOR ADDRESS CHANGE, ANYTHING ELSE FOR NO CHANGE.
0100 01F5            ADDRESS ENTERED AFTER "<".
F: DDR               TO FIND "DDR"; END ASCII STRING WITH CARRIAGE RETURN.
010F                 "DDR" LOCATED AT THIS ADDRESS.
H,A,M?H              ROUTINE PROMPTS AGAIN AFTER ADDRESS PRINT-OUT.
ADDR?N               A LOOK FOR HEX WITH NO ADDRESS CHANGE.
F: FF0122<D          TO FIND "FF0122"; END HEX STRING WITH HEX CARRIAGE RETURN (0D).
0165 017A 01D9 01EA  "FF0122" IS RESIDENT AT THESE ADDRESSES.
                     NOW TO CHECK MIKBUG FOR "A600":
H,A,M?H ADDR?Y<E000 E1F7
F: A600<D
E07E E0BF E0C3 E1AF
H,A,M?M              MY MONITOR IS "MIKADOS" AND IT RESPONDS WITH A COLON.
:
```

```
.A2

;187000A90085518554AD542E854EAD552E854F201F70A5AFF061C90B03
;18701848B05D8550D00D6C0600981654E854E9002E64FA901F81809D0
;1870306551855 1D8A900D200BFEA91E200BFEA450A92088914ED0FB0BDA
;187048A55129F04A4A4A4A186930200BFEA551290F186930200BFE08E9
;187060A92E8D3F02200BFEA9208D3E02200BFEA90085528553F02809E5
;18707860200BFEA453C8914EC8A90A200BFE914EC8A554C97CD0920D12
;187090C003D08EA44EA54F6C0800A907200BFEAD00FC4A90FAAD010B97
;1870A8FC297FC920901DC450B0E8C4539002845385 54A552D00AA50CE5
;1870C054914EC8200BFED0D74C737148A552F009A9008552A907200BCB
;1870D80BFE68C90DF09AC917D0034C3570C450B035C918F0D5C9130D50
;1870F0D006A9078552D0CCC909D0239829F8186908C550B01698290C14
;18710807855438A908E554AAA918200BFEC8CAD0F984534CA070C90C7E
;18712004F011C91EF03BC000F0F1C919D0ED08A908D0C28454A201 0D46
;187138C453B00DC8B14E88914E200BFEE8C8D0EFE0029002C653A90D91
;18715020914E200BFEA454A908200BFECAD0FAF0BA8453A920914E0C90
;187168C8C45090F9A453A91ED083C8A6548454C450B00FA45088880DD8
;187180B14EC8914EC454F002B0F3888A914EE653A553C5509002C60DEB
;18719853B14E200BFEC8C453F0F690F4A920200BFEA908200BFE880D39
;0871B0C454B0F8C84CA070060D
```

SMALL MONITOR SEARCH SUBROUTINE FOR M6800

Dear *Dr. Dobb's*,

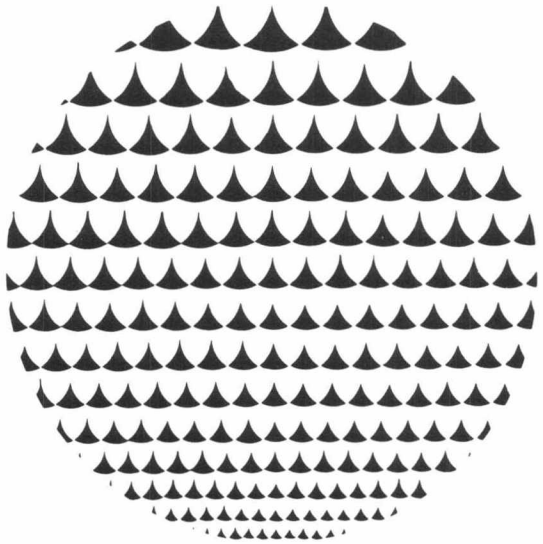
This is offered as a replacement for the single character search subroutines given in small monitors for the M6800. Short enough to relocate manually without much effort, it fits into 255 bytes, including 10 bytes for the search string. Following the Hex dump I'll use it for Hex and ASCII searches.

This routine seems to work fine and saves loading an editor which possibly will load over your search area; most editors do use much memory!

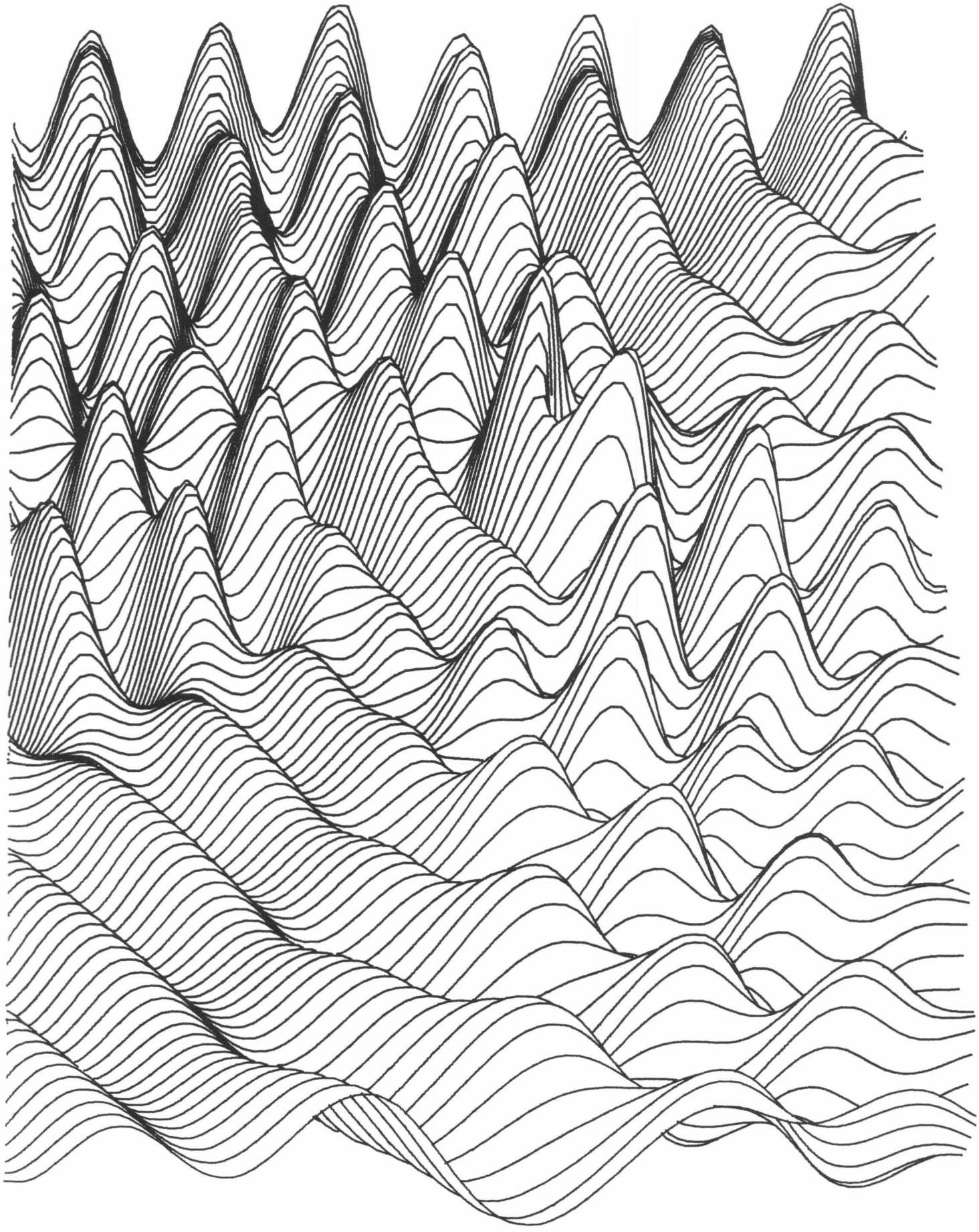
As indicated, I use "MIKADOS" (INPRO MICRO SYSTEMS, P.O. Box 3434, Thousand Oaks, CA 91359) and it's a jewel. I haven't seen anything on it in print since I bought it in early 1977, although they still sell it. For a small single pass assembler, disassembler, and mini-O/S it does the job. Admittedly, I can't write a PASCAL compiler with it, but *I* can't write a PASCAL compiler with *any* assembler!

Sincerely,
Neal Champion

602 Copper Basin Rd
Space 27
Prescott, AZ 86301



```
0100  00 0A 0D 04 0A 0D 48 2C 41 2C 4D 3F 04 20 41 44
0110  44 52 3F 04 3C 04 0A 0D 46 3A 20 04 00 00 40 00
0120  01 F6 40 00 3D 64 01 F6 00 4D 7E E0 47 7E E0 55
0130  CE 01 00 7E E0 7E 7E E0 C8 7E E1 AC 7E E1 D1 CE
0140  01 04 8D EF 8D F3 B7 01 29 81 4D 26 03 7E 26 14
0150  CE 01 0D 8D DE 8D E2 81 59 26 17 CE 01 14 8D D3
0160  8D C8 FF 01 1C FF 01 22 86 20 8D D0 8D BC 08 FF
0170  01 1E CE 01 16 8D BC FE 01 1C FF 01 22 CE 01 F6
018E  FF 01 26 FF 01 20 7F 01 28 09 08 B6 01 29 81 48
0190  27 04 8D A5 20 02 8D 95 A7 00 81 0D 26 EC 8D 90
01A0  FE 01 20 A6 00 FE 01 22 BC 01 1E 27 92 81 0D 26
01B0  11 CE 01 24 BD 01 36 FE 01 26 FF 01 20 7F 01 28
01C0  20 DE A1 00 27 18 FE 01 26 FF 01 20 7F 01 28 A6
01D0  00 FE 01 22 A1 00 27 06 08 FF 01 22 20 C2 7D 01
01E0  28 26 06 7C 01 28 FF 01 24 08 FF 01 22 FE 01 20
01F0  08 FF 01 20 20 AA
```



© David Dameron 1979

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 34 | April 1979 | Volume 4, Issue 4

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

Falconer Floating Point Arithmetic, Part II 16
Charles B. Falconer

WP-68 Word Processor 4
Gary D. Gaugler

Bringing Up UCSD Pascal Without CP/M 26
Mike Gabrielson

A Conversation with Bob Albrecht, Dennis Allison and Rick Bakalinsky 32
Suzanne Rodriguez

Increase Your Poly 88 Pleasure 40
David L. Johnson

Conversion: 8-Bit Memories to 16-Bits 30
George Lyons

Product Evaluation: PERCOM LFD-400 41
Douglas K. Beck

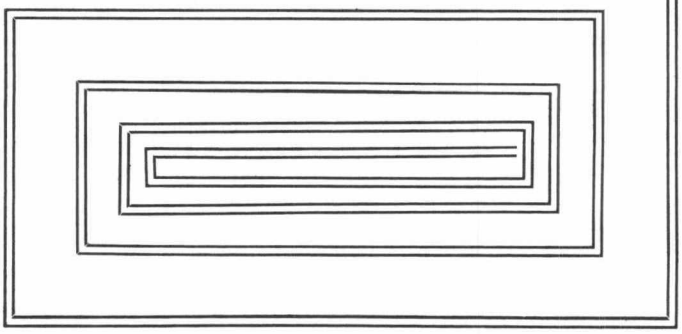
Unspecified 8085 Op Codes Enhance Programming 28
Wolfgang Dehnhardt and Villy M. Sorensen

OSI Basic for the KIM-1 37
Jonathan M. Prigot

A Disk for Frits 35
Dale Puckett

WP-68 Word Processor

BY GARY D. GAUGLER
5675 Spyglass Lane
Citrus Heights, CA 95610



Normally we typeset articles, but we reproduce this here the way we received it so you can get an idea of what the program does. – SMR

Note: This is the input which produced the text describing the WP-68 Word Processor.

```
.W
&1:WP68.1:WP68D4.DOC
*CHANGE FORM - 1=SKIP PAGE  0=PRINT PAGE

                    WP-68 WORD PROCESSOR

     THE WP68 WORD PROCESSOR IS A POWERFUL ADDITION TO THE
SMOKE SIGNAL BBS-68 DISK SYSTEM. WP68 PROCESSES FILES WHICH
ARE READ FROM THE DISK. THE FILES ARE CREATED BY USING THE
SSB SUPER EDITOR OR OTHER MEANS TO PRODUCE A FILE CONTAINING
TEXT AND SPECIAL COMMANDS WHICH ARE PROCESSED BY THE WORD
PROCESSOR.

          THE COMMANDS TO WP68 ARE SIGNIFIED BY A
     PERIOD (.) IN THE FIRST COLUMN, FOLLOWED BY A
     LETTER WHICH REPRESENTS THE COMMAND TO THE WORD
     PROCESSOR. THERE ARE 24 SEPARATE COMMANDS WHICH
     THE WORD PROCESSOR RECOGNIZES. THESE COMMANDS
     ALLOW THE TEXT TO BE RIGHT AND LEFT JUSTIFIED, TO
     BE PARAGRAPHED, PAGED, INDENTED, TARRED, MARGINED,
     MULTIPLE SPACED, AND OTHERS. COMMANDS ARE
     AVAILABLE TO ROUTE PROCESSED OUTPUT TO THE CONTROL
     TERMINAL OR TO A SEPARATE PRINTING DEVICE.
```

WP68 DESCRIPTION

ANY LENGTH FILE CAN BE WORD PROCESSED USING THE VIRTUAL MEMORY FEATURE OF THE SSB BFD68 DISK. AS THE BUFFER IS PROCESSED. IF MORE DATA IS LEFT ON THE DISK, THE REMAINING SOURCE WILL BE PULLED OFF UNTIL AN END- OF-FILE IS READ.

THE WORD PROCESSOR IS MOST EFFECTIVE WHEN USED WITH A VIDEO TERMINAL SUCH AS THE SOROC IQ120 AND A SEPARATE PRINTING DEVICE. THE TERMINAL AND PRINTER SHOULD BE ABLE TO HANDLE 64 CHARACTERS PER LINE.

COMMAND SUMMARY

THE FOLLOWING IS A QUICK SUMMARY OF THE WP68 COMMANDS:

COMMAND DESCRIPTIONS

.B CAUSE FOLLOWING INFORMATON TO BE BLOCKED

.B A SECOND ONE CAUSES THE BLOCK TO BE TURNED OFF

. THIS, BY ITSELF, CAUSES THE BLOCKING MODE TO BREAK PARAGRAPH, BUT WILL NOT RESET THE BLOCK MODE CONDITION.

.C A VALUE OF ZERO CAUSES THE NEXT LINE TO BE CENTERED W THE BLOCK LENGTH.

.CNN CAUSES THE LINE COUNT PER PAGE TO BE SET TO THE VALUE (DEFAULT COUNT IS 35 LINES PER PAGE).

.E CAUSES THE BUFFER TO BE PRINTED AS IS: THUS NOT INTERPRETING THE COMMANDS. THIS IS USEFUL FOR DOCUMENTING WP68 DATA FILES.

.H CAUSES THE DATA FOLLOWING THE H TO ENTER INTO THE HEADER BUFFER AND WILL BE PRINTED AT THE TOP OF EACH PAGE. DO NOT ENTER A SPACE BETWEEN THE H AND THE HEADER CHARACTERS. THE HEADER CAN BE TURNED OFF BY ENTERING .H(SPACE)(CR). THE HEADER MAY BE CHANGED WITHIN THE EDIT FILE AND WILL REMAIN THE SAME AS THE LAST ENCOUNTERED H COMMAND UNTIL A NEW ONE IS REACHED AS THE BUFFER IS PRINTED.

.INN CAUSES THE NEXT LINE TO BE INDENTED NN SPACES.

.-INN CAUSES A LINE TO START NN SPACES BEFORE THE POSITION SET BY THE LEFT MARGIN COMMAND. THIS WILL NOT WORK IF THE VALUE OF THE MINUS INDENT IS LARGER THAN THE VALUE OF THE MARGIN COMMAND.

.LNN ESTABLISH HOW MANY CHARACTER POSITIONS LONG A BLOCKED LINE WILL BE. IT DOES NOT AFFECT LINES OTHER THAN THOSE WHICH ARE BLOCKED OR CENTERED. (DEFAULT IS 64 POSITIONS)

.M CAUSES A LEFT MARGIN OF NN SPACES. BEWARE OF USING THIS WITH THE TAB MODE.

.N0 THIS COMMAND TURNS OFF ALL HALTS INCLUDING THE HALT FOR A PAGE CHANGE. IT IS USEFUL TO OBTAIN A PRINTOUT ON CONTINUOUS PAPER FOR EXAMINATION OF THE TEXT WITHOUT HAVING TO INTERACT WITH THE SYSTEM DURING PRINTING.

.N1 THIS COMMAND RESTORES THE HALT FEATURES.

.NP THIS COMMAND TURNS OFF THE PAGE NUMBERING FEATURE. IF THE HALTS ARE ENABLED, THE PRINTER WILL HALT AT THE PROPER PLACE FOR THE USER TO MANUALLY INSERT THE PAGE NUMBER IF SO DESIRED. THIS DOES NOT AFFECT THE PAGE COUNTER WHICH IS STILL INCREMENTING FOR EACH PAGE.

.NS THIS IS THE COMMAND USED TO RESTORE THE PAGE NUMBER PRINTING FEATURE TO THE DEFAULT (ON) CONDITION.

.P CAUSES THE CURRENT PAGE TO BE ENDED (PAGE NUMBER PRINTED AT BOTTOM) AND A NEW PAGE STARTED WITH THE NEXT PAGE NUMBER.

.PNN CAUSES THE PAGE COUNTER TO BE SET TO NN. THIS IS USEFUL WHEN PROCESSING DISK FILES SEQUENTIALLY OR WHEN ONE JOB IS TOO LARGE TO FIT ON ONE DISK.

.PSNN CAUSES THE PAGE COUNTER TO BE INCREMENTED BY NN. THIS IS VERY USEFUL WHEN IT IS DESIRED TO INSERT PHOTOGRAPHS, DRAWINGS, ETC. WITHIN THE TEXT AND YOU WISH TO RESERVE A PAGE NUMBER FOR THESE ITEMS.

.R CAUSES A JSR TO THE PRINTER INITIALIZATION ROUTINE.

.S0 RESET ALL TAB STOPS; I.E. NO TABS SET.

.SNN SETS A TAB STOP AT NN; THIS STOP IS CALULATED FROM POSITION 1, EVEN IF THE MARGIN MODE IS ACTIVE.

.T THIS CAUSES THE OUTPUT TO BE TABBED TO THE NEXT TAB STOP IN THE TAB TABLE.

.U? ? CAN BE EQUAL TO ANY CHARACTER WHICH IS TO BE PRINTED AS A SPACE BUT IS TREATED AS A VALID CHARACTER BY THE BLOCKING MODE. THIS IS USEFUL FOR LEAVING SPACES FOR INSERTION OF SOME CHARACTER WHICH MAY NOT BE AVAILABLE ON YUR PRINTER. IT KEEPS THE BLOCKING MODE FROM BREAKING UP A STRING OF CHARACTERS (DEFAULT IS THE POUND SIGN).

.W? ? CAN BE EQUAL TO ANY CHARACTER WHICH IS TO BE INTERPRETED AS THE HALT CHARACTER. HALT CHARACTERS ARE USED IN PAIRS LIKE QUOTATION MARKS. THE TEXT WHICH OCCURS BETWEEN THE HALT CHARACTERS IS PRINTED ON THE TERMINAL (MIKBUG) DEVICE WHEN THE HALT OCCURS. THIS IS USEFUL FOR INSERTNG OPERATOR MESSAGES SUCH AS 'CHANGE TO TYPE BALL PRESTIGE ELITE.' IF TWO OR MORE HALT STRINGS OCCUR WITHIN A SINGLE LINE OF TYPE, THEN THE MESSAGE WHICH APPEARS WITHIN THE LAST STRING WILL BE PRINTED FOR EACH OCCURENCE ON THAT LINE. TWO HALT CHARACTERS ARE REQUIRED FOR EACH HALT AND IF ADJACENT TO EACH OTHER, WILL NOT PRINT A MESSAGE (EXCEPT IN THE CASE OF MULTIPLE HALTS PER LINE). THE DEFAULT CHARACTER IS THE AND SIGN (AMPERSAND).

.ZN THIS IS THE PRINTER CONTROL FEATURE. IF N=0, THEN ALL OUTPUT WILL BE TRANSMITTED TO THE MIKBUG TERMINAL DEVICE.
IF N=1, THEN THE PROCESSED TEXT WILL BE OUTPUT TO THE PRINTER AND MESSAGES (HALT AND PAGE CHANGE) WILL BE TRANSMITTED TO THE TERMINAL (MIKBUG) DEVICE.
IF N=2, THEN A FORM FEED CHARACTER (CONTROL L) WILL BE TRANSMITTED AT THE END OF EACH PAGE. THIS IS USEFUL WHEN A LINE PRINTER IS EMPLOYED WHICH HAS A FORMS CONTROL CAPABILITY.
IF N=3, THEN THE FORM FEED CHARCTER IS SUPPRESSED.

HALT THIS FUNCTION IS THE CONTROL C HALT AND WAIT FOR INSTRUCTIONS. IF THE NEXT CHARACTER AFTER THE CONTROL C IS A C, THEN THE OUTPUT WILL RESUME WHERE IT LEFT OFF. ANY OTHER CHARACTER WILL INITIATE AN ABORT OF THE WP68 PROGRAM.

BACKSPACE A BACKSPACE MAY BE ADJOINED TO ANY WORD FOR THE PURPOSE OF UNDERLINING OR OVERSTRICKING. THE BACKSPACE CHARACTER IS SET TO BE A CONTROL H. CONSULT THE INSTRUCTIONS FOR THE LOCATION OF THE BACKSPACE CHARACTER IN ORDER TO CHANGE IT TO ANOTHER CHARACTER. DO NOT LEAVE A SPACE BETWEEN WORDS AND THE BACKSPACE (FILL WITH THE UNPRINTABLE CHARACTER IF NECESSARY) TO ENSURE THAT THE BLOCKING MODE WILL PLACE THE BACKSPACE AND THE WORD TO BE UNDERLINED ON THE SAME LINE.

```
                                NOTES

        IF PAGE NUMBER PRINTING IS TURNED OFF AND THE HALT
      COMMAND IS TURNED ON (DEFAULT IS ON), THEN WHEN THE OUTPUT
      DEVICE HALTS AT THE PLACE WHERE A PAGE NUMBER WOULD BE
      PRINTED, AN EOT IS TRANSMITTED TO THE OUTPUT PRINTER. THIS
      IS A SIGNAL WHICH IS USED BY MANY IBM SELECTRIC TERMINALS TO
      TRANSFER THE TERMINAL TO A SEND MODE (THIS ALLOWS MANULAL
      TYPING OF THE PAGE NUMBER, ETC. IF DESIRED).

        THE CHARACTER WHICH IS TRANSMITTED MAY BE CHANGED BY

COMMAND DESCRIPTIONS

      CHANGING THE MEMORY LOCATIONS CONTAINING THE CHARACTER.

        THIS SAME CHARACTER IS TRANSMITTED EACH TIME A HALT IS
      INSERTED. THIS ALLOWS THE OPERATOR TO INSERT A FEW (AS MANY
      AS DESIRED) CHARACTERS MANUALLY. A PRINTER INITIALIZE IS
      THEN PERFORMED AFTER EACH HALT (PAGE NUMBER HALTS, ETC.) TO
      PLACE THE PRINTER/TERMINAL IN THE RECEIVE MODE.

        BEWARE OF MANUAL TYPING ON LINES WHICH HAVE TABS, ARE
      CENTERED, OR ARE WITHEN A BLOCKED PARAGRAPH! THE SPACING
      WILL DEFINITELY SUFFER UNDER THESE CONDITIONS UNLESS YOU
      'RESERVE' A SPECIFIED NUMBER OF SPACES USING THE UNPRINTABLE
      CHARACTER (IN WHIH CASE YOU MUST TYPE THAT NUMBER OF
      CHARACTERS, OR LESS, THEN BACKSPACE MANUALLY BEFORE ALLOWING
      OUTPUT TO RESUME).
```

INPUT EXAMPLE

```
WP68D4


.N0,,S0,,S10,,C55,,F2,,S20,,L60,,U#
.HWP68 DESCRIPTION
.C,WP-68 WORD PROCESSOR
.F0,,F0
.B
.I5
THE WP68 WORD PROCESSOR IS A POWERFUL ADDITION TO THE
SMOKE SIGNAL BFD-68 DISK SYSTEM.  WP68 PROCESSES FILES
WHICH ARE READ FROM THE DISK.  THE FILES ARE CREATED BY
USING THE SSB SUPER EDITOR OR OTHER MEANS TO PRODUCE A
FILE CONTAINING TEXT AND SPECIAL COMMANDS WHICH ARE
PROCESSED BY THE WORD PROCESSOR.
.
.M5,,L50,,F3,,F0,,I5
THE COMMANDS TO WP68 ARE SIGNIFIED BY A PERIOD (.) IN
THE FIRST COLUMN, FOLLOWED BY A LETTER WHICH REPRESENTS
THE COMMAND TO THE WORD PROCESSOR.  THERE ARE 24 SEPARATE
COMMANDS WHICH THE WORD PROCESSOR RECOGNIZES.  THESE
COMMANDS ALLOW THE TEXT TO BE RIGHT AND LEFT JUSTIFIED,
TO BE PARAGRAPHED, PAGED, INDENTED, TABBED, MARGINED,
MULTIPLE SPACED, AND OTHERS.  COMMANDS ARE AVAILABLE TO
ROUTE PROCESSED OUTPUT TO THE CONTROL TERMINAL OR TO
A SEPARATE PRINTING DEVICE.
.B
.C
SPECIAL NOTE
.M0,,L60,,F0,,B,,I5
ANY LENGTH FILE CAN BE WORD PROCESSED USING THE
VIRTUAL MEMORY FEATURE OF THE SSB BFD68 DISK.  AS THE
BUFFER IS PROCESSED, IF MORE DATA IS LEFT ON THE DISK,
THE REMAINING SOURCE WILL BE PULLED OFF UNTIL AN END-
OF-FILE IS READ.
.
.F0,,I5
THE WORD PROCESSOR IS MOST EFFECTIVE  WHEN USED
WITH A VIDEO TERMINAL SUCH AS THE SOROC IQ120 AND A
SEPARATE PRINTING DEVICE.  THE TERMINAL AND PRINTER
SHOULD BE ABLE TO HANDLE 64 CHARACTERS PER LINE.
.B
.HCOMMAND SUMMARY
.P
.I5
THE FOLLOWING IS A QUICK SUMMARY OF THE WP68 COMMANDS:
.F0
.F2
.T,#.B.T,BLOCK ON
.T,#.B.T,2ND TIME TURNS BLOCK OFF
.T,#..T,BLOCK BREAK
.T,#.C.T,CENTER
.T,#.CNN.T,LINES PER PAGE
.T,#.E.T,PRINT AS-IS
.T,#.H.T,HEADER
.T,#.INN.T,INDENT
.T,#.-INN.T,MINUS INDENT
```

listing

```
 1                        NAM    WP68
 2                *
 3                ******************************************************
 4                ******************************************************
 5                ****                                              ****
 6                ****                 WP-68                        ****
 7                ****                                              ****
 8                ****  WORD (TEXT) PROCESSOR FOR THE 6800 AND       ****
 9                ****  SMOKE SIGNAL BFD-68 DISC.                   ****
10                ****                                              ****
11                ****                                              ****
12                ****  (C) COPYRIGHT 1978 BY GARY D. GAUGLER       ****
13                ****  ALL RIGHTS RESERVED.  NO REPRODUCTION       ****
14                ****  FOR COMMERCIAL PURPOSES MAY BE MADE         ****
15                ****  WITHOUT WRITTEN PERMISSION FROM THE         ****
16                ****  AUTHOR.                                     ****
17                ****                                              ****
18                ******************************************************
19                ******************************************************
20                ************** VER. 6.5  REL. 1.0 ******************
21                ******************************************************

23 8009           TERMDA EQU    $8009
24 72A3           ZWARMS EQU    $72A3
25 7294           ZGCHAR EQU    $7294
26 7291           ZFLSPC EQU    $7291
```

```
 27 72A9           ZTYPDE EQU   $72A9
 28 7786           DFM    EQU   $7786
 29 7783           CDFM   EQU   $7783
 30 E0C8           OT4HS  EQU   $E0C8
 31 0000           XFC    EQU   0
 32 0004           Q04R   EQU   4
 33 0005           QRED   EQU   5
 34 0006           QSRC   EQU   6
 35 E0CA           OT2HS  EQU   $E0CA
 36 E07E           PDATA  EQU   $E07E
 37 E1AC           INEEE  EQU   $E1AC
 38 E1D1           OUTEEF EQU   $E1D1

 40                       OPT   NOG,NOP

 42 0100                  ORG   $100

 44 0100 7E 01BA START    JMP   START3
 45 0103 7E F803 PRINTR   JMP   $F803      TO PRINTER ROUTINE
 46 0106 7E F800 PRNINZ   JMP   $F800      TO PRINTR INIT ROUTINE
 47 0109 7E E0CA OUT2HS   JMP   OT2HS
 48 010C 7E E0C8 OUT4HS   JMP   OT4HS
 49 010F 7E E07E PDATA1   JMP   PDATA
 50 0112 7E E1AC INCH     JMP   INEEE
 51 0115 7E E1D1 OUTCH    JMP   OUTEEE

 53 0118 23      NOPRT    FCB   '#         THIS REPRESENTS THE UNPRINTABLE CHARACTER
 54 0119 26      HALT     FCB   '&         DEFAULT HALT CHAR
 55 011A 00 00   HPTRLO   FDB   0
 56 011C 00 00   HPTRHI   FDB   0
 57 011E 08      BKSPC    FCB   'H-$40     DEFINE BACKSPACE CHAR

 59 011F FF 09D5 WAIT     STX   LDBX
 60 0122 FE 011A          LDX   HPTRLO
 61 0125 BC 011C WAIT1    CPX   HPTRHI
 62 0128 27 18            BEQ   WAIT2
 63 012A A6 00            LDA A 0,X
 64 012C BD E7            BSR   OUTCH
 65 012E A6 00            LDA A 0,X
 66 0130 81 0D            CMP A #$D
 67 0132 26 0B            BNE   WA1A
 68 0134 FF 011A          STX   HPTRLO
 69 0137 CE 03D8          LDX   #CRLF+1
 70 013A BD D3            BSR   PDATA1
 71 013C FE 011A          LDX   HPTRLO
 72 013F 08      WA1A     INX
 73 0140 20 E3            BRA   WAIT1
 74 0142 FE 09D5 WAIT2    LDX   LDBX
 75 0145 7D 0A06          TST   STPFLG
 76 0148 26 12            BNE   PRNT2       RTS IF HALT OFF
 77 014A 86 04            LDA A #4          EOT TO IBM
 78 014C BD 0F            BSR   PRNTST
 79 014E BD C2            BSR   INCH
 80 0150 36               PSH A
 81 0151 37               PSH B
 82 0152 FF 09D5          STX   LDBX
 83 0155 BD AF            BSR   PRNINZ
 84 0157 FE 09D5          LDX   LDBX
 85 015A 33               PUL B
 86 015B 32               PUL A
 87 015C 39      PRNT2    RTS

 89 015D B1 0119 PRNTST   CMP A HALT
 90 0160 27 BD            BEQ   WAIT
 91 0162 81 0D            CMP A #$D
 92 0164 27 1D            BEQ   OUTFF
 93 0166 81 0A            CMP A #$A
 94 0168 27 F2            BEQ   PRNT2
 95 016A 7D 09E4 PRNT1    TST   SKIP
 96 016D 26 08            BNE   RBT2
 97 016F 7D 0967          TST   PNTON       IS PRINTER ON?
 98 0172 26 0A            BNE   RBT1        YES
 99 0174 BD 0115          JSR   OUTCH       NO GO TO MIKBUG
100 0177 BD 094D RBT2     JSR   BREAK
101 017A 2B E0            BMI   PRNT2
102 017C 20 53            BRA   QEXIT

104 017E BD 0103 RBT1     JSR   PRINTR
105 0181 20 F4            BRA   RBT2

107 0183 37      OUTFF    PSH B
108 0184 F6 0966          LDA B FMFEED
109 0187 5A               DEC B
110 0188 2B 1B            BMI   ERR
111 018A 27 15            BEQ   RON         FMFEED=1=RTN TO REG PRINT1

113 018C BD 08DF          JSR   OCRLF       OUTPUT 1ST CRLF

115 018F 7A 09D8 OUTFF1   DEC   CRCT
116 0192 26 05            BNE   OUTFF2
117 0194 33               PUL B
118 0195 7C 09D8          INC   CRCT
119 0198 39               RTS

121 0199 BD 08DF OUTFF2   JSR   OCRLF
122 019C 5A               DEC B
123 019D 26 F0            BNE   OUTFF1
124 019F 33               PUL B
125 01A0 39               RTS

127 01A1 33      RON      PUL B
128 01A2 7E 08DF          JMP   OCRLF

130 01A5 7E 0742 ERR      JMP   CONTRL

132 01A8 CE 0B3E SETBUF   LDX   #BUFBEG     BEGINNING OF TEXT BUFFER
133 01AB 86 FF            LDA A #$FF
134 01AD A7 00            STA A 0,X
135 01AF 08               INX
136 01B0 FF 09DF          STX   BUFPNT
137 01B3 CE 0A09          LDX   #BLOCK
138 01B6 FF 09DA          STX   BLKLNE
139 01B9 39               RTS

141                *
142                *  DISK PROCESSOR
143                *
144 01BA 86 01    START3  LDA A #1          INIT PAGE COUNT
145 01BC B7 0A93          STA A PGCT
146 01BF BD 7294          JSR   ZGCHAR      SEE IF ARG FOLLOWS
147 01C2 81 0D            CMP A #$D
148 01C4 27 0B            BEQ   QEXIT
149 01C6 CE 0A94 FILSPC   LDX   #FCB
150 01C9 BD 7291          JSR   ZFLSPC      GET SOURCE FILE NAME
151 01CC 24 20            BCC   CONT
152 01CE BD 72A9 EREXIT   JSR   ZTYPDE      ERROR
153 01D1 BD 7783 QEXIT    JSR   CDFM        CLOSE FILE
154 01D4 7E 7283 WARMS    JMP   ZWARMS

156 01D7 BD 7294 MORE     JSR   ZGCHAR      SEE IF ANOTHER FILE TO GO
157 01DA 81 0D            CMP A #$D         C/R?
158 01DC 27 F6            BEQ   WARMS       YEP: GET OUT
159 01DE 20 E6            BRA   FILSPC      NO: GET NEW FILE INFO AND GO TO IT

161 01E0 CE 0A94 CLOSE    LDX   #FCB
162 01E3 86 06            LDA A #QSRC
163 01E5 A7 00            STA A XFC,X
164 01E7 BD 7786          JSR   DFM
165 01EA 27 EB            BEQ   MORE
166 01EC 20 E0            BRA   EREXIT

168 01EE          CONT    EQU   *
169               * CLEAR WORK AREA
170               *
171 01EE CE 096A          LDX   #LENGTH+1
172 01F1 6F 00    ST1     CLR   0,X
173 01F3 08               INX
174 01F4 8C 0A93          CPX   #PGCT
175 01F7 26 F8            BNE   ST1
176 01F9 86 26            LDA A #'&         SET DEFAULT FOR HALT
177 01FB B7 0119          STA A HALT
178 01FE 86 23            LDA A #35
179 0200 B7 0968          STA A CRCT1
180 0203 B7 09D8          STA A CRCT
181 0206 86 01            LDA A #1
182 0208 B7 0966          STA A FMFEED
183 020B 7F 0967          CLR   PNTON       DEFAULT FOR NO PRINTER
184 020E 86 40            LDA A #64
185 0210 B7 0969          STA A LENGTH
186 0213 8D 93            BSR   SETBUF
187 0215 7F 0A8A          CLR   ENDFLG
188 0218 CE 0B3E          LDX   #BUFBEG
189 021B 08               INX
190 021C FF 0A91          STX   SRCBEG
191 021F CE 3DFF          LDX   #$3DFF
192 0222 09               DEX
```

```
193 0223 FF 0A8F         STX    BUFEND
194 0226 CE 0A94         LDX    #FCB
195 0229 86 04           LDA A  #Q04R      OPEN FOR READ
196 022B A7 00           STA A  XFC,X      DFM FUNCTION CODE
197 022D BD 7786         JSR    DFM
198 0230 26 9C           BNE    EREXIT
199 0232 86 05           LDA A  #QRED      READ FUNCTION
200 0234 A7 00           STA A  XFC,X
201              * FILE OPEN NOW
202 0236 8D 0F   LOAD    BSR    FILBUF
203 0238 BD 0366         JSR    FRMFD1
204 023B 8D 55           BSR    LOOP
205 023D 7D 0A8A         TST    ENDFLG
206 0240 26 9E           BNE    CLOSE
207 0242 BD 0148         JSR    SETBUF
208 0245 20 FF           BRA    LOAD

210 0247 7F 0A89 FILBUF  CLR    FLAG
211 024A FE 0A91         LDX    SRCBEG
212 024D BD 29   FILB1   BSR    READ
213 024F BC 0A8F         CPX    BUFEND
214 0252 26 03           BNE    FILB2
215 0254 7C 0A89         INC    FLAG
216 0257 81 0D   FILB2   CMP A  #$D
217 0259 27 12           BEQ    FILB5
218 025B 81 1A           CMP A  #$1A
219 025D 26 09           BNE    FILB3
220 025F B7 0A8A         STA A  ENDFLG
221 0262 86 FF           LDA A  #$FF
222 0264 A7 00           STA A  0,X
223 0266 20 0C           BRA    FILB6

225 0268 A7 00   FILB3   STA A  0,X
226 026A 08      FILB4   INX
227 026B 20 E0           BRA    FILB1
228 026D A7 00   FILB5   STA A  0,X
229 026F 7D 0A89         TST    FLAG
230 0272 27 F6           BEQ    FILB4
231 0274 FF 0A8D FILB6   STX    SRCEND
232 0277 39              RTS

234 0278 FF 0A8B READ    STX    XTEM
235 027B CE 0A94         LDX    #FCB
236 027E BD 7786         JSR    DFM
237 0281 27 0B           BEQ    READ1
238 0283 A6 01           LDA A  1,X
239 0285 81 06           CMP A  #6
240 0287 27 03           BEQ    ENDFIL
241 0289 7E 01CE         JMP    EREXIT
242 028C 86 1A   ENDFIL  LDA A  #$1A
243 028E FE 0A8B READ1   LDX    XTEM
244 0291 39              RTS
246              *
247              *  BEGIN WORD PROCESSOR
248              *
249 0292         LOOP    EQU    *
250 0292 FE 09DF         LDX    BUFPNT
251 0295 09              DEX
252 0296 E6 00           LDA B  0,X
253 0298 08              INX
254 0299 A6 00           LDA A  0,X
255 029B 08              INX
256 029C FF 09DF         STX    BUFPNT
257 029F 09              DEX
258 02A0 BC 0A8D         CPX    SRCEND
259 02A3 27 04           BEQ    JEOB
260 02A5 81 FF           CMP A  #$FF       END OF BUFFER?
261 02A7 26 03           BNE    LP4
262 02A9 7E 0341 JEOB    JMP    EOB        END OF BUFFER: STOP

264 02AC 7D 0A04 LP4     TST    ASIS
265 02AF 26 37           BNE    P3
266              * CHECK FOR SPECIAL FORMAT COMMANDS

268 02B1 81 5C           CMP A  #$5C       BACKSLASH?
269 02B3 27 18           BEQ    LP36
270 02B5 81 2F           CMP A  #'.        WP68 COMMAND?
271 02B7 26 17           BNE    LP1
272 02B9 C1 2C           CMP B  #',        MORE COMMANDS?
273 02BB 27 10           BEQ    LP36
274 02BD C1 0D           CMP B  #$D
275 02BF 27 0C           BEQ    LP36
276 02C1 C1 FF           CMP B  #$FF
277 02C3 27 08           BEQ    LP36

279              *
280              * CHECK FOR TABS IN A DATA STRING
281              *
282 02C5 E6 01           LDA B  1,X
283 02C7 C1 2C           CMP B  #',        LOOK FOR ,T, SEQUENCE
284 02C9 27 02           BEQ    LP36
285 02CB 20 03           BRA    LP1
286 02CD 7E 0620 LP36    JMP    SPCTL

288 02D0 7D 09D9 LP1     TST    BLKFLG     BLOCK MODE?
289 02D3 27 03           BEQ    LP35
290 02D5 7E 03FD         JMP    BLKRT2

292 02D8 81 3C   LP35    CMP A  #$3C       LEFT CARAT? <
293 02DA 26 03           BNE    LP2
294 02DC 7E 03A2         JMP    TNBLK

296 02DF 81 5B   LP2     CMP A  #$5B       LEFT BRACKET? [
297 02E1 27 14           BEQ    TNCNT
298 02E3 7D 09E7         TST    CENTER
299 02E6 26 1C           BNE    CNTRTN
300 02E8 BD 0451 P3      JSR    LDBUF
301 02EB 81 0D           CMP A  #$D
302 02ED 26 A3           BNE    LOOP
303 02EF 7D 0A05         TST    HFLAG
304 02F2 26 9E           BNE    LOOP
305 02F4 7E 0501         JMP    OUTOUT

307 02F7 09      TNCNT   DEX
308 02F8 7F 09E8 TNCNT1  CLR    CNTRCT
309 02FB FF 09E9         STX    CNTBUF
310 02FE 7C 09E7         INC    CENTER
311 0301 7E 0292         JMP    LOOP

313 0304 7E 055D CNTRTN  JMP    RTNCNT
314 0307 37      OUTPUT  PSH B
315 0308 81 0118         CMP A  NOPRT      UNPRINTABLE CHARACTER?
316 030B 27 04           BEQ    P32        YES
317 030D 81 FF           CMP A  #$FF
318 030F 26 02           BNE    P8
319 0311         P32     EQU    *
320 0311 86 20           LDA A  #$20
321 0313         P8      EQU    *
322 0313         P1      EQU    *
323 0313 BD 0335         JSR    LOUT
324 0316 26 18           BNE    P9
325 0318 86 0A           LDA A  #$A
326 031A BD 015D         JSR    PRNTST
327 031D 86 16   P15     LDA A  #$16
328 031F BD 015D         JSR    PRNTST
329 0322 7A 09D8         DEC    CRCT       DECREMENT CR COUNTER
330 0325 26 03           BNE    P29
331 0327 BD 035A         JSR    FRMFD
332 032A B6 09E5 P29     LDA A  LAST
333 032D 7F 09ED         CLR    TABCT
334 0330         P11     EQU    *
335 0330         P9      EQU    *
336 0330 B6 09E5         LDA A  LAST
337 0333 33      P10     PUL B
338 0334 39              RTS

340 0335 B7 09E5 LOUT    STA A  LAST
341 0338 BD 015D         JSR    PRNTST
342 033B B6 09E5         LDA A  LAST
343 033E 81 0D           CMP A  #$D
344 0340 39              RTS

346 0341         EOB     EQU    *
347 0341 7D 0A8A         TST    ENDFLG
348 0344 27 13           BEQ    EOBRTS
349 0346 7D 09D8 P2      TST    CRCT
350 0349 27 08           BEQ    *+10
351 034B BD 0ADF         JSR    OCRLF
352 034E 7A 09D8         DEC    CRCT
353 0351 20 F3           BRA    P2
354 0353 BD 0874         JSR    PGPT
355 0356 BD 086A         JSR    PGAD
356 0359 39      EOBRTS  RTS

358              * GENERATE FORM FEED
359 035A B6 0968 FRMFD   LDA A  CRCT1
```

```
360 0350 B7 09D8          STA A  CRCT
361 0360 BD 0B74          JSR    PGPT
362 0363 BD 0B6A          JSR    PGAD
363 0366 7D 0A06 FRMFD1   TST    STPFLG
364 0369 26 02            BNE    *+4          SKIP IF HALT=OFF
365 036B BD 0D            BSR    FM1
366 036D CE 03D7          LDX    #CRLF
367 0370 BD 010F          JSR    PDATA1
368 0373 B6 096A          LDA A  HDR
369 0376 81 20            CMP A  #$20
370 0378 27 1F            BEQ    HD03
371 037A 7D 096A          TST    HDR
372 037D 27 19            BEQ    HD03
373 037F CE 096A          LDX    #HDR
374 0382 A6 00    HD01    LDA A  0,X
375 0384 08               INX
376 0385 BD AE            BSR    LOUT
377 0387 26 F9            BNE    HD01
378 0389 BD 0BDF          JSR    OCRLF
379 038C BD 0BDF          JSR    OCRLF
380 038F 7A 09D8          DEC    CRCT
381 0392 7A 09D8          DEC    CRCT

383 0395 7A 09D8 HD05     DEC    CRCT
384 0398 CE 039F HD03     LDX    #CM
385 039B BD 010F          JSR    PDATA1
386 039E 39       HD02    RTS

388               *HOME-UP AND CLEAR SCREEN
389 039F 0D       CM      FCB    $D,$16,$4
390 03A2 B7 09D9 TNBLK    STA A  BLKFLG
391 03A5 7F 09DC          CLR    LNGCTR
392 03A8 7F 09E6          CLR    ELMSPC
393 03AB 7E 0292          JMP    LOOP

395 03AE 1B       CHGMSG  FCB    $1B,$2A     ESC * FOR SOROC
396 03B0 43               FCC    /CHANGE FORM - 1=SKIP PAGE  0=PRINT PAGE/
397 03D7 0D       CRLF    FCB    $D,$A,4
398               *  PAGE CHANGE
399               * SKIP = 0 = PRINT NEXT PAGE
400               * SKIP = 1 = SKIP NEXT PAGE
401               *
402 03DA CE 03AE FM1      LDX    #CHGMSG
403 03DD BD 010F          JSR    PDATA1
404 03E0 CE 0A93          LDX    #PGCT
405 03E3 BD 0109          JSR    OUT2HS
406 03E6 BD 0112          JSR    INCH
407 03E9 36               PSH A
408 03EA BD 0106          JSR    PRNINZ      INIT PRINTR BEFORE EACH PAGE
409 03ED 32               PUL A
410 03EE 81 31            CMP A  #'1
411 03F0 27 04            BEQ    FM2         SKIP THE PAGE
412 03F2 7F 09E4          CLR    SKIP
413 03F5 39               RTS

415 03F6 7C 09E4 FM2      INC    SKIP
416 03F9 39               RTS
417 03FA 7E 0292 LOOP2    JMP    LOOP

419               *
420               * BLOCK ROUTINE
421               *
422 03FD          RLKRT2  EQU    *
423 03FD 81 0D            CMP A  #$0D
424 03FF 27 09            BEQ    EL1
425 0401 81 20            CMP A  #$20
426 0403 27 05            BEQ    EL1
427 0405 7F 09E6          CLR    ELMSPC
428 0408 20 09            BRA    EL2

430               *
431               * ELIMINATE SPACES OR CR AT BEGINING OF NEW LINE
432               *
433 040A 7D 09E6 EL1      TST    ELMSPC
434 040D 26 E8            BNE    LOOP2
435               *
436               * ELIMINATE SPACE JUST BEFORE A CR
437               *
438 040F C1 20            CMP B  #'           LAST = SPACE?
439 0411 27 E7            BEQ    LOOP2        BR = YES = SPACE FOLLOWED BY CR
440 0413 81 0D    EL2     CMP A  #$D
441 0415 26 02            BNE    BLK3
442 0417 86 20            LDA A  #$20

443 0419 81 3E    BLK3    CMP A  #$3E         RIGHT CARAT?  >
444 041B 27 24            BEQ    TFBLK
445 041D BD 0451          JSR    LDBUF
446 0420 81 011E          CMP A  BKSPC
447 0423 26 05            BNE    *+7
448 0425 7A 09DC          DEC    LNGCTR
449 0428 20 0D            BRA    BLK1
450 042A B1 0119          CMP A  HALT
451 042D 27 08            BEQ    BLK1         SUPRESS LINE COUNTER INCR
452 042F 7D 0A05          TST    HFLAG
453 0432 26 03            BNE    BLK1
454 0434 7C 09DC          INC    LNGCTR
455 0437 F6 0969 BLK1     LDA B  LENGTH
456 043A F1 09DC          CMP B  LNGCTR
457 043D 27 5D            BEQ    LSTSPC
458 043F 20 B9            BRA    LOOP2

460 0441 7F 09D9 TFBLK    CLR    BLKFLG
461 0444 BD 0618          JSR    PLSBUF
462 0447 FE 09DA TFBLK1   LDX    BLKLNE
463 044A 86 0D            LDA A  #$D
464 044C A7 00            STA A  0,X
465 044E 7E 0501          JMP    OUTOUT

467 0451 FF 09D5 LDBUF    STX    LDBX
468 0454 B1 0119          CMP A  HALT
469 0457 26 02            BNE    *+4
470 0459 BD 2E            BSR    HLTRTN
471 045B 7D 0A05 LDBUF1   TST    HFLAG
472 045E 26 1F            BNE    LDBUF2
473 0460 FE 09DA          LDX    BLKLNE
474 0463 A7 00            STA A  0,X
475 0465 08               INX
476 0466 FF 09DA          STX    BLKLNE
477 0469 FE 09D5          LDX    LDBX
478 046C B1 011E          CMP A  BKSPC
479 046F 26 04            BNE    *+6
480 0471 7A 09ED          DEC    TABCT
481 0474 39               RTS
482 0475 B1 0119          CMP A  HALT
483 0478 27 03            BEQ    *+5
484 047A 7C 09ED          INC    TABCT
485 047D 39               RTS

487 047E FE 011C LDBUF2   LDX    HPTRHI
488 0481 08               INX
489 0482 FF 011C          STX    HPTRHI
490 0485 FE 09D5          LDX    LDBX
491 0488 39               RTS

493 0489 7D 0A05 HLTRTN   TST    HFLAG
494 048C 26 0A            BNE    SETFLG
495 048E FF 011A          STX    HPTRLO
496 0491 FF 011C          STX    HPTRHI
497 0494 7C 0A05          INC    HFLAG
498 0497 39               RTS

500 0498 7F 0A05 SETFLG   CLR    HFLAG
501 049B 39               RTS

503 049C 7F 09E1 LSTSPC   CLR    FILCTR
504 049F 20 07            BRA    LS3
505 04A1 81 20    LS1     CMP A  #'           SPACE
506 04A3 27 15            BEQ    LS2
507 04A5 7C 09E1 L        INC    FILCTR
508 04A8          LS3     EQU    *
509 04A8 FE 09DF          LDX    BUFPNT
510 04AB 09               DEX
511 04AC FF 09DF          STX    BUFPNT
512 04AF FE 09DA          LDX    BLKLNE
513 04B2 09               DEX
514 04B3 FF 09DA          STX    BLKLNE
515 04B6 A6 00            LDA A  0,X
516 04B8 20 E7            BRA    LS1

518               * CHECK FOR MULTIPLE SPACES AT END

520 04BA 09       LS2     DEX
521 04BB A6 00            LDA A  0,X
522 04BD 81 20            CMP A  #$20         ANOTHER SPACE
523 04BF 27 E4            BEQ    L            YES
524 04C1 08               INX                 NO,REPOSITION
525 04C2 86 0D            LDA A  #$0D         PUT IN A CR
```

```
526 04C4 A7 00          STA A  0,X
527 04C6 FF 09DD        STX    CRPOS
528 04C9 20 10          BRA    FL2

530 04CB CE 0A09 FILL   LDX    #BLOCK
531 04CE A6 00   FLOP   LDA A  0,X
532 04D0 81 20          CMP A  #' 		SPACE
533 04D2 27 0E          BEQ    SHRT
534 04D4 81 0D          CMP A  #$0D
535 04D6 27 03          BEQ    FL2
536 04D8 08             INX
537 04D9 20 F3          BRA    FLOP
538 04DB 7D 09E1 FL2    TST    FILCTR
539 04DE 26 EB          BNE    FILL
540 04E0 20 1F          BRA    OUTOUT

542 04E2 FF 09E2 SHRT   STX    FILPNT
543 04E5 FE 09DD        LDX    CRPOS
544 04E8 08             INX
545 04E9 FF 09DD        STX    CRPOS
546 04EC 09             DEX
547 04ED A6 00   SH2    LDA A  0,X
548 04EF A7 01          STA A  1,X
549 04F1 09             DEX
550 04F2 BC 09E2        CPX    FILPNT
551 04F5 26 F6          BNE    SH2
552 04F7 08      INSRT  INX
553 04F8 86 FF          LDA A  #$FF
554 04FA A7 00          STA A  0,X
555 04FC 7A 09E1        DEC    FILCTR
556 04FF 26 CD          BNE    FLOP
557 0501 7D 09D7 OUTOUT TST    HDFLG 		TEST FOR NEW HEADER
558 0504 27 21          BEQ    HD4 		BR NO
559 0506 7F 0510        CLR    HD1+1
560 0509 7F 0515        CLR    HD2+1
561 050C CE 0A09 HD5    LDX    #BLOCK
562 050F A6 00   HD1    LDA A  0,X
563 0511 CE 096A        LDX    #HDR
564 0514 A7 00   HD2    STA A  0,X
565 0516 81 0D          CMP A  #$D
566 0518 26 05          BNE    HD3
567 051A 7F 09D7        CLR    HDFLG
568 051D 20 19          BRA    OT1
569 051F 7C 0510 HD3    INC    HD1+1
570 0522 7C 0515        INC    HD2+1
571 0525 20 E5          BRA    HD5

573                     *
574                     * NORMAL OUTPUT
575                     *
576 0527 CE 0A09 HD4    LDX    #BLOCK
577 052A A6 00   OTOT   LDA A  0,X
578 052C 81 FE          CMP A  #$FE
579 052E 27 08          BEQ    OT1
580 0530 BD 0307        JSR    OUTPUT
581 0533 08             INX
582 0534 81 0D          CMP A  #$0D
583 0536 26 F2          BNE    OTOT
584 0538 CE 0A09 OT1    LDX    #BLOCK
585 053B FF 09DA        STX    BLKLNE
586 053E 7F 09DC        CLR    LNGCTR
587 0541 7D 09EB        TST    MARG
588 0544 27 03          BEQ    OT3
589 0546 BD 0758        JSR    DOMARG
590 0549 7D 09EC OT3    TST    KPLNG 		WAS THIS AN INDENT?
591 054C 27 09          BEQ    OT4 		NO
592 054E F6 09EC        LDA B  KPLNG 		YES
593 0551 F7 0969        STA B  LENGTH
594 0554 7F 09EC        CLR    KPLNG
595 0557 7C 09E6 OT4    INC    ELMSPC
596 055A 7E 03FA        JMP    LOOP2

598                     *
599                     * CENTER ROUTINE
600                     *
601 055D B1 0119 RTNCNT CMP A  HALT
602 0560 26 03          BNE    *+5
603 0562 BD 0489        JSR    HLTRTN
604 0565 7D 0A05        TST    HFLAG
605 0568 26 15          BNE    CNT5
606 056A B1 5D          CMP A  #$5D
607 056C 27 14          BEQ    CNT1 		YES
608 056E B1 0D          CMP A  #$D 		END OF CENTER LINE?
609 0570 27 10          BEQ    CNT1 		YES
610 0572 B1 011E        CMP A  BKSPC
611 0575 26 05          BNE    *+7
612 0577 7A 09E8        DEC    CNTRCT
613 057A 20 03          BRA    CNT5
614 057C 7C 09E8        INC    CNTRCT
615 057F 7E 0292 CNT5   JMP    LOOP

617                     *
618                     * LOAD BUFFER
619                     *
620 05A2 F6 0969 CNT1   LDA B  LENGTH
621 05A5 F0 09E8        SUB B  CNTRCT
622 05A8 57             ASR B
623 05A9 86 FF   CNT2   LDA A  #$FF
624 05AB BD 0451        JSR    LDBUF 		PUT CHAR IN BUFF
625 05AE 5A             DEC B
626 05AF 26 F8          BNE    CNT2
627 0591 FE 09E9        LDX    CNTBUF
628 0594 08      CNT3   INX
629 0595 A6 00          LDA A  0,X
630 0597 8D 1F          BSR    CNTHLT
631 0599 B1 5D          CMP A  #$5D 		RIGHT BRACKET?
632 059B 27 09          BEQ    CNT4 		YES
633 059D B1 0D          CMP A  #$D
634 059F 27 08          BEQ    CNT6
635 05A1 BD 0451        JSR    LDBUF
636 05A4 20 EE          BRA    CNT3
637 05A6 08      CNT4   INX
638 05A7 A6 00          LDA A  0,X 		GET CARRIAGE RETURN
639 05A9 BD 0451 CNT6   JSR    LDBUF
640 05AC FF 09DF        STX    BUFPNT
641 05AF BD 0618        JSR    PLSBUF
642 05B2 7F 09E7        CLR    CENTER
643 05B5 7E 0501        JMP    OUTOUT

645 05B8 B1 0119 CNTHLT CMP A  HALT
646 05BB 27 01          BEQ    *+3
647 05BD 39             RTS

649 05BE BD 0451 NTHL   JSR    LDBUF
650 05C1 08             INX
651 05C2 A6 00          LDA A  0,X
652 05C4 B1 0119        CMP A  HALT
653 05C7 26 F5          BNE    NTHL
654 05C9 39             RTS

656 05CA CE 09D1 CONVRT LDX    #BEGA
657 05CD 7F 09D3        CLR    TEMP
658 05D0 6F 00          CLR    0,X
659 05D2 6F 01          CLR    1,X
660 05D4 BD 06D4        JSR    GETCVT
661 05D7 81 30   DECHEX CMP A  #'0
662 05D9 2B 04          BMI    CVT2
663 05DB 81 39          CMP A  #'9
664 05DD 2F 03          BLE    CVT3
665 05DF 7E 0742 CVT2   JMP    CONTRL
666 05F2 84 0F   CVT3   AND A  #$0F
667 05F4 B7 09D0        STA A  TW
668 05F7 A6 00          LDA A  0,X
669 05F9 F6 01          LDA B  1,X
670 05FB 58             ASL B
671 05FC 49             ROL A
672 05FD 8D 1C          BSR    CVT5
673 05FF EB 01          ADD B  1,X
674 05F1 A9 00          ADC A  0,X
675 05F3 FB 09D0        ADD B  TW
676 05F6 89 00          ADC A  #$00
677 05F8 A7 00          STA A  0,X
678 05FA E7 01          STA B  1,X
679 05FC BD 06D4        JSR    GETCVT
680 05FF B1 2C          CMP A  #',
681 0601 27 04          BEQ    CVT8
682 0603 B1 0D          CMP A  #$0D
683 0605 26 D0          BNE    DECHEX
684 0607 FE 09DF CVT8   LDX    BUFPNT
685 060A 39             RTS

687 060B 68 01   CVT5   ASL    1,X
688 060D 69 00          ROL    0,X
689 060F 68 01          ASL    1,X
690 0611 69 00          ROL    0,X
691 0613 68 01          ASL    1,X
```

```
0615 69 00          ROL    0,X
0617 39             RTS

0618 FE 09DF PLSBUF LDX    BUFPNT
061B 08             INX
061C FF 09DF        STX    BUFPNT
061F 39             RTS

0620 FE 09DF SPCTL  LDX    BUFPNT
0623 A6 00          LDA A  0,X
0625 BD 0618        JSR    PLSBUF
                * COMMAND LOOK-UP TABLE
0628 81 4C          CMP A  #'L
062A 26 03          BNE    SPCT
062C 7E 06E3        JMP    CVT10
062F 81 50  SPCT    CMP A  #'P
0631 26 03          BNE    SPC0
0633 7E 06F0        JMP    CVT11
0636 81 4D  SPC0    CMP A  #'M
0638 26 03          BNE    SPC1
063A 7E 0749        JMP    MARGIN

063D 81 49  SPC1    CMP A  #'I
063F 26 03          BNE    SPC2
0641 7E 076C        JMP    INDENT
0644        SPC2    EQU    *
0644 81 53          CMP A  #'S
0646 26 03          BNE    SPC3
0648 7E 0783        JMP    TABSET
064B        SPC3    EQU    *
064B 81 54          CMP A  #'T
064D 26 03          BNE    SPC5
064F 7E 07A3        JMP    DOTAB
0652        SPC5    EQU    *
0652 81 5A          CMP A  #'Z
0654 26 03          BNE    SPC7
0656 7E 07C9        JMP    ZRTN
0659        SPC7    EQU    *
0659 81 43          CMP A  #'C
065B 26 03          BNE    SPC8
065D 7E 07F5        JMP    CRTN
0660        SPC8    EQU    *
0660 81 46          CMP A  #'F
0662 26 03          BNE    SPC9
0664 7E 08EE        JMP    FRTN
0667        SPC9    EQU    *
0667 81 52          CMP A  #'R
0669 26 03          BNE    SPC10
066B 7E 082D        JMP    INIT1
066E        SPC10   EQU    *
066E 81 2D          CMP A  #'-
0670 26 03          BNE    SPC11
0672 7E 0836        JMP    MINUS
0675        SPC11   EQU    *
0675 81 48          CMP A  #'H
0677 26 03          BNE    SPC12
0679 7E 0864        JMP    HDON
067C        SPC12   EQU    *
067C 81 42          CMP A  #'B
067E 26 03          BNE    SPC13
0680 7E 0936        JMP    BRTN
0683        SPC13   EQU    *
0683 81 45          CMP A  #'E
0685 26 03          BNE    SPC14
0687 7E 0944        JMP    ERTN
068A        SPC14   EQU    *
068A 81 55          CMP A  #'U          SET UPRINTABLE CHARACTER
068C 26 03          BNE    SPC15
068E 7E 0817        JMP    URTN
0691        SPC15   EQU    *
0691 81 57          CMP A  #'W
0693 26 03          BNE    SPC16
0695 7E 0822        JMP    WRTN
0698        SPC16   EQU    *
0698 81 4E          CMP A  #'N
069A 26 03          BNE    SPC17
069C 7E 06A2        JMP    NRTN

                * PERIOD ONLY COMMAND
069F 7E 0447 SPC17  JMP    TFBLK1

06A2 E6 00  NRTN    LDA B  0,X
06A4 C1 30          CMP B  #'0
06A6 26 05          BNE    NRTN1
06A8 7C 0A06        INC    STPFLG       TURN HALTS OFF
06AB 20 1E          BRA    NRTN5
06AD C1 31  NRTN1   CMP B  #'1
06AF 26 05          BNE    NRTN2
06B1 7F 0A06        CLR    STPFLG       TURN HALTS ON
06B4 20 15          BRA    NRTN5
06B6 C1 50  NRTN2   CMP B  #'P
06B8 26 05          BNE    NRTN3
06BA 7C 0A07        INC    PGPTFG       TURN OFF PAGE # PRINTING
06BD 20 0C          BRA    NRTN5
06BF C1 53  NRTN3   CMP B  #'S
06C1 26 05          BNE    NRTN4
06C3 7F 0A07        CLR    PGPTFG       TURN PAGE # PRINT ON
06C6 20 03          BRA    NRTN5
06C8 7E 0742 NRTN4  JMP    CONTRL       ERROR
06CB BD 0618 NRTN5  JSR    PLSBUF
06CE BD 0618        JSR    PLSBUF
06D1 7E 0292        JMP    LOOP

06D4 FF 09CE GETCVT STX    KPCT
06D7 FE 09DF        LDX    BUFPNT
06DA A6 00          LDA A  0,X
06DC BD 0618        JSR    PLSBUF
06DF FE 09CE        LDX    KPCT
06E2 39             RTS

06E3 BD 05CA CVT10  JSR    CONVRT
06E6 F6 09D2        LDA B  BEGA+1
06E9 5C             INC B
06EA F7 0969        STA B  LENGTH
06ED 7E 0292        JMP    LOOP

06F0 F6 00  CVT11   LDA B  0,X
06F2 C1 53          CMP B  #'S
06F4 26 1A          BNE    CVT11B       NO SKIP PAGE #
                * SKIP N PAGE
06F6 BD 0618        JSR    PLSBUF
06F9 BD 05CA        JSR    CONVRT
06FC F6 09D2        LDA B  BEGA+1
06FF B6 0A93        LDA A  PGCT
0702 5D     CVT11A  TST B
0703 26 03          BNE    *+5
0705 7E 0292        JMP    LOOP
0708 5A             DEC B
0709 4C             INC A
070A 19             DAA
070B B7 0A93        STA A  PGCT
070E 20 F2          BRA    CVT11A

0710 C1 2C  CVT11B  CMP B  #',
0712 27 04          BEQ    PAGE
0714 C1 0D          CMP B  #$D
0716 26 16          BNE    CVT11C       NOT PAGE COMMAND

                * PAGE COMMAND. CAUSES NEW PAGE
0718 BD 0618 PAGE   JSR    PLSBUF
071B 7D 09D8 PG4    TST    CRCT
071E 27 08          BEQ    PG5
0720 BD 08DF        JSR    OCRLF
0723 7A 09D8        DEC    CRCT
0726 20 F3          BRA    PG4
0728 BD 035A PG5    JSR    FRMFD
072B 7E 0292        JMP    LOOP

                * START PAGE NUMBER WITH NUMBER INPUTTED
072E BD 05CA CVT11C JSR    CONVRT
0731 F6 09D2        LDA B  BEGA+1
0734 7F 0A93        CLR    PGCT
0737 4F             CLR A
0738 20 C8          BRA    CVT11A

073A 10     CVTMSG  FCB    $10+$16
073C 45             FCC    /ERROR/
0741 04             FCB    $4

0742 CE 073A CONTRL LDX    #CVTMSG
0745 BD 010F        JSR    PDATA1
0748 39             RTS

0749 BD 05CA MARGIN JSR    CONVRT
074C F6 09D2        LDA B  BEGA+1
074F F7 09EB        STA B  MARG
```

```
858 0752 BD 0758         JSR    DOMARG
859 0755 7E 0292         JMP    LOOP

861 0758 CE 0A09 DOMARG LDX    #BLOCK
862 075B FF 09DA         STX    BLKLNE
863 075E F6 09EB         LDA B  MARG
864 0761 27 08           BEQ    DM2
865 0763 86 FF    DM1    LDA A  #$FF
866 0765 BD 0451         JSR    LDBUF
867 0768 5A              DEC B
868 0769 26 F8           BNE    DM1
869 076B 39       DM2    RTS

871                      *
872                      * PLUS INDENT
873                      *
874 076C BD 05CA INDENT JSR    CONVRT
875 076F F6 0969         LDA B  LENGTH
876 0772 F7 09EC         STA B  KPLNG
877 0775 F0 09D2         SUB B  BEGA+1
878 0778 F7 0969         STA B  LENGTH
879 077B F6 09D2         LDA B  BEGA+1
880 077E BD E3           BSR    DM1
881 07A0 7E 0292         JMP    LOOP

883 07A3 BD 05CA TABSET JSR    CONVRT
884 07A6 F6 09D2         LDA B  BEGA+1
885 07A9 27 10           BEQ    CLRTAB
886 07AB FE 09EE         LDX    TABIX
887 07AE F7 00           STA B  0,X
888 0790 08              INX
889 0791 FF 09EE         STX    TABIX
890 0794 C6 FF    CL1    LDA B  #$FF
891 0796 F7 00           STA B  0,X
892 0798 7E 0292         JMP    LOOP

894 079B CE 09F0 CLRTAB LDX    #TABTBL
895 079E FF 09EE         STX    TABIX
896 07A1 20 F1           BRA    CL1

898 07A3 BD 0618 DOTAB  JSR    PLSBUF
899 07A6 F6 09ED         LDA B  TABCT      CURRENT POSITION
900 07A9 5C              INC B
901 07AA CE 09F0         LDX    #TABTBL
902 07AD A6 00    DT1    LDA A  0,X
903 07AF 81 FF           CMP A  #$FF
904 07B1 27 13           BEQ    DT3
905 07B3 11              CBA
906 07B4 22 03           BHI    DT2
907 07B6 08              INX
908 07B7 20 F4           BRA    DT1
909 07B9 A6 00    DT2    LDA A  0,X
910 07BB 11              CBA
911 07BC 27 08           BEQ    DT3
912 07BE 86 0D           LDA A  #'
913 07C0 BD 0451         JSR    LDBUF
914 07C3 5C              INC B
915 07C4 20 F3           BRA    DT2
916 07C6 7E 0292 DT3     JMP    LOOP

918                      *
919                      * PRINTER CONTROL SWITCHES
920                      *
921 07C9 A6 00    ZRTN   LDA A  0,X
922 07CB BD 0618         JSR    PLSBUF
923 07CE 81 30           CMP A  #'0
924 07D0 26 05           BNE    ZRTN1
925 07D2 7F 0967         CLR    PNTON
926 07D5 20 45           BRA    URTN1
927 07D7 81 31    ZRTN1  CMP A  #'1
928 07D9 26 05           BNE    ZRTN2
929 07DB B7 0967         STA A  PNTON
930 07DE 20 3C           BRA    URTN1
931 07F0 81 32    ZRTN2  CMP A  #'2
932 07F2 26 05           BNE    ZRTN3
933 07F4 B7 0A08         STA A  FRMFLG
934 07F7 20 33           BRA    URTN1
935 07F9 81 33    ZRTN3  CMP A  #'3
936 07FB 26 05           BNE    ZRTN4
937 07FD 7F 0A08         CLR    FRMFLG
938 07F0 20 2A           BRA    URTN1
939 07F2 7E 0742 ZRTN4   JMP    CONTRL

941                      *
```

```
942                      * SET PRINTER LINE COUNT
943                      *
944 07F5          CRTN   EQU    *
945 07F5 E6 00           LDA B  0,X
946 07F7 C1 0D           CMP B  #$D
947 07F9 27 16           BEQ    CRTN1
948 07FB C1 2C           CMP B  #',
949 07FD 27 12           BEQ    CRTN1
950 07FF BD 05CA         JSR    CONVRT
951 0802 B6 09D2         LDA A  BEGA+1
952                      * ADJUST FOR 3 LINES AT BOTTOM OF PAGE
953 0805 4A              DEC A
954 0806 4A              DEC A
955 0807 4A              DEC A
956 0808 B7 0968         STA A  CRCT1
957 080B B7 09D8         STA A  CRCT
958 080E 7E 0292         JMP    LOOP

960 0811 BD 0618 CRTN1  JSR    PLSBUF
961 0814 7E 02F7         JMP    TNCNT

963                      *
964                      * SET UNPRINTABLE CHARACTER
965                      *
966 0817 A6 00    URTN   LDA A  0,X
967 0819 B7 0118         STA A  NOPRT
968 081C BD 0618 URTN1  JSR    PLSBUF
969 081F 7E 0292         JMP    LOOP

971 0822 A6 00    WRTN   LDA A  0,X
972 0824 B7 0119         STA A  HALT       SET HALT CHAR
973 0827 BD 0618         JSR    PLSBUF
974 082A 7E 0292         JMP    LOOP

976                      *
977                      * INITIALIZE PRINTER
978                      *
979 082D BD 0618 INIT1  JSR    PLSBUF
980 0830 BD 0106         JSR    PRNINZ
981 0833 7E 0292         JMP    LOOP

983                      *
984                      * MINUS ROUTINES
985                      *
986 0836 FE 09DF MINUS  LDX    BUFPNT
987 0839 A6 00           LDA A  0,X
988 083B BD 0618         JSR    PLSBUF
989 083E 81 49           CMP A  #'I
990 0840 27 03           BEQ    MI1
991 0842 7E 0742         JMP    CONTRL
992 0845 BD 05CA MI1    JSR    CONVRT
993 0848 F6 0969         LDA B  LENGTH
994 084B F7 09EC         STA B  KPLNG
995 084E FB 09D2         ADD B  BEGA+1
996 0851 F7 0969         STA B  LENGTH
997 0854 FE 09DA         LDX    BLKLNE
998 0857 F6 09D2         LDA B  BEGA+1
999 085A 09       MI2    DEX
1000 085B 5A             DEC B
1001 085C 26 FC          BNE    MI2
1002 085E FF 09DA        STX    BLKLNE
1003 0861 7E 0292        JMP    LOOP


1005                     *
1006                     * TURN LOAD HEADER BUFF FLAG ON
1007                     *
1008 0864 7C 09D7 HDON   INC    HDFLG
1009 0867 7E 0292        JMP    LOOP


1011                     *
1012                     * PAGE ROUTINES
1013                     *
1014 086A         PGAD   EQU    *
1015 086A B6 0A93        LDA A  PGCT
1016 086D 8B 01          ADD A  #1
1017 086F 19             DAA
1018 0870 B7 0A93        STA A  PGCT
1019 0873 39             RTS

1021 0874 8D 69   PGPT   BSR    OCRLF
1022 0876 8D 67          BSR    OCRLF
1023 0878 B6 09EB        LDA A  MARG
```

```
1024 087B 4D       K1     TST A
1025 087C 27 05           BEQ    K2
1026 087E 4A              DEC A
1027 087F 8D 57           BSR    OUTS
1028 0881 20 F8           BRA    K1
1029              *
1030              * IF IN INDENT MODE AND LAST LINE OF PAGE,
1031              * THEN ADJUST PAGE NUMBER TO CENTER
1032 0883 7D 09EC K2      TST    KPLNG
1033 0886 26 05           BNE    K2A
1034 0888 B6 0969         LDA A  LENGTH
1035 088B 20 03           BRA    K2B

1037 088D B6 09EC K2A     LDA A  KPLNG
1038 0890 44      K2B     LSR A
1039 0891 4A              DEC A
1040 0892 8D 44   K3      BSR    OUTS
1041 0894 4A              DEC A
1042 0895 26 FB           BNE    K3
1043 0897 7D 0A07         TST    PGPTFG
1044 089A 26 20           BNE    PG3A        NO PRINT PAGE # IF SET
1045 089C B6 0A93         LDA A  PGCT
1046 089F 84 F0           AND A  #$F0
1047 08A1 26 04           BNE    PG1
1048 08A3 8D 33           BSR    OUTS
1049 08A5 20 08           BRA    PG2
1050 08A7 44      PG1     LSR A
1051 08A8 44              LSR A
1052 08A9 44              LSR A
1053 08AA 44              LSR A
1054 08AB 8A 30           ORA A  #$30
1055              * OUTPUT M.S. DIGIT
1056 08AD 8D 26           BSR    OUTJMP
1057 08AF B6 0A93 PG2     LDA A  PGCT
1058 08B2 84 0F           AND A  #$F
1059 08B4 8A 30           ORA A  #$30

1061              * OUTPUT L.S. DIGIT
1062 08B6 8D 1D           BSR    OUTJMP
1063 08B8 8D 25   PG3     BSR    OCRLF
1064 08BA 20 12           BRA    PG4A

1066 08BC 7D 0A06 PG3A    TST    STPFLG
1067 08BF 26 F7           BNE    PG3
1068 08C1 86 04           LDA A  #4          EOT TO IBM
1069 08C3 8D 015D         JSR    PRNTST
1070 08C6 8D 0112         JSR    INCH
1071 08C9 8D 0106         JSR    PRNINZ
1072 08CC 20 EA           BRA    PG3

1074 08CE 7D 0A08 PG4A    TST    FRMFLG      TEST FF FLAG; IF SET,
1075              *   OUTPUT FORM FEED STRING
1076 08D1 27 0A           BEQ    END1
1077 08D3 86 0C           LDA A  #$C         FORM FEED CHAR.
1078 08D5 7E 015D OUTJMP  JMP    PRNTST


1080 08D8 36      OUTS    PSH A
1081 08D9 86 20           LDA A  #$20
1082 08DB 8D F8           BSR    OUTJMP
1083 08DD 32              PUL A
1084 08DE 39      END1    RTS

1086 08DF CE 03D7 OCRLF   LDX    #CRLF
1087 08E2 A6 00   PDAT    LDA A  0,X
1088 08E4 81 04           CMP A  #4
1089 08E6 27 F6           BEQ    END1
1090 08E8 08              INX
1091 08E9 8D 016A         JSR    PRNT1
1092 08EC 20 F4           BRA    PDAT

1094 08EE A6 00   FRTN    LDA A  0,X
1095 08F0 81 30           CMP A  #'0
1096 08F2 27 0C           BEQ    FLINE

1098 08F4 8D 05CA FRTN1   JSR    CONVRT
1099 08F7 B6 09D2         LDA A  BEGA+1
1100 08FA B7 0966         STA A  FMFEED
1101 08FD 7E 0292         JMP    LOOP
1102 0900 E6 01   FLINE   LDA B  1,X
1103 0902 C1 2C           CMP B  #',
1104 0904 27 04           BEQ    FLINE1
1105 0906 C1 0D           CMP B  #$D
1106 0908 26 EA           BNE    FRTN1
1107 090A B6 0966 FLINE1  LDA A  FMFEED
1108 090D 36              PSH A
1109 090E 86 01           LDA A  #1
1110 0910 B7 0966         STA A  FMFEED
1111 0913 8D 08DF         JSR    OCRLF
1112 0916 32              PUL A
1113 0917 B7 0966         STA A  FMFEED
1114 091A 8D 0618         JSR    PLSBUF
1115 091D 7A 09D8         DEC    CRCT
1116 0920 26 09           BNE    FLINE2
1117 0922 8D 0618         JSR    PLSBUF
1118 0925 8D 035A         JSR    FRMFD
1119 0928 7E 0292         JMP    LOOP


1121 092B 8D 0618 FLINE2  JSR    PLSBUF
1122 092E 26 03           BNE    *+5
1123 0930 7E 035A         JMP    FRMFD
1124 0933 7E 0292         JMP    LOOP


1126 0936         BRTN    EQU    *           BLOCK ROUTINE
1127 0936 7D 09D9         TST    BLKFLG
1128 0939 26 06           BNE    BRTN1
1129 093B 8D 0618         JSR    PLSBUF
1130 093E 7E 03A2         JMP    TNBLK


1132 0941 7E 0441 BRTN1   JMP    TFBLK
1133 0944 7C 0A04 ERTN    INC    ASIS
1134 0947 8D 0618         JSR    PLSBUF
1135 094A 7E 0292         JMP    LOOP


1137              * BREAK ROUTINE FOR ACIA

1139 094D B6 8008 BREAK   LDA A  TERMDA-1    GET STATUS
1140 0950 47              ASR A
1141 0951 25 03           BCS    BRK
1142 0953 4F      NOBRK   CLR A
1143 0954 4A              DEC A
1144 0955 39              RTS

1146 0956 B6 8009 BRK     LDA A  TERMDA
1147 0959 81 03           CMP A  #3           AC?
1148 095B 26 F6           BNE    NOBRK
1149 095D 8D 0112         JSR    INCH
1150 0960 81 43           CMP A  #'C
1151 0962 27 EF           BEQ    NOBRK        CONTINUE ON C
1152 0964 39              RTS
1153 0965 39              RTS

1155 0966 01      FMFEED  FCB    1
1156 0967 01      PNTON   FCB    1
1157 0968 23      CRCT1   FCB    35
1158 0969 40      LENGTH  FCB    64
1159 096A         HDR     RMB    100
1160 09CE 00      KPCT    FCB    0,0
1161 09D0 00      TW      FCB    0
1162 09D1 00      BEGA    FCB    0,0
1163 09D3 00      TEMP    FCB    0,0
1164 09D5 00      LDBX    FCB    0,0

1166 09D7 00      HDFLG   FCB    0
1167 09D8 00      CRCT    FCB    0
1168 09D9 00      BLKFLG  FCB    0
1169 09DA 00      BLKLNE  FCB    0,0
1170 09DC 00      LNGCTR  FCB    0
1171 09DD 00      CRPOS   FCB    0,0
1172 09DF 00      BUFPNT  FCB    0,0
1173 09E1 00      FILCTR  FCB    0
1174 09E2 00      FILPNT  FCB    0,0
1175 09E4 00      SKIP    FCB    0

1177 09E5 00      LAST    FCB    0
1178 09E6 00      ELMSPC  FCB    0
1179 09E7 00      CENTER  FCB    0
1180 09E8 00      CNTRCT  FCB    0
1181 09E9 00      CNTBUF  FCB    0,0
1182 09EB 00      MARG    FCB    0
1183 09EC 00      KPLNG   FCB    0
1184 09ED 00      TABCT   FCB    0
```

```
1185 09EE 00        TABIX   FCB     0,0
1186 09F0           TABTBL  RMB     20

1188 0A04 00        ASIS    FCB     0
1189 0A05 00        HFLAG   FCB     0
1190 0A06 00        STPFLG  FCB     0           HALT DISABLE FLAG
1191 0A07 00        PGPTFG  FCB     0           PAGE PRINT FLAG
1192 0A08 00        FRMFLG  FCB     0           FORM FEED FLAG
1193 0A09           BLOCK   RMB     128
1194 0A89           FLAG    RMB     1
1195 0A8A           ENDFLG  RMB     1
1196 0A8B           XTEM    RMB     2
1197 0A8D           SRCEND  RMB     2
1198 0A8F           BUFEND  RMB     2
1199 0A91           SRCBEG  RMB     2
1200 0A93 00        PGCT    FCB     0
1201 0A94           FCB     RMB     170


1203 0B3E           BUFREG  EQU     *
1204                *
1205                        END
```

SYMBOL TABLE:

ASIS	0A04	BEGA	09D1	BKSPC	011E	BLK1	0437	BLK3	0419
BLKFLG	09D9	BLKLNE	09DA	BLKRT2	03FD	BLOCK	0A09	BREAK	094D
BRK	0956	BRTN	0936	BRTN1	0941	BUFREG	0B3E	BUFEND	0A8F
BUFPNT	09DF	CDFM	7783	CENTER	09E7	CHGMSG	03AE	CL1	0794
CLOSE	01E0	CLRTAB	079B	CM	039F	CNT1	0582	CNT2	0589
CNT3	0594	CNT4	05A6	CNT5	057F	CNT6	05A9	CNTBUF	09E9
CNTHLT	05B8	CNTRCT	09E8	CNTRTN	0304	CONT	01EE	CONTRL	0742
CONVRT	05CA	CRCT	09D8	CRCT1	0968	CRLF	03D7	CRPOS	09DD
CRTN	07F5	CRTN1	0811	CVT10	06E3	CVT11	06F0	CVT11A	0702
CVT11B	0710	CVT11C	072E	CVT2	05DF	CVT3	05E2	CVT5	060B
CVT8	0607	CVTMSG	073A	DECHEX	05D7	DFM	77B6	DM1	0763
DM2	076B	DOMARG	075B	DOTAB	07A3	DT1	07AD	DT2	07B9
DT3	07C6	EL1	040A	EL2	0413	ELMSPC	09E6	END1	08DE
ENDFIL	028C	ENDFLG	0A8A	EOB	0341	EOBRTS	0359	EREXIT	01CE
ERR	01A5	ERTN	0944	FCB	0A94	FILB1	024D	FILB2	0257
FILB3	0268	FILB4	026A	FILB5	026D	FILB6	0274	FILBUF	0247
FILCTR	09E1	FILL	04CB	FILPNT	09E2	FILSPC	01C6	FL2	04DB
FLAG	0A89	FLINE	0900	FLINE1	090A	FLINE2	092B	FLOP	04CE
FM1	03DA	FM2	03F6	FMFEED	0966	FRMFD	035A	FRMFD1	0366
FRMFLG	0A08	FRTN	08EE	FRTN1	08F4	GETCVT	06D4	HALT	0119
HD1	050F	HD2	0514	HD3	051F	HD4	0527	HD5	050C
HDFLG	09D7	HDO1	0382	HDO2	039E	HDO3	0398	HDO5	0395
HDON	0864	HDR	096A	HFLAG	0A05	HLTRTN	04B9	HPTRHI	011C
HPTRLO	011A	INCH	0112	INDENT	076C	INEEE	E1AC	INIT1	082D
INSRT	04F7	JEOB	02A9	K1	0878	K2	0883	K2A	088D
K2B	0890	K3	0892	KPCT	09CE	KPLNG	09FC	L	04A5
LAST	09E5	LDBUF	0451	LDBUF1	045B	LDBUF2	047E	LDBX	09D5
LENGTH	0969	LNGCTR	09DC	LOAD	0236	LOOP	0292	LOOP2	03FA
LOUT	0335	LP1	02D0	LP2	02DF	LP35	02D8	LP36	02CD
LP4	02AC	LS1	04A1	LS2	04BA	LS3	04A8	LSTSPC	049C
MARG	09EB	MARGIN	0749	MI1	0845	MI2	085A	MINUS	0836
MORE	01D7	NOBRK	0953	NOPRT	0118	NRTN	06A2	NRTN1	06AD
NRTN2	06B6	NRTN3	06BF	NRTN4	06C8	NRTN5	06CB	NTHL	058E
OCRLF	08DF	OT1	0538	OT2HS	E0CA	OT3	0549	OT4	0557
OT4HS	F0C8	OTOT	052A	OUT2HS	0109	OUT4HS	010C	OUTCH	0115
OUTEEE	E1D1	OUTFF	0183	OUTFF1	018F	OUTFF2	0199	OUTJMP	08D5
OUTOUT	0501	OUTPUT	0307	OUTS	08D8	P1	0313	P10	0333
P11	0330	P15	0310	P2	0346	P29	032A	P3	02E8
P32	0311	P8	0313	P9	0330	PAGE	0718	PDAT	08E2
PDATA	E07E	PDATA1	010F	PG1	08A7	PG2	08AF	PG3	0888
PG3A	08BC	PG4	071B	PG4A	08CE	PG5	0728	PGAD	086A
PGCT	0A93	PGPT	0874	PGPTFG	0A07	PLSBUF	0618	PNTON	0967
PRINTR	0103	PRNINZ	0106	PRNT1	016A	PRNT2	015C	PRNTST	015D
QEXIT	01D1	Q04R	0004	QRED	0005	QSRC	0006	RBT1	017E
RBT2	0177	READ	0278	READ1	028E	RON	01A1	RTNCNT	055D
SETBUF	01A8	SETFLG	0498	SH2	04ED	SHRT	04E2	SKIP	09E4
SPC0	0636	SPC1	063D	SPC10	066E	SPC11	0675	SPC12	067C
SPC13	0683	SPC14	068A	SPC15	0691	SPC16	0698	SPC17	069F
SPC2	0644	SPC3	064B	SPC5	0652	SPC7	0659	SPC8	0660
SPC9	0667	SPCT	062F	SPCTL	0620	SRCBEG	0A91	SRCEND	0A8D
ST1	01F1	START	0100	START3	01BA	STPFLG	0A06	TABCT	09ED
TABIX	09EE	TABSET	0783	TABTBL	09F0	TEMP	09D3	TERMDA	8009
TFBLK	0441	TFBLK1	0447	TNBLK	03A2	TNCNT	02F7	TNCNT1	02F8
TW	09D0	URTN	0817	URTN1	081C	WAIA	013F	WAIT	011F
WAIT1	0125	WAIT2	0142	WARMS	01D4	WRTN	0822	XFC	0000
XTEM	0A8B	ZFLSPC	7291	ZGCHAR	7294	ZRTN	07C9	ZRTN1	07D7
ZRTN2	07E0	ZRTN3	07E9	ZRTN4	07F2	ZTYPDE	72A9	ZWARMS	7283

CROSS REFERENCE- WP68.SRC

SYMBOL	DEFINED	REFERENCED									
ASIS	1188 :	264	1133								
BEGA	1162 :	656	803	814	842	856	877	879	884	951	995
		998	1099								
BKSPC	57 :	446	478	610							
BLK1	455 :	449	451	453							
BLK3	443 :	441									
BLKFLG	1168 :	288	390	460	1127						
BLKLNE	1169 :	138	462	473	476	512	514	585	862	997	1002
BLKRT2	422 :	290									
BLOCK	1193 :	137	530	561	576	584	861				
BREAK	1139 :	100									
BRK	1146 :	1141									
BRTN	1126 :	752									
BRTN1	1132 :	1128									
BUFBEG	1203 :	132	188								
BUFEND	1198 :	193	213								
BUFPNT	1172 :	136	250	256	509	511	640	684	695	697	700
		796	986								
CDFM	29 :	153									
CENTER	1179 :	298	310	642							
CHGMSG	395 :	402									
CL1	890 :	896									
CLOSE	161 :	206									
CLRTAB	894 :	885									
CM	389 :	384									
CNT1	620 :	607	609								
CNT2	623 :	626									
CNT3	628 :	636									
CNT4	637 :	632									
CNT5	615 :	605	613								
CNT6	639 :	634									
CNTBUF	1181 :	309	627								
CNTHLT	645 :	630									
CNTRCT	1180 :	308	612	614	621						
CNTRTN	313 :	299									
CONT	168 :	151									
CONTRL	851 :	130	665	790	939	991					
CONVRT	656 :	802	813	841	855	874	883	950	992	1098	
CRCT	1167 :	115	118	180	329	349	352	360	380	381	383
		832	835	957	1115						
CRCT1	1157 :	179	359	956							
CRLF	397 :	69	366	1086							
CRPOS	1171 :	527	543	545							
CRTN	944 :	732									

SYMBOL	DEFINED	REFERENCED									
CRTN1	960 :	947	949								
CVT10	802 :	706									
CVT11	808 :	709									
CVT11A	816 :	823	845								
CVT11B	825 :	810									
CVT11C	841 :	828									
CVT2	665 :	662									
CVT3	666 :	664									
CVT5	687 :	672									
CVT8	684 :	681									
CVTMSG	847 :	851									
DECHEX	661 :	683									
DFM	28 :	164	197	236							
DM1	865 :	868	880								
DM2	869 :	864									
DOMARG	861 :	589	858								
DOTAB	898 :	724									
DT1	902 :	908									
DT2	909 :	906	915								
DT3	916 :	904	911								
EL1	433 :	424	426								
EL2	440 :	428									
ELMSPC	1178 :	392	427	433	595						
END1	1084 :	1076	1089								
ENDFIL	242 :	240									
ENDFLG	1195 :	187	205	220	347						
EOB	346 :	262									
EOBRTS	356 :	348									
EREXIT	152 :	166	198	241							
ERR	130 :	110									
ERTN	1133 :	756									

```
FILB1    212 :  227
FILB2    216 :  214
FILB3    225 :  219
FILB4    226 :  230
FILB5    228 :  217
FILB6    231 :  223
FILBUF   210 :  202
FILCTR  1173 :  503   507   538   555
FILL     530 :  539
FILPNT  1174 :  542   550
FILSPC   149 :  159
FL2      538 :  528   535
FLAG    1194 :  210   215   229
FLINE   1102 : 1096
FLINE1  1107 : 1104
FLINE2  1121 : 1116
FLOP     531 :  537   556
FM1      402 :  365
FM2      415 :  411

SYMBOL DEFINED REFERENCED
====== ======= ==========

FMFEED  1155 :  108   182  1100  1107  1110  1113
FRMFD    359 :  331   837  1118  1123
FRMFD1   363 :  203
FRMFLG  1192 :  933   937  1074
FRTN    1094 :  736
FRTN1   1098 : 1106
GETCVT   795 :  660   679
HALT      54 :   89   177   450   468   482   601   645   652   972
HD1      562 :  559   569
HD2      564 :  560   570
HD3      569 :  566
HD4      576 :  558
HD5      561 :  571
HDFLG   1166 :  557   567  1008
HDO1     374 :  377
HDO2     386 :
HDO3     384 :  370   372
HDO5     383 :
HDON    1008 :  748
HDR     1159 :  368   371   373   563
HFLAG   1189 :  303   452   471   493   497   500   604
HLTRTN   493 :  470   603
HPTRHI    56 :   61   487   489   496
HPTRLO    55 :   60    68    71   495
INCH      50 :   79   406  1070  1149
INDENT   874 :  716
INEEE     37 :   50
INIT1    979 :  740
INSRT    552 :
JEOB     262 :  259
K1      1024 : 1028
K2      1032 : 1025
K2A     1037 : 1033
K2B     1038 : 1035
K3      1040 : 1042
KPCT    1160 :  795   799
KPLNG   1183 :  590   592   594   876   994  1032  1037
L        507 :  523
LAST    1177 :  332   336   340   342
LDBUF    467 :  300   445   624   635   639   649   866   913
LDBUF1   471 :
LDBUF2   487 :  472
LDBX    1164 :   59    74    82    84   467   477   490
LENGTH  1158 :  171   185   455   593   620   805   875   878   993   996
               1034
LNGCTR  1170 :  391   448   454   456   586
LOAD     202 :  208
LOOP     249 :  204   302   304   311   393   417   615   793   806   818
                838   859   881   892   916   958   969   974   981  1003
               1009  1101  1119  1124  1135

SYMBOL DEFINED REFERENCED
====== ======= ==========

LOOP2    417 :  434   439   458   596
LOUT     340 :  323   376
LP1      288 :  271   285
LP2      296 :  293
LP35     292 :  289
LP36     286 :  269   273   275   277   284
LP4      264 :  261
LS1      505 :  516
LS2      520 :  506
LS3      508 :  504
LSTSPC   503 :  457
MARG    1182 :  587   857   863  1023
MARGIN   855 :  712
MI1      992 :  990
MI2      999 : 1001
MINUS    986 :  744
MORE     156 :  165
NOBRK   1142 : 1148  1151
NOPRT     53 :  315   967
NRTN     773 :  768
NRTN1    778 :  775
NRTN2    782 :  779
NRTN3    786 :  783
NRTN4    790 :  787
NRTN5    791 :  777   781   785   789
NTHL     649 :  653
OCRLF   1086 :  113   121   128   351   378   379   834  1021  1022  1063
               1111
OT1      584 :  568   579
OT2HS     35 :   47
OT3      590 :  588
OT4      595 :  591
OT4HS     30 :   48
OTOT     577 :  583
OUT2HS    47 :  405
OUT4HS    48 :
OUTCH     51 :   64    99
OUTEEE    38 :   51
OUTFF    107 :   92
OUTFF1   115 :  123
OUTFF2   121 :  116
OUTJMP  1078 : 1056  1062  1082
OUTOUT   557 :  305   465   540   643
OUTPUT   314 :  580
OUTS    1080 : 1027  1040  1048
P1       322 :
P10      337 :
P11      334 :
P15      327 :
P2       349 :  353

SYMBOL DEFINED REFERENCED
====== ======= ==========

P29      332 :  330
P3       300 :  265
P32      319 :  316
P8       321 :  318
P9       335 :  324
PDAT    1087 : 1092
PDATA     36 :   49
PDATA1    49 :   70   367   385   403   852
PG1     1050 : 1047
PG2     1057 : 1049
PG3     1063 : 1067  1072
PG3A    1066 : 1044
PG4      832 :  836
PG4A    1074 : 1064
PG5      837 :  833
PGAD    1014 :  355   362
PGCT    1200 :  145   174   404   815   822   843  1015  1018  1045  1057
PGPT    1021 :  354   361
PGPTFG  1191 :  784   788  1043
PLSBUF   695 :  461   641   702   791   792   798   812   898   922   960
                968   973   979   988  1114  1117  1121  1129  1134
PNTON   1156 :   97   183   925   929
PRINTR    45 :  104
PRNINZ    46 :   83   408   980  1071
PRNT1     95 : 1091
PRNT2     87 :   76    94   101
PRNTST    89 :   78   326   328   341  1069  1078
QEXIT    153 :  102   148
Q04R      32 :  195
QRED      33 :  199
QSRC      34 :  162
RBT1     104 :   98
RBT2     100 :   96   105
READ     234 :  212
```

```
READ1    243 :  237
RON      127 :  111
RTNCNT   601 :  313
SETBUF   132 :  186   207
SETFLG   500 :  494
SH2      547 :  551
SHRT     542 :  533
SKIP    1175 :   95   412   415
SPC0     710 :  708
SPC1     714 :  711
SPC10    741 :  739
SPC11    745 :  743
SPC12    749 :  747
SPC13    753 :  751
SPC14    757 :  755
SPC15    761 :  759

SYMBOL DEFINED REFERENCED
====== ======= ==========

SPC16    765 :  763
SPC17    771 :  767
SPC2     717 :  715
SPC3     721 :  719
SPC5     725 :  723
SPC7     729 :  727
SPC8     733 :  731
SPC9     737 :  735
SPCT     707 :  705
SPCTL    700 :  286
SRCREG  1199 :  190   211
SRCEND  1197 :  231   258
ST1      172 :  175
START     44 :
START3   144 :   44
STPFLG  1190 :   75   363   776   780  1066
TARCT   1184 :  333   480   484   899
TABIX   1185 :  886   889   895
TABSET   883 :  720
TABTBL  1186 :  894   901
TEMP    1163 :  657
TERMDA    23 : 1139  1146
TFBLK    460 :  444  1132
TFBLK1   462 :  771
TNBLK    390 :  294  1130
TNCNT    307 :  297   961
TNCNT1   908 :
TW      1161 :  667   675
URTN     966 :  760
URTN1    968 :  926   930   934   938
WAIA      72 :   67
WAIT      59 :   90
WAIT1     61 :   73
WAIT2     74 :   62
WARMS    154 :  158
WRTN     971 :  764
XFC       31 :  163   196   200
XTEM    1196 :  234   243
ZFLSPC    26 :  150
ZGCHAP    25 :  146   156
ZRTN     921 :  728
ZRTN1    927 :  924
ZRTN2    931 :  928
ZRTN3    935 :  932
ZRTN4    939 :  936
ZTYPDE    27 :  152
ZWARMS    24 :  154

TOTAL SYMBOLS:  283
```

The End

Falconer F·l·o·a·t·i·n·g Point Arithmetic

BY CHARLES B. FALCONER
Chief Instrumentation Engineer
Yale University
School of Medicine
333 Cedar Street
New Haven, Connecticut 06510

Herewith Falconer Floating Point Arithmetic II – the concluding chapter – which we continue from last issue. – SR

Listing

```
                        0080000 ;
                        0081000 ; ****************************************
                        0082000 ;   START THE CODE
                        0083000 ; ****************************************
                        0084000 ;
                        0085000 ; CHECK (A) TO BE IN RANGE 0-9 (ASCII)
                        0086000 ; RETURN CARRY FOR NON-NUMERIC CHARACTER
                        0087000 ; F
0000 FE3A               0088000 QNUM:   CPI     '9'+1;      THIS FIRST TO SPEED EXIT FOR
0002 3F                 0089000         CMC                                     ALPHA
0003 D8                 0090000         RC;                 < 0, NON-NUMERIC
0004 FE30               0091000         CPI     '0'
0006 C9                 0092000         RTN
                        0093000 ;
                        0094000 ; CHECK (HL) FOR VALUE < 6554
                        0095000 ; SET CARRY IF GREATER
                        0096000 ; F
0007 C5                 0097000 QMAX:   PUSH    B
0008 47                 0098000         MOV     B,A
0009 7D                 0099000         MOV     A,L
000A D69A               0100000         SUI     6554 MOD 256
000C 7C                 0101000         MOV     A,H
000D DE19               0102000         SBI     6554/256
000F 3F                 0103000         CMC
0010 78                 0104000         MOV     A,B
0011 C1                 0105000         POP     B
0012 C9                 0106000         RTN
                        0107000 ;
                        0108000 ; TRANSFER CONTROL TO (BC)
0013 C5                 0109000 JBC:    PUSH    B;          SET ADDRESS ON STACK
0000                    0110000 .LVL    SET     .LVL-1;     COMPENSATE FOR STACKED ADR.
0014 C9                 0111000         RTN;                GO EXECUTE
                        0112000 ;
                        0113000 ; INPUT A CHARACTER, IGNORING BLANKS
                        0114000 ; A,F
0015 CD1300             0115000 DEBLK:  GETCH;
0018 FE20               0116000         CPI     ' '
001A C0                 0117000         RNZ
001B C31500             0118000         JMP     DEBLK;      BYPASS A BLANK
                        0119000 ;
                        0120000 ; INPUT A FLOATING POINT VALUE FROM A CHAR. STRING
                        0121000 ; AT ENTRY:
                        0122000 ;   (BC)  => CHARACTER INPUT ROUTINE
                        0123000 ; AT EXIT:
                        0124000 ; (A)=(L) =  CHARACTER FOLLOWING NUMERICAL STRING
                        0125000 ;  (DE,H) = VALUE
                        0126000 ;     IF ERROR, (A)= ERROR CODE, (L)=EXIT CHAR., CARRY SET
                        0127000 ; CARRY FOR OVERFLOW OR ILLEGAL 1ST. CHAR.
                        0128000 ; A,F,D,E,H,L
001E CD1500             0129000 IVAL:   CALL    DEBLK;      BYPASS LEADING BLANKS
                        0130000 ;
                        0131000 ; ALTERNATE ENTRY WITH FIRST CHAR. IN (A)
0021 FE20               0132000 IVALC:  CPI     ' '
0023 CA1E00             0133000         JZ      IVAL;       IGNORE LEADING SPACES
0026 FE2B               0134000         CPI     '+'
0028 CA1E00             0135000         JZ      IVAL;       IGNORE UNARY +
002B CD0000             0136000         CALL    QNUM
002E 210000             0137000         LXI     H,0;        CLEAR ACC.
0031 1640               0138000         MVI     D,40H;      AND EXPONENT
0033 D29300             0139000         JNC     IVAL6;      INITIAL NUMERIC ENTRY
0036 FE2E               0140000         CPI     '.'
0038 CA4A00             0141000         JZ      IVAL1;      INITIAL DEC. PT.
003B FE2D               0142000         CPI     '-'
003D 37                 0143000         STC
003E 6F                 0144000         MOV     L,A
003F C0                 0145000         RNZ;                ERROR, RETURN 0 & CARRY, (A) IS
0040 CD1E00             0146000         CALL    IVAL;       RECURSIVE, UNARY -      ILLEGAL CHAR.
0043 F5                 0147000         PUSH    PSW;        SAVE EXIT CHAR
0044 7AEE8057           0148000         FSIGN   D
0048 F1                 0149000         POP     PSW
0049 C9                 0150000         RTN
004A CD1300             0151000 IVAL1:  GETCH;              AFTER INITIAL DEC. PT
004D CD0000             0152000         CALL    QNUM
0050 D27700             0153000         JNC     IVAL4;      GOT THE REQUIRED DIGIT
0053 6F                 0154000         MOV     L,A;        EXIT CHAR TO (L)
0054 37                 0155000         STC
0055 C9                 0156000         RTN;                ILLEGAL INITIAL CHARACTER
0056 14                 0157000 IVAL2:  INR     D;          INCORPORATE DIGIT
0057 CD1300             0158000         GETCH;              GET NEXT DIGIT
005A CD0000             0159000         CALL    QNUM
005D D25600             0160000         JNC     IVAL2;      STILL DIGIT STRING
0060 FE2E               0161000         CPI     '.';        WILL BE IGNORED
0062 C2AF00             0162000         JNZ     IVAL8;      CHECK FOR EXPONENT
0065 CD1300             0163000 IVAL3:  GETCH; ;            DIGITS AFTER DEC. PT.
0068 CD0000             0164000         CALL    QNUM
006B DAAF00             0165000         JC      IVAL8;      NON-DIGIT
006E CD0700             0166000         CALL    QMAX
0071 DA6500             0167000         JC      IVAL3;      NO ROOM, IGNORE
0074 CD0000             0168000         CALL    MUL10
0077 E60F               0169000 IVAL4:  ANI     0FH;        MASK OFF DIGIT
0079 15                 0170000         DCR     D;          MODIFY EXPONENT,DIGITS AFTER D.P.
007A 85                 0171000         ADD     L
007B 6F                 0172000         MOV     L,A
007C 7C                 0173000         MOV     A,H
007D CE00               0174000         ACI     0
007F 67                 0175000         MOV     H,A
0080 D26500             0176000         JNC     IVAL3;      NO OVERFLOW
0083 219A19             0177000         LXI     H,6554;     SET MAX
0086 14                 0178000         INR     D
0087 C36500             0179000         JMP     IVAL3
008A CD0700             0180000 IVAL5:  CALL    QMAX;       DIGITS TO LEFT OF D.P.
008D DA5600             0181000         JC      IVAL2;      NO MORE DIGIT ROOM
0090 CD0000             0182000         CALL    MUL10
0093 E60F               0183000 IVAL6:  ANI     0FH;        MASK OFF DIGIT
0095 85                 0184000         ADD     L
0096 6F                 0185000         MOV     L,A
0097 7C                 0186000         MOV     A,H
0098 CE00               0187000         ACI     0
009A 67                 0188000         MOV     H,A;        INCORPORATE DIGIT
009B D2A100             0189000         JNC     IVAL7;      NO OVERFLOW
009E 21FFFF             0190000         LXI     H,65535;    MAX
00A1 CD1300             0191000 IVAL7:  GETCH; ;            GET NEXT DIGIT/CHAR.
00A4 CD0000             0192000         CALL    QNUM
00A7 D28A00             0193000         JNC     IVAL5;      DIGIT
00AA FE2E               0194000         CPI     '.'
00AC CA6500             0195000         JZ      IVAL3;      DEC. PT
00AF FE45               0196000 IVAL8:  CPI     'E'
00B1 CAB600             0197000         JZ      IVAL9
00B4 FE65               0198000         CPI     'E';        LOWER CASE ALLOWED
00B6 EB                 0199000 IVAL9:  XCHG
00B7 37                 0200000         STC
00B8 3F                 0201000         CMC;                CLEAR ANY CARRY
00B9 CC3A01             0202000         CZ      REXP;       "E", READ EXPONENT
00BC 6F                 0203000         MOV     L,A;        EXIT CHAR.
00BD D4C400             0204000         CNC     UNFIX;      CONVERT FORMAT IF NO OVERFLOW YET
00C0 D0                 0205000         RNC;                NO OVERFLOW
00C1 3E80               0206000         MVI     A,80H;      OVERFLOW CODE
00C3 C9                 0207000         RTN
                        0208000 ;
                        0209000 ; "FIXED" POINT REPRESENTATION CONSISTS OF A 16 BIT
                        0210000 ; POSITIVE INTEGER (IN THE RANGE 0 TO 65535), AND A
                        0211000 ; 7 BIT OFFSET (BY 40H) INTEGER EXPONENT, WHICH
                        0212000 ; REPRESENTS A POWER OF TEN MULTIPLIER. THE EIGHTH
                        0213000 ; EXPONENT BIT REPRESENTS THE SIGN OF THE MANTISSA
                        0214000 ; THIS REPRESENTATION IS USED FOR INPUT/OUTPUT ONLY
```

```
                        0215000 ;
                        0216000 ; CONVERT "FIXED" FORMAT TO "REAL"
                        0217000 ; CARRY FOR INPUT OUT OF RANGE
                        0218000 ; F,D,E,H
00C4 C5E5               0219000 UNFIX:  SAVE   BC.L
00C6 F5                 0220000         PUSH   PSW
00C7 7C                 0221000         MOV    A,H
00C8 E680               0222000         ANI    80H
00CA 47                 0223000         MOV    B,A;      SIGN OF RESULT
00CB 7C                 0224000         MOV    A,H
00CC E67F               0225000         ANI    7FH
00CE D640               0226000         SUI    40H
00D0 4F                 0227000         MOV    C,A;      DECIMAL POINT
00D1 CD0000             0228000         CALL   FLOTD
00D4 7C                 0229000         MOV    A,H
00D5 B7                 0230000         ORA    A
00D6 CAFA00             0231000         JZ     UNFIX4;   ZERO VALUE
00D9 7A                 0232000         MOV    A,D
00DA B0                 0233000         ORA    B
00DB 57                 0234000         MOV    D,A;      INCORPORATE SIGN
00DC 79                 0235000         MOV    A,C
00DD B7                 0236000         ORA    A
0003                    0237000 @01     SET    .LVL
00DE CAFA00             0238000 UNFIX1: JZ     UNFIX4;   REDUCED TO REAL
00E1 FAF000             0239000         JM     UNFIX3;   NEGATIVE EXPONENT
00E4 CD0000             0240000         CALL   FMULT;    POSITIVE EXPONENT
00E7 0D                 0241000         DCR    C
00E8 D2DE00             0242000         JNC    UNFIX1;   IN RANGE
00EB F1                 0243000 UNFIX2: POP    PSW
00EC 37                 0244000         STC;             MARK OVERFLOW
00ED C3FC00             0245000         JMP    UNFIX5
0003                    0246000 .LVL    SET    @01
00F0 CD0000             0247000 UNFIX3: CALL   FDIVT;    NEG. EXPONENT
00F3 0C                 0248000         INR    C
00F4 D2DE00             0249000         JNC    UNFIX1;   CONTINUE
00F7 C3E800             0250000         JMP    UNFIX2;   UNDERFLOW
00FA F1                 0251000 UNFIX4: POP    PSW
00FB B7                 0252000         ORA    A;        RESET CARRY,NO OVERFLOW
00FC C169C1             0253000 UNFIX5: RELOAD BC.L
00FF C9                 0254000         RTN
                        0255000 ;
                        0256000 ; READ 2 DIGIT SIGNED DECIMAL EXPONENT
                        0257000 ; TO (A). RETURN EXIT CHARACTER IN (D)
                        0258000 ; A,F,D,E
0100 CD1300             0259000 R2DC:   GETCH; ;          GET CHAR.
0103 CD0000             0260000         CALL   QNUM
0106 DA2601             0261000         JC     R2DC3;     NOT DIGIT
0109 110000             0262000 R2DC1:  LXI    D,0
010C 15                 0263000 R2DC2:  DCR    D
010D 14                 0264000         INR    D
010E 37                 0265000         STC
010F C0                 0266000         RNZ;               OVERFLOW, 3 DIGITS ENTERED, 1ST
                                                                        NON-ZERO
0110 53                 0267000         MOV    D,E
0111 E60F               0268000         ANI    0FH
0113 5F                 0269000         MOV    E,A
0114 CD1300             0270000         GETCH;
0117 CD0000             0271000         CALL   QNUM
011A D20C01             0272000         JNC    R2DC2
011D F5                 0273000         PUSH   PSW
011E 7A                 0274000         MOV    A,D
011F 87                 0275000         ADD    A
0120 87                 0276000         ADD    A;         4*
0121 82                 0277000         ADD    D
0122 87                 0278000         ADD    A;         10*
0123 83                 0279000         ADD    E;         VALUE MOD 100
0124 D1                 0280000         POP    D
0125 C9                 0281000         RTN
0126 FE2B               0282000 R2DC3:  CPI    '+'
0128 CA0001             0283000         JZ     R2DC;      IGNORE UNARY +
012B FE2D               0284000         CPI    '-'
012D C23601             0285000         JNZ    R2DC4;     NOT UNARY -
0130 CD0001             0286000         CALL   R2DC;
0133 2F                 0287000         CMA
0134 3C                 0288000         INR    A
0135 C9                 0289000         RTN
0136 57                 0290000 R2DC4:  MOV    D,A
0137 3E00               0291000         MVI    A,0;       RETURN 0, NONE CAME
0139 C9                 0292000         RTN
                        0293000 ;
                        0294000 ; READ EXPONENT & COMBINE WITH "FIXED" VALUE
                        0295000 ; RETURN EXIT CHAR. IN (A)
                        0296000 ; A,F,D
013A D5                 0297000 REXP:   PUSH   D
013B CD0001             0298000         CALL   R2DC;      GET EXPONENT
013E DA4F01             0299000         JC     REXP1;     OVERFLOW
0141 D5                 0300000         PUSH   D;         SAVE EXIT CHAR
0142 57                 0301000         MOV    D,A;       EXPONENT
0143 7C                 0302000         MOV    A,H
0144 E680               0303000         ANI    80H
0146 5F                 0304000         MOV    E,A;       SIGN
0147 7C                 0305000         MOV    A,H
0148 E67F               0306000         ANI    7FH;       EXPONENT ALONE
014A 82                 0307000         ADD    D
014B F25201             0308000         JP     REXP2;     NO OVERFLOW
0002                    0309000 @01     SET    .LVL
014E F1                 0310000         POP    PSW;       EXIT CHAR
014F D1                 0311000 REXP1:  POP    D
0150 37                 0312000         STC;              SIGNAL OVERFLOW
0151 C9                 0313000         RTN
0002                    0314000 .LVL    SET    @01
0152 B3                 0315000 REXP2:  ORA    E;         ORIGINAL SIGN
0153 67                 0316000         MOV    H,A;       RESULTANT EXPONENT
0154 F1                 0317000         POP    PSW;       RESTORE EXIT CHAR.
0155 D1                 0318000         POP    D;         RESTORE MANTISSA
0156 B7                 0319000         ORA    A;         CLEAR CARRY, NO OVERFLOW
0157 C9                 0320000         RTN
                        0321000 ;
0000                    0322000         END.   0
PASS 2 ERRORS = 0
.LVL    RED     00000H    @01     RED     00002H    DEBLK   IDR     00015H
FDIVT   EXT     00002H    FLOTD   EXT     00003H    FMULT   EXT     00004H
FSIGN   MACRO             GETCH   MACRO             IVAL    IDR     0001EH
IVAL1   IDR     0004AH    IVAL2   IDR     00056H    IVAL3   IDR     00065H
IVAL4   IDR     00077H    IVAL5   IDR     0008AH    IVAL6   IDR     00093H
IVAL7   IDR     000A1H    IVAL8   IDR     000AFH    IVAL9   IDR     000B6H
IVALC   IDR     00021H    JBC     IDR     00013H    MUL10   EXT     00005H
QMAX    IDR     00007H    QNUM    IDR     00000H    R2DC    IDR     00100H
R2DC1   IDR     00109H    R2DC2   IDR     0010CH    R2DC3   IDR     00126H
R2DC4   IDR     00136H    RELOAD  MACRO             REXP    IDR     0013AH
REXP1   IDR     0014FH    REXP2   IDR     00152H    RTN     MACRO
SAVE    MACRO             UNFIX   IDR     000C4H    UNFIX1  IDR     000DEH
UNFIX2  IDR     000EBH    UNFIX3  IDR     000F0H    UNFIX4  IDR     000FAH
UNFIX5  IDR     000FCH

##########################################
########    SEGMENT  FLTOUT      ##########
##########################################

                        0001000 ; COPYRIGHT YALE UNIVERSITY
                        0002000 ;
                        0003000 ; BY C.B. FALCONER                (203) 436-2603
                        0004000 ;    E.A. LEE
                        0005000 ;                  ROOM 6016CB
                        0006000 ;                  YALE SCHOOL OF MEDICINE
                        0007000 ;                  789 HOWARD AVENUE
                        0008000 ;                  NEW HAVEN, CONN 06504
                        0009000 ;
                        0010000 ;                        *************************
                        0011000         RSEG;            * FLOATING POINT OUTPUT *
                        0012000 ;                        *************************
                        0013000 ;
                        0014000 ; EXTERNAL CALLS REQUIRED
                        0015000         EXTRN  DERC,DTEN; IN INTARITH
                        0016000         EXTRN  FDIVT,FLOAD,FMULT; IN FLTARITH
                        0017000 ;
                        0018000 ; EXTERNAL CONNECTORS TO LIST AND CONSOLE DRIVERS
                        0019000         EXTRN  LOUT,COUT; UNDEFINED IN SYSTEM, OUTPUT (C),
                        0020000 ;                                        SET (A):=(C)
                        0021000 ; ENTRY POINTS ALLOWED
                        0022000         ENTRY  LFLT,TFLT,OFLT,FMAT
                        0023000 ;
                        0024000 ; ENTRY POINTS TO UTILITY ROUTINES
```

```
                0025000          ENTRY  EXDG,OTCBK,OTCCL
                0026000          ENTRY  OPT,ONEG,ODZS
                0027000 ;
                0028000 ; **********************
                0029000 ; * MACRO DEFINITIONS *
                0030000 ; **********************
                0031000 ;
                0032000 ; LOAD (REG) FROM TOS & LEAVE ON STACK   (REG)
                0033000 LTOS     MACRO  REG
                0034000          POP    REG
                0035000          PUSH   REG
                0036000          ENDM
                0037000 ;
                0038000 ; "RETURN" AND CHECK STACK LEVEL ZERO
                0039000 RTN      MACRO
                0040000          IF     .LVL
                0041000          ERROR  "0"+.LVL
                0042000 .LVL     SET    0
                0043000          ENDIF
                0044000          RET
                0045000          ENDM
                0046000 ;
                0047000 ; TRADE (A) DIGITS, LEAVE LSB IN CARRY (A,F)
                0048000 TDIG     MACRO
                0049000          RLC
                0050000          RLC
                0051000          RLC
                0052000          RLC
                0053000          ENDM
                0054000 ;
                0055000 ; ***********************************
                0056000 ; *        UTILITY  ROUTINES        *
                0057000 ; ***********************************
                0058000 ;
                0059000 ; OUTPUT BLANK TO CONSOLE/LISTER
                0060000 ; USE LISTER IF (A) SIGN BIT=1;ELSE CONSOLE
                0061000 ; A,F,C
0000 0E20       0062000 OTCBK:   MVI    C,' '
                0063000 ;
                0064000 ; OUTPUT A CHARACTER
                0065000 ; USE LISTER IF (A) SIGN BIT=1;ELSE CONSOLE
                0066000 ; A,F,C
0002 07         0067000 OTCCL:   RLC
0003 DA0000     0068000          JC     LOUT
0006 C30000     0069000          JMP    COUT;     ON CONSOLE
                0070000 ;
                0071000 ; OUTPUT (A) BLANKS
                0072000 ; USE LISTER IF (A) SIGN BIT=1;ELSE CONSOLE
                0073000 ; A,F
0009 C5         0074000 NBLK:    PUSH   B
000A 47         0075000          MOV    B,A
000B C31400     0076000          JMP    NBLK2;    CHECK FOR ZERO
000E 78         0077000 NBLK1:   MOV    A,B
000F CD0000     0078000          CALL   OTCBK
0012 05         0079000          DCR    B
0013 78         0080000          MOV    A,B
0014 E67F       0081000 NBLK2:   ANI    7FH
0016 C20E00     0082000          JNZ    NBLK1
0019 C1         0083000          POP    B
001A C9         0084000          RTN
                0085000 ;
                0086000 ; OUTPUT A DEC. PT.
                0087000 ; USE LISTER IF (A) SIGN BIT=1;ELSE CONSOLE
                0088000 ; A,F,C
001B 0E2E       0089000 OPT:     MVI    C,'.'
001D C30200     0090000          JMP    OTCCL
                0091000 ;
                0092000 ; OUTPUT "-"
                0093000 ; USE LISTER IF (A) SIGN BIT=1;ELSE CONSOLE
                0094000 ; A,F,C
0020 0E2D       0095000 ONEG:    MVI    C,'-'
0022 C30200     0096000          JMP    OTCCL
                0097000 ;
                0098000 ; OUTPUT (HL) IN DECIMAL, SUPPRESS LEADING ZEROS
                0099000 ; USE LISTER IF (A) SIGN BIT=1;ELSE CONSOLE
0025 C5         0100000 ODZS:    PUSH   B
0026 0605       0101000          MVI    B,5
0028 F5         0102000          PUSH   PSW;      PRESERVE
0029 CD4300     0103000 ODZS2:   CALL   EXDG;     EXTRACT A DIGIT
002C C23700     0104000          JNZ    ODZS4;    NON-ZERO, END SUPPRESS
002F 05         0105000          DCR    B
0030 C22900     0106000          JNZ    ODZS2;    CONTINUE SUPPRESSION
0033 04         0107000          INR    B;        RE-EXTRACT FINAL ZERO AND OUTPUT
0034 CD4300     0108000 ODZS3:   CALL   EXDG;     GET NEXT DIGIT
0037 F1F5       0109000 ODZS4:   LTOS   PSW
0039 CD0200     0110000          CALL   OTCCL;     OUTPUT TO  CONSOLE OR LISTER
003C 05         0111000          DCR    B
003D C23400     0112000          JNZ    ODZS3
0040 F1         0113000          POP    PSW
0041 C1         0114000          POP    B
0042 C9         0115000          RTN
                0116000 ;
                0117000 ; EXTRACT A DEC. DIGIT, 10^((B)-1), FROM (HL)
                0118000 ; ASCII DIGIT RETURNED IN (C) & (A)
                0119000 ; WITH ZERO FLAG FOR DIGIT=ZERO
                0120000 ; A,F,C
0043 E5         0121000 EXDG:    PUSH   H
0044 C5         0122000          PUSH   B
0045 CD0000     0123000 EXDG1:   CALL   DTEN
0048 05         0124000          DCR    B
0049 C24500     0125000          JNZ    EXDG1
004C C630       0126000          ADI    '0'
004E FE30       0127000          CPI    '0'
0050 C1         0128000          POP    B
0051 E1         0129000          POP    H
0052 4F         0130000          MOV    C,A
0053 C9         0131000          RTN
                0132000 ;
                0133000 ; * END UTILITY ROUTINES *
                0134000 ; ************************
                0135000 ;
                0136000 ; "FIXED" POINT REPRESENTATION CONSISTS OF A 16 BIT
                0137000 ; POSITIVE INTEGER (IN THE RANGE 0 TO 65535), AND A
                0138000 ; 7 BIT OFFSET (BY 40H) INTEGER EXPONENT, WHICH
                0139000 ; REPRESENTS A POWER OF TEN MULTIPLIER. THE EIGHTH
                0140000 ; EXPONENT BIT REPRESENTS THE SIGN OF THE MANTISSA
                0141000 ; THIS REPRESENTATION IS USED FOR INPUT/OUTPUT ONLY
                0142000 ;
                0143000 ; CONVERT "REAL" FORMAT TO "FIXED" FORMAT
                0144000 ; A,F,D,E,H
0054 7C         0145000 FIX:     MOV    A,H
0055 B7         0146000          ORA    A
0056 C25F00     0147000          JNZ    FIX1;     VALUE NOT ZERO
0059 2640       0148000          MVI    H,40H;    00000.
005B 110000     0149000          LXI    D,0
005E C9         0150000          RTN
005F FE91       0151000 FIX1:    CPI    91H
0061 C5         0152000          PUSH   B
0062 0640       0153000          MVI    B,40H;    DECIMAL EXPONENT
0001            0154000 @01      SET    .LVL
0064 D28400     0155000          JNC    FIX3;     >65535, INTEGER
0067 FE8E       0156000          CPI    8EH
0069 D28900     0157000          JNC    FIX6;     TREAT AS LEFT SHIFTED INTEGER
006C CD0000     0158000 FIX2:    CALL   FMULT;    <32768
006F 05         0159000          DCR    B
0070 7C         0160000          MOV    A,H
0071 D690       0161000          SUI    90H
0073 DA6C00     0162000          JC     FIX2;     STILL FRACTIONAL SEGMENT
0076 C28400     0163000          JNZ    FIX3;     NOT NOW INTEGER
0079 7A         0164000 FIX3:    MOV    A,D
007A E680       0165000          ANI    80H;      EXTRACT SIGN
007C B0         0166000          ORA    B
007D 67         0167000          MOV    H,A
007E 7A         0168000          MOV    A,D
007F F680       0169000          ORI    80H;      SET MSBIT, RANGE 32768/65535
0081 57         0170000          MOV    D,A
0082 C1         0171000          POP    B
0083 C9         0172000          RTN
0001            0173000 .LVL     SET    @01
0084 CD0000     0174000 FIX5:    CALL   FDIVT;    INTEGER > 65535
0087 04         0175000          INR    B
0088 7C         0176000          MOV    A,H
```

```
0089 D690      0177000 FIX6:   SUI     90H
008B CA7900    0178000         JZ      FIX3;       NOW INTEGER REPRESENTATION
008E D28400    0179000         JNC     FIX5
0091 67        0180000         MOV     H,A;        RANGE -1 TO -4
0092 7A        0181000         MOV     A,D
0093 E680      0182000         ANI     80H;        RESULT SIGN
0095 B0        0183000         ORA     B
0096 47        0184000         MOV     B,A
0097 7A        0185000         MOV     A,D
0098 F680      0186000         ORI     80H
009A 57        0187000         MOV     D,A
009B B7        0188000 FIX7:   ORA     A;          RESET CARRY
009C CD0000    0189000         CALL    DERC
009F 24        0190000         INR     H
00A0 C29B00    0191000         JNZ     FIX7;       SHIFT OFF FRACTIONAL SEGMENT
00A3 D2A700    0192000         JNC     FIX8;       NO ROUNDING NEEDED
00A6 13        0193000         INX     D
00A7 60        0194000 FIX8:   MOV     H,B
00AB C1        0195000         POP     B
00A9 B7        0196000         ORA     A;          RESET CARRY
00AA C9        0197000         RTN
               0198000 ;
               0199000 ; ***** OUTPUT ROUTINES *******
               0200000 ;
               0201000 ; OUTPUT (DE.H) IN "FIXED" FORM
               0202000 ; SUPPRESS LEADING ZEROES
               0203000 ;IF A>=0, TO CONSOLE, IF A<0, TO LISTER
               0204000 ; A,F,B,C,D,E,H,L
00AB F5        0205000 OFIX:   PUSH    PSW
00AC 7C        0206000         MOV     A,H
00AD E680      0207000         ANI     80H
00AF F2B700    0208000         JP      OFIX1;      POSITIVE
00B2 F1F5      0209000         LTOS    PSW;        OUTPUT "-" SIGN
00B4 CD2000    0210000         CALL    ONEG;       SEND A '-' SIGN
00B7 7C        0211000 OFIX1:  MOV     A,H
00B8 E67F      0212000         ANI     7FH;        REMOVE SIGN
00BA D63A      0213000         SUI     40H-6;      SIGNED DEC. EXPONENT
00BC 67        0214000         MOV     H,A
00BD EB        0215000         XCHG;               VALUE IN HL, EXP IN D
00BE 0606      0216000         MVI     B,6;        1ST DIGIT
00C0 05        0217000 OFIX2:  DCR     B
0001           0218000 @01     SET     .LVL
00C1 CA4901    0219000         JZ      OFIXA;      DONE, ALL DIGITS 0
00C4 15        0220000         DCR     D
00C5 CA0C01    0221000         JZ      OFIX7;      DEC. PT HERE
00C8 FA0C01    0222000         JM      OFIX7;      .XXXXE-XX
00CB CD4300    0223000         CALL    EXDG
00CE CAC000    0224000         JZ      OFIX2;      SUPPRESS A ZERO
00D1 F1F5      0225000 OFIX3:  LTOS    PSW
00D3 CD0200    0226000         CALL    OTCCL;      LIST A DIGIT
00D6 15        0227000         DCR     D
00D7 C2DF00    0228000         JNZ     OFIX4
00DA F1F5      0229000         LTOS    PSW
00DC CD1B00    0230000         CALL    OPT;        DEC. PT. HERE
00DF 05        0231000 OFIX4:  DCR     B
00E0 CAE900    0232000         JZ      OFIX5;      ALL DIGITS LISTED
00E3 CD4300    0233000         CALL    EXDG
00E6 C3D100    0234000         JMP     OFIX3;      LIST NEXT DIGIT
00E9 AF        0235000 OFIX5:  XRA     A
00EA 92        0236000         SUB     D
00EB F24E01    0237000         JP      OFIXB;      NOT XXXX.E+XX
00EE C603      0238000         ADI     3
00F0 FAFE00    0239000         JM      OFIX6;      >9999000
00F3 0E30      0240000         MVI     C,"0"
00F5 F1F5      0241000         LTOS    PSW
00F7 CD0200    0242000         CALL    OTCCL
00FA 15        0243000         DCR     D
00FB C3E900    0244000         JMP     OFIX5
00FE 0E45      0245000 OFIX6:  MVI     C,'E'
0100 F1F5      0246000         LTOS    PSW
0102 CD0200    0247000         CALL    OTCCL
0105 6A        0248000         MOV     L,D
0106 2600      0249000         MVI     H,0
0108 F1        0250000         POP     PSW
0109 C32500    0251000         JMP     ODZS;       LIST EXPONENT
0001           0252000 .LVL    SET     @01

010C F1F5      0253000 OFIX7:  LTOS    PSW
010E CD1B00    0254000         CALL    OPT;        .XXXXE-XX
0111 7A        0255000 OFIX8:  MOV     A,D
0112 B7        0256000         ORA     A
0113 CA2601    0257000         JZ      OFIX9;      ZERO EXPONENT
0116 C603      0258000         ADI     3
0118 FA2601    0259000         JM      OFIX9;      RANGE -1 TO -3, INSERT 0'S
011B 0E30      0260000         MVI     C,"0"
011D F1F5      0261000         LTOS    PSW
011F CD0200    0262000         CALL    OTCCL
0122 14        0263000         INR     D
0123 C31101    0264000         JMP     OFIX8;      CHECK FOR MORE 0'S
0126 CD4300    0265000 OFIX9:  CALL    EXDG
0129 F1F5      0266000         LTOS    PSW
012B CD0200    0267000         CALL    OTCCL
012E 05        0268000         DCR     B
012F C22601    0269000         JNZ     OFIX9
0132 AF        0270000         XRA     A
0133 67        0271000         MOV     H,A
0134 92        0272000         SUB     D
0135 6F        0273000         MOV     L,A
0136 CA4E01    0274000         JZ      OFIXB;      IGNORE ZERO EXPONENT
0139 0E45      0275000         MVI     C,'E'
013B F1F5      0276000         LTOS    PSW
013D CD0200    0277000         CALL    OTCCL
0140 F1F5      0278000         LTOS    PSW
0142 CD2000    0279000         CALL    ONEG
0145 F1        0280000         POP     PSW
0146 C32500    0281000         JMP     ODZS;       LIST EXPONENT & EXIT
0001           0282000 .LVL    SET     @01
0149 F1F5      0283000 OFIXA:  LTOS    PSW
014B CD0200    0284000         CALL    OTCCL;      LIST A ZERO
014E F1        0285000 OFIXB:  POP     PSW
014F C9        0286000         RTN
               0287000 ;
               0288000 ; OUTPUT "REAL" (DE.H) TO LISTER
0150 F5        0289000 LFLT:   PUSH    PSW
0151 3EFF      0290000         MVI     A,-1;       IDENTIFY AS LISTER OUTPUT
0153 CD6001    0291000         CALL    OFLT
0156 F1        0292000         POP     PSW
0157 C9        0293000         RTN
               0294000 ;
               0295000 ; OUTPUT "REAL" (DE.H) TO CONSOLE
015B F5        0296000 TFLT:   PUSH    PSW
0159 3E00      0297000         MVI     A,0;        IDENTIFY AS CONSOLE OUTPUT
015B CD6001    0298000         CALL    OFLT
015E F1        0299000         POP     PSW
015F C9        0300000         RTN
               0301000 ;
               0302000 ; OUTPUT "REAL"
               0303000 ; IF A<0, TO LISTER, IF A>=0, TO CONSOLE
               0304000 ; A
0160 C5        0305000 OFLT:   PUSH    B
0161 D5        0306000         PUSH    D
0162 E5        0307000         PUSH    H
0163 F5        0308000         PUSH    PSW
0164 CD5400    0309000         CALL    FIX
0167 F1        0310000         POP     PSW;        GET DESTINATION
0168 CDAB00    0311000         CALL    OFIX
016B E1        0312000         POP     H
016C D1        0313000         POP     D
016D C1        0314000         POP     B
016E C9        0315000         RTN
               0316000 ;
               0317000 ; OUTPUT (DE.H) WITH FORMAT SPECIFICATION (A)
               0318000 ; (A) BIT FIELD     MEANING
               0319000 ; =============     =======
               0320000 ;  0:1 (LH BIT)     1=TO LISTER, 0=TO CONSOLE
               0321000 ;  1:3              PLACES TO LEFT OF D.P.
               0322000 ;  4:1              USE FREE FORMAT, IGNORE PLACES SPEC.
               0323000 ;  5:3 (3 RH BITS) PLACES TO RT. OF D.P.
016F F5        0324000 FMAT:   PUSH    PSW
0170 C5        0325000         PUSH    B
0171 D5        0326000         PUSH    D
0172 E5        0327000         PUSH    H
0173 F5        0328000         PUSH    PSW;        SAVE PLACES
```

```
0174 CD5400      0329000          CALL    FIX
0177 F1F5        0330000          LTOS    PSW
0179 CD8801      0331000          CALL    AJFX;       ADJUST ON RIGHT OF D.P.
017C CD8701      0332000          CALL    TPSN;       POSITION THE FIELD
017F F1          0333000          POP     PSW;        FOR LIST/CONSOLE DESTINATION
0180 CDAB00      0334000          CALL    OFIX;       OUTPUT THE DATA
0183 E1          0335000          POP     H;          AND RESTORE REGISTERS
0184 D1          0336000          POP     D
0185 C1          0337000          POP     B
0186 F1          0338000          POP     PSW
0187 C9          0339000          RTN
                 0340000 ;
                 0341000 ; ADJUST "FIXED" FORMAT FOR (A) DIGITS AFTER DEC. PT.
                 0342000 ; MAX DIGITS=7, 8 BIT FOR FLOATING FORMAT
                 0343000 ; ROUND THE RESULT
                 0344000 ; D,E,H
0188 F5          0345000 AJFX:    PUSH    PSW
0189 C5          0346000          PUSH    B
018A E5          0347000          PUSH    H;          SAVE BC.L
018B E60F        0348000          ANI     15
018D FE08        0349000          CPI     8
018F D2B201      0350000          JNC     AJFX4;      FLOATING FORMAT, NO ADJUST
0192 E607        0351000          ANI     7;          7 DIGITS MAX.
0194 4F          0352000          MOV     C,A;        DIGITS REQUIRED
0195 7C          0353000 AJFX1:   MOV     A,H
0196 E67F        0354000          ANI     7FH;        REMOVE SIGN
0198 D640        0355000          SUI     40H
019A 81          0356000          ADD     C
019B F2B201      0357000          JP      AJFX4;      NO EXCESS FRACTIONAL SEGMENT
019E 47          0358000          MOV     B,A
019F EB          0359000          XCHG
01A0 CD0000      0360000 AJFX2:   CALL    DTEN;       REMOVE A DIGIT
01A3 14          0361000          INR     D;          ADJUST DEC. EXPONENT
01A4 04          0362000          INR     B
01A5 FAA001      0363000          JM      AJFX2;      REMOVE MORE
01A8 FE05        0364000          CPI     5
01AA DAAE01      0365000          JC      AJFX3;      NO ROUNDING
01AD 23          0366000          INX     H
01AE EB          0367000 AJFX3:   XCHG
01AF C39501      0368000          JMP     AJFX1;      IN CASE ROUNDING ADDED DIGIT
01B2 C1          0369000 AJFX4:   POP     B;          RELOAD BC.L
01B3 69          0370000          MOV     L,C
01B4 C1          0371000          POP     B
01B5 F1          0372000          POP     PSW
01B6 C9          0373000          RTN
                 0374000 ;
                 0375000 ; CONTROL LEADING BLANKS VIA BITS 1:3 OF (A)
                 0376000 ; BIT 4:1 SPECIFIES FREE FORMAT
                 0377000 ; ELSE OUTPUT BLANKS REQUIRED TO PLACE THE DP
                 0378000 ; AT THE FIELD POSITION TO RIGHT OF STARTING PT.
01B7 F5          0379000 TPSN:    PUSH    PSW
01B8 C5          0380000          PUSH    B
01B9 F5          0381000          PUSH    PSW
01BA E608        0382000          ANI     8
01BC C20002      0383000          JNZ     TPSN9;      FREE FORMAT
01BF F1          0384000          POP     PSW
01C0 E6F0        0385000          ANI     0F0H
01C2 47          0386000          MOV     B,A
01C3 E670        0387000          ANI     70H
01C5 07070707    0388000          TDIG
01C9 4F          0389000          MOV     C,A;        COUNT OF PLACES NEEDED
01CA C5          0390000          PUSH    B
01CB 7C          0391000          MOV     A,H
01CC E67F        0392000          ANI     7FH
01CE D640        0393000          SUI     40H;        FORM DP LOCATION WRT RIGHT DIGIT
01D0 FADA01      0394000          JM      TPSN5;      FRACT. SEGMENT
01D3 3C          0395000          INR     A
01D4 B8          0396000          CMP     B
01D5 D20002      0397000          JNC     TPSN9;      TOO LARGE, USE FREE FORMAT
01D8 3E00        0398000          MVI     A,0;        USE VALUE ZERO?
01DA 47          0399000 TPSN5:   MOV     B,A;        -VE VALUE
01DB CD0402      0400000          CALL    NDIG;       COUNT OF SIG. DIGITS IN VALUE
01DE 80          0401000          ADD     B;          TO LEFT OF DP
01DF F2E401      0402000          JP      TPSN6;      THERE ARE SOME
01E2 3E00        0403000          MVI     A,0;        ONE + ZEROES AFTER DP
01E4 C1          0404000 TPSN6:   POP     B
01E5 2F          0405000          CMA
01E6 3C          0406000          INR     A
01E7 81          0407000          ADD     C;          SPACES REQUIRED
01E8 F2ED01      0408000          JP      TPSN8
01EB 3E00        0409000          MVI     A,0;        CAN'T ALLOW -VE COUNT
01ED 4F          0410000 TPSN8:   MOV     C,A
01EE 7C          0411000          MOV     A,H
01EF B7          0412000          ORA     A
01F0 F2F801      0413000          JP      TPSN7;      +VE VALUE
01F3 0D          0414000          DCR     C;          ALLOW FOR -SIGN
01F4 F2F801      0415000          JP      TPSN7;      ROOM
01F7 0C          0416000          INR     C;          NO ROOM, MOVE ALL RIGHT
01F8 78          0417000 TPSN7:   MOV     A,B
01F9 E680        0418000          ANI     80H;        FILE FLAG
01FB B1          0419000          ORA     C
01FC CD0900      0420000          CALL    NBLK;       SPACE AS REQUIRED
01FF F5          0421000          PUSH    PSW
0200 F1          0422000 TPSN9:   POP     PSW
0201 C1          0423000          POP     B
0202 F1          0424000          POP     PSW
0203 C9          0425000          RTN
                 0426000 ;
                 0427000 ; RETURN COUNT OF SIGNIFICANT DIGITS IN (DE)
                 0428000 ; TREATING (DE) AS DECIMAL INTEGER WITH LEADING
                 0429000 ; ZEROES SUPPRESSED.  RETURN 1 FOR VALUE 00000
                 0430000 ; A,F
0204 C5          0431000 NDIG:    PUSH    B
0205 D5          0432000          PUSH    D
0206 EB          0433000          XCHG
0207 0600        0434000          MVI     B,0
0209 04          0435000 NDIG1:   INR     B
020A CD0000      0436000          CALL    DTEN
020D 7C          0437000          MOV     A,H
020E B5          0438000          ORA     L
020F C20902      0439000          JNZ     NDIG1;      MORE DIGITS LEFT
0212 78          0440000          MOV     A,B
0213 EB          0441000          XCHG
0214 D1          0442000          POP     D
0215 C1          0443000          POP     B
0216 C9          0444000          RTN
                 0445000 ;
0000             0446000          END.    0
PASS 2 ERRORS = 0
.LVL    RED     00000H   @01     RED     00001H   AJFX    IDR     001B8H
AJFX1   IDR     00195H   AJFX2   IDR     001A0H   AJFX3   IDR     001AEH
AJFX4   IDR     001B2H   CDUT    EXT     00008H   DERC    EXT     00002H
DTEN    EXT     00003H   EXDG    IDR     00043H   EXDG1   IDR     00045H
FDIVT   EXT     00004H   FIX     IDR     00054H   FIX1    IDR     0005FH
FIX2    IDR     0006CH   FIX3    IDR     00079H   FIX5    IDR     00084H
FIX6    IDR     00089H   FIX7    IDR     0009BH   FIX8    IDR     000A7H
FLOAD   EXT     00005H   FMAT    IDR     0016FH   FMULT   EXT     00006H
LFLT    IDR     00150H   LDUT    EXT     00007H   LTOS    MACRO
NBLK    IDR     00009H   NBLK1   IDR     0000EH   NBLK2   IDR     00014H
NDIG    IDR     00204H   NDIG1   IDR     00209H   ODZS    IDR     00025H
ODZS2   IDR     00029H   ODZS3   IDR     00034H   ODZS4   IDR     00037H
OFIX    IDR     000ABH   OFIX1   IDR     000B7H   OFIX2   IDR     000C0H
OFIX3   IDR     000D1H   OFIX4   IDR     000DFH   OFIX5   IDR     000E9H
OFIX6   IDR     000FEH   OFIX7   IDR     0010CH   OFIX8   IDR     00111H
OFIX9   IDR     00126H   OFIXA   IDR     00149H   OFIXB   IDR     0014EH
OFLT    IDR     00160H   ONEG    IDR     00020H   OPT     IDR     0001BH
OTCBK   IDR     00000H   OTCCL   IDR     00002H   RTN     MACRO
TDIG    MACRO            TFLT    IDR     00158H   TPSN    IDR     001B7H
TPSN5   IDR     001DAH   TPSN6   IDR     001E4H   TPSN7   IDR     001F8H
TPSN8   IDR     001EDH   TPSN9   IDR     00200H

*********************************************
********    SEGMENT FUNCTION    ***********
*********************************************
                 0001000 ;
                 0002000 ;                          ******************************
                 0003000          RSEG;             * USEFUL FLOATING FUNCTIONS *
                 0004000 ;                          ******************************
                 0005000 ;
                 0006000 ; EXTERNAL ROUTINES REQUIRED, SEE FLTARITH
                 0007000 ; ****************************************
                 0008000 ;
```

```
0009000 ; EXTERNAL ARITHMETIC ERROR TRAP
0010000         EXTRN   AERC
0011000 ;
0012000 ; EXTERNAL FLOATING ARITHMETIC
0013000         EXTRN   FADD,FDIV,FDIVR
0014000         EXTRN   FINT,FIXR
0015000         EXTRN   FMUL,FRCIP,FSUBR
0016000 ;
0017000 ; EXTERNAL FORMAT CONVERSION
0018000         EXTRN   FLOT,FLOTA,FLOTD
0019000 ;
0020000 ; EXTERNAL TESTS & MANIPULATION
0021000         EXTRN   FXCHG,FCMP
0022000 ;
0023000 ; EXTERNAL MEMORY ACCESS
0024000         EXTRN   FLOAD,LFBS
0025000 ;
0026000 ; ENTRY POINTS ALLOWED
0027000 ; ********************
0028000 ;
0029000 ; FUNCTIONS
0030000         ENTRY   FRACT,FMOD,POLY
0031000         ENTRY   LOG2,LOGB,EXP2,EXPX
0032000 ;
0033000 ; LOGICAL OPERATORS
0034000         ENTRY   .OR.,.AND.,.XOR.
0035000 ;
0036000 ; RELATIONAL OPERATORS
0037000         ENTRY   .EQU.,.NE.,.LE.,.GT.
0038000         ENTRY   .LT.,.GE.,.GG.
0039000 ;
0040000 ; *********************
0041000 ; * MACRO DEFINITIONS *
0042000 ; *********************
0043000 ;
0044000 ; LOAD (& POP) REAL (REG) FROM TOS, STORED BY SFTS MACRO
0045000 LFTS    MACRO   REG
0046000 BC.L    EQU     B
0047000 DE.H    EQU     D
0048000         POP     REG
0049000         MOV     5-REG/2,REG
0050000         POP     REG
0051000         IF      REG*(REG-D)
0052000         ERROR   "R"
0053000         ENDIF
0054000         ENDM
0055000 ;
0056000 ; MOVE OPERATION ON REGISTER PAIR B,D, OR H
0057000 MOVD    MACRO   R1,R2                                   0000 C5E5
0058000         IF      ((R1-D)*(R1-H)*R1) OR ((R2-D)*(R2-H)*R2) 0002 D5E5
0059000         ERROR   "R"                                     0004 CD0000
0060000         ENDIF                                           0007 C16BC1
0061000         MOV     R1,R2                                   000A CD0000
0062000         MOV     R1+1,R2+1                               000D C169C1
0063000         ENDM                                            0010 C9
0064000 ;
0065000 ; MOVE FLOATING OPERAND FROM REG1 TO REG2
0066000 MOVF    MACRO   R2,R1
0067000 BC.L    EQU     B
0068000 DE.H    EQU     D
0069000         IF      (R1*(R1-D)) OR (R2*(R2-D))              0011 CD0000
0070000         ERROR   "R"                                     0014 D8
0071000         ENDIF                                           0015 CD0000
0072000         MOV     R2,R1                                   0018 C30000
0073000         MOV     R2+1,R1+1
0074000         MOV     5-R2/2,5-R1/2
0075000         ENDM
0076000 ;
0077000 ; RELOAD (BC.L), STORED BY PUSH B, PUSH H SEQUENCE      001B C5E5
0078000 RELOAD  MACRO   REG                                     001D D5E5
0079000 BC.L    EQU     B                                       001F 78
0080000         IF      REG-B                                   0020 4H
0081000         ERROR   'R'                                     0021 95
0082000         DB      0,0,0                                   0022 000000
0083000         ENDIF                                           0025
0084000         IFZ     REG-B                                   0025 DA4700
0085000         POP     B
0086000         MOV     L,C
0087000         POP     B
0088000         ENDIF
0089000         ENDM
0090000 ;
0091000 ; "RETURN" AND CHECK STACK LEVEL ZERO
0092000 RTN     MACRO
0093000         IF      .LVL
0094000         ERROR   "0"+.LVL
0095000 .LVL    SET     0
0096000         ENDIF
0097000         RET
0098000         ENDM
0099000 ;
0100000 ; SAVE (BC.L), TO BE RESTORED BY RELOAD BC.L LATER
0101000 SAVE    MACRO   REG
0102000 BC.L    EQU     B
0103000         IF      REG-B
0104000         ERROR   'R'
0105000         DB      0,0
0106000         ENDIF
0107000         IFZ     REG-B
0108000         PUSH    B
0109000         PUSH    H
0110000         ENDIF
0111000         ENDM
0112000 ;
0113000 ; STORE REAL VALUE ON TOP OF STACK; NOTE SFTS B AFFECTS (A)
0114000 SFTS    MACRO   R
0115000 BC.L    EQU     B
0116000 DE.H    EQU     D
0117000         IF      R*(R-D)
0118000         ERROR   "R"
0119000         ENDIF
0120000         IFZ     R
0121000         MOV     A,L
0122000         ENDIF
0123000         PUSH    R
0124000         PUSH    PSW-R
0125000         ENDM
0126000 ;
0127000 ; ******************
0128000 ; * START THE CODE *
0129000 ; ******************
0130000 ;
0131000 ; EXTRACT FRACTIONAL PART OF (DE.H)
0132000 ; A,F,D,E,H
0133000 FRACT:  SAVE    BC.L
0134000         SFTS    D
0135000         CALL    FINT
0136000         LFTS    B;          ORIG. VALUE TO (BC.L)
0137000         CALL    FSUBR;      REMOVE INTEGER PORTION
0138000         RELOAD  BC.L
0139000         RTN
0140000 ;
0141000 ; CONVERT (DE.H) AND (BC.L) TO ROUNDED INTEGERS
0142000 ; IN (BC) AND (DE) RESPECTIVELY
0143000 ; A,F,B,C,D,E,H,L
0144000 FIXRT:  CALL    FIXR
0145000         RC;                 OVERFLOW
0146000         CALL    FXCHG
0147000         JMP     FIXR
0148000 ;
0149000 ; MODULO ARITHMETIC
0150000 ; (DE.H):=(BC.L) MODULO (DE.H)
0151000 ; SYSTEM TRAP FOR (DE.H)=0
0152000 ; A,F,D,E,H
0153000 FMOD:   SAVE    BC.L
0154000         SFTS    D
0155000         MOV     A,B
0156000         XRA     D;          COMPARE SIGNS
0157000         PUSH    PSW;        SAVE FOR EXIT
0158000         CALL    FDIVR;      BC.L/DE.H
0159000 @01     SET      LVL
0160000         JC      FMOD2;      OVERFLOW
```

```
002B CD0000      0161000          CALL    FINT;       INTEGER (BC.L/DE.H)
002B DA4700      0162000          JC      FMOD2;      OVERFLOW
002E F1          0163000          POP     PSW
002F F23A00      0164000          JP      FMOD1;      SIGNS SAME
0032 010080      0165000          LXI     B,8000H
0035 2E81        0166000          MVI     L,81H;      -1.000
0037 CD0000      0167000          CALL    FADD;       CORRECT
003A C16BC1      0168000 FMOD1:   LFTS    B;          ORIGINAL DE.H
003D CD0000      0169000          CALL    FMUL;       DE.H * INTEGER (BC.L/DE.H)
0040 C169C1      0170000          RELOAD  BC.L
0043 D20000      0171000          JNC     FSUBR;      BC.L-DE.H * INTEGER(BC.L/DE.H)
0046 C9          0172000          RTN;                FMUL OVERFLOWED
0005             0173000 .LVL     SET     @01
0047 F1          0174000 FMOD2:   POP     PSW
0048 D162D1      0175000          LFTS    D;          OVERFLOW OCCURRED,RESTORE INPUT
004B 37          0176000          STC                                  CONDITION
004C C169C1      0177000          RELOAD  BC.L
004F C9          0178000          RTN
                 0179000 ;
                 0180000 ; CONVERT (DE.H) TO LOGARITHM, BASE 2
                 0181000 ; TRAP IF (DE.H) <= 0; I.E. ERROR
                 0182000 ; TIME APPROX 9 MILLISEC
                 0183000 ; A,F,D,E,H
0050 7C          0184000 LOG2:    MOV     A,H
0051 B7          0185000          ORA     A
0052 CC0000      0186000          CZ      AERC;       ZERO, TRAP
0055 D8          0187000          RC
0056 7A          0188000          MOV     A,D
0057 17          0189000          RAL
0058 DC0000      0190000          CC      AERC;       NEGATIVE, TRAP
005B D8          0191000          RC
005C C5E5        0192000          SAVE    BC.L
005E E5          0193000          PUSH    H;          SAVE EXPONENT
005F 424B        0194000          MOVD    B,D;        X TO BC.L
0061 110235      0195000          LXI     D,3502H;    SQRT(2)
0064 218181      0196000          LXI     H,8181H;    X RANGE 1 TO 2
0067 CD0000      0197000          CALL    FADD;       X+SQRT(2)
006A D5E5        0198000          SFTS    D;          AND SAVE
006C 11CDB5      0199000          LXI     D,0B502H;   -SQRT(2)
006F 2681        0200000          MVI     H,81H
0071 CD0000      0201000          CALL    FADD;       X-SQRT(2)
0074 C16BC1      0202000          LFTS    B
0077 CD0000      0203000          CALL    FDIV;       FORM TERM
007A 424B6C      0204000          MOVF    B,D;        AND COPY
007D CD0000      0205000          CALL    FMUL
0080 CD0000      0206000          CALL    FMUL;       TERM^3
0083 D5E5        0207000          SFTS    D
0085 11A638      0208000          LXI     D,38A6H;    2.8852
0088 2682        0209000          MVI     H,82H
008A CD0000      0210000          CALL    FMUL
008D C16BC1      0211000          LFTS    B
0090 D5E5        0212000          SFTS    D
0092 11C87B      0213000          LXI     D,7BC8H;    0.9835
0095 2680        0214000          MVI     H,80H
0097 CD0000      0215000          CALL    FMUL
009A C16BC1      0216000          LFTS    B
009D CD0000      0217000          CALL    FADD
00A0 424B6C      0218000          MOVF    B,D;        PARTIAL TERM TO BC.L
00A3 D1          0219000          POP     D;          GET EXPONENT
00A4 7A          0220000          MOV     A,D
00A5 D681        0221000          SUI     81H
00A7 CD0000      0222000          CALL    FLOTA;      CONVERT
00AA CD0000      0223000          CALL    FADD;       ADD CHARACTERISTIC IN
00AD 010000      0224000          LXI     B,0
00B0 2E80        0225000          MVI     L,80H;      0.5000
00B2 CD0000      0226000          CALL    FADD
00B5 C169C1      0227000          RELOAD  BC.L
00B8 C9          0228000          RTN
                 0229000 ;
                 0230000 ; LOG (DE.H), BASE (BC.L) => (DE.H)
                 0231000 ; CARRY FOR OVERFLOW; RETURNS MAX. VALUES OR 0
                 0232000 ; (BC.L) OR (DE.H) <= 0 CAUSES TRAP
                 0233000 ; TIME APPROX. 20 MILLISEC.
                 0234000 ; A,F,D,E,H
00B9 C5E5        0235000 LOGB:    SAVE    BC.L
00BB D5E5        0236000          SFTS    D
00BD 505965      0237000          MOVF    D,B
00C0 CD5000      0238000          CALL    LOG2
00C3 C16BC1      0239000          LFTS    B;          RESTORE OPERAND
00C6 D5E5        0240000          SFTS    D;          SAVE LOG OF BASE
00C8 505965      0241000          MOVF    D,B
00CB CD5000      0242000          CALL    LOG2
00CE C16BC1      0243000          LFTS    B;          RESTORE LOG BASE
00D1 CD0000      0244000          CALL    FDIV
00D4 C169C1      0245000          RELOAD  BC.L
00D7 C9          0246000          RTN
                 0247000 ;
                 0248000 ; EVALUATE POLYNOMIAL IN (DE.H)= X
                 0249000 ; (DE.H):=A(N)*X^N+A(N-1)*X^(N-1)+...A(1)*X+A(0)
                 0250000 ; CARRY FOR ARITHMETIC OVERFLOW
                 0251000 ; (BC) SPECIFIES ADDRESS OF COEFFICIENTS
                 0252000 ; FIRST COEFFICIENT IS ORDER OF POLYNOMIAL (128 MAX)
                 0253000 ; A,F,D,E,H
00DB C5E5        0254000 POLY:    SAVE    BC.L
00DA 0A          0255000          LDAX    B;          GET ORDER
00DB 03          0256000          INX     B;          ADVANCE COEFF. PTR.
00DC D5E5        0257000          SFTS    D;          SAVE ARGUMENT
0004             0258000 @ARG     SET     .LVL;       ARGUMENT STACK ADDRESS
00DE 2600        0259000          MVI     H,0;        CLEAR PARTIAL VALUE
00E0 F5          0260000          PUSH    PSW;        SAVE ORDER COUNTER
00E1 C5          0261000 POLY1:   PUSH    B;          SAVE COEFF. LC'N
00E2 D5E5        0262000          SFTS    D;          SAVE PARTIAL VALUE
00E4 CD0000      0263000          CALL    FLOAD;      GET COEFFICIENT
00E7 C16BC1      0264000          LFTS    B;          RECOVER PARTIAL VALUE TO (BC.H)
00EA CD0000      0265000          CALL    FADD;       ADD IN
00ED C1          0266000          POP     B;          COEFF. PTR
00EE DA0701      0267000          JC      POLY2;      ARITH. OVERFLOW
00F1 F1          0268000          POP     PSW;        ORDER COUNTER
00F2 3D          0269000          DCR     A
00F3 FA0801      0270000          JM      POLY3;      DONE
00F6 F5          0271000          PUSH    PSW;        SAVE ORDER COUNTER
00F7 C5          0272000          PUSH    B;          SAVE COEFFICIENT POINTER
00FB 3E02        0273000          MVI     A,.LVL-@ARG
00FA CD0000      0274000          CALL    LFBS;       GET ARGUMENT
00FD CD0000      0275000          CALL    FMUL;       MULTIPLY
0100 C1          0276000          POP     B;          RESTORE COEFFICIENT PTR.
0101 03          0277000          INX     B
0102 03          0278000          INX     B
0103 03          0279000          INX     B;          ADVANCE TO NEXT COEFFICIENT
0104 D2E100      0280000          JNC     POLY1;      NO ARITH. ERROR
0107 C1          0281000 POLY2:   POP     B;          ERROR EXIT, PURGE STACK
0108 C1          0282000 POLY3:   POP     B
0109 C1          0283000          POP     B;          PURGE ARGUMENT FROM STACK
010A C169C1      0284000          RELOAD  BC.L
010D C9          0285000          RTN
                 0286000 ;
                 0287000 ; EXPONENTIAL (DE.H):=2^(DE.H)
                 0288000 ; CARRY FOR OVERFLOW
                 0289000 ; A,F,D,E,H
010E 7A          0290000 EXP2:    MOV     A,D
010F B7          0291000          ORA     A
0110 F22001      0292000          JP      EXP21
0113 EE80        0293000          XRI     80H;        SET POSITIVE
0115 57          0294000          MOV     D,A
0116 CD2001      0295000          CALL    EXP21
0119 D40000      0296000          CNC     FRCIP;      NEG. EXPONENT
011C D0          0297000          RNC
011D 2600        0298000          MVI     H,0;        ZERO FOR -VE OVERFLOW
011F C9          0299000          RTN
0120 C5E5        0300000 EXP21:   SAVE    BC.L
0122 424B6C      0301000          MOVF    B,D;        COPY ARGUMENT TO B
0125 CD0000      0302000          CALL    FIXR
0128 DA4801      0303000          JC      EXP22;      TOO LARGE, OVERFLOW
012B D5          0304000          PUSH    D;          SAVE INTEGER PORTION
012C CD0000      0305000          CALL    FLOTD
012F CD0000      0306000          CALL    FSUBR;      FORM FRACTIONAL PORTION
0132 015101      0307000          LXI     B,EX2C;     POINT TO COEFFICIENTS
0135 CDDB00      0308000          CALL    POLY;       FORM 2^(FRACT(X))
0138 424B6C      0309000          MOVF    B,D
013B CD0000      0310000          CALL    FMUL;       FORM (1+A1*X+...+AN*X^N)^2
013E C1          0311000          POP     B;          GET INTEGER PORTION(X)
013F 78          0312000          MOV     A,B
```

```
0140 B7          0313000          ORA     A
0141 37          0314000          STC
0142 C24801      0315000          JNZ     EXP22;     TOO LARGE, OVERFLOW
0145 79          0316000          MOV     A,C
0146 84          0317000          ADD     H
0147 67          0318000          MOV     H,A;       EXPONENT OVERFLOW CAUSES CARRY
0148 C169C1      0319000 EXP22:   RELOAD BC.L
014B D0          0320000          RNC
014C 11FF7F      0321000          LXI     D,7FFFH
014F 63          0322000          MOV     H,E;       SET MAX. VALUE
0150 C9          0323000          RTN
                 0324000 ;
                 0325000 ; POLYNOMIAL COEFFICIENTS FOR 2^(X)
0151 03          0326000 EX2C:    DB      3;         POLYNOMIAL ORDER
0152 7A0106      0327000          DB      7AH,01H,06H; 0.0081790
0155 7C0D73      0328000          DB      7CH,0DH,73H; 0.059340
0158 7F8131      0329000          DB      7FH,81H,31H; 0.34669
015B 810000      0330000          DB      81H,0,0;   1.0000
                 0331000 ;
                 0332000 ; EXPONENTIAL (DE.H):=(BC.L)^(DE.H)
                 0333000 ; (BC.L) < 0 ILLEGAL, DIVERT TO TRAP
                 0334000 ; (BC.L) AND (DE.H) = 0 ILLEGAL, TRAP
                 0335000 ; CARRY FOR OVER/UNDERFLOW, RETURNS MAX,0,1
                 0336000 ; A,F,D,E,H
015E 7D          0337000 EXPX:    MOV     A,L
015F B7          0338000          ORA     A
0160 C26A01      0339000          JNZ     EXPX1;     (BC.L) <> 0
0163 B4          0340000          ORA     H
0164 CC0000      0341000          CZ      AERC;      ILLEGAL, TRAP
0167 2600        0342000          MVI     H,0;       0^ANY=0
0169 C9          0343000          RTN
016A 78          0344000 EXPX1:   MOV     A,B
016B B7          0345000          ORA     A
016C FC0000      0346000          CM      AERC;      ILLEGAL,TRAP
016F D8          0347000          RC
0170 7C          0348000          MOV     A,H
0171 B7          0349000          ORA     A
0172 C27B01      0350000          JNZ     EXPX3
0175 110000      0351000 EXPX2:   LXI     D,0
0178 2681        0352000          MVI     H,81H;     ANY^0=1.000
017A C9          0353000          RTN
017B C5E5        0354000 EXPX3:   SAVE    BC.L
017D D5E5        0355000          SFTS    D
017F 505965      0356000          MOVF    D,B
0182 CD5000      0357000          CALL    LOG2
0185 C168C1      0358000          LFTS    B;         RESTORE ARGUMENT
0188 CD0000      0359000          CALL    FMUL
018B C169C1      0360000          RELOAD BC.L
018E D20E01      0361000          JNC     EXP2
0191 7C          0362000          MOV     A,H
0192 B7          0363000          ORA     A
0193 37          0364000          STC
0194 CA7501      0365000          JZ      EXPX2;     UNDERFLOW, RETURN 1.000
0197 7A          0366000          MOV     A,D
0198 B7          0367000          ORA     A
0199 37          0368000          STC
019A F0          0369000          RP;                +VE OVERFLOW, RETURN MAX
019B 2600        0370000          MVI     H,0;       -VE OVERFLOW, RETURN 0
019D C9          0371000          RTN
                 0372000 ;
                 0373000 ; ***********************************************
                 0374000 ; * THE LOGICAL OPERATORS                       *
                 0375000 ; *  TREAT ALL ARGUMENTS AS SIGNED INTEGERS,    *
                 0376000 ; *  AND RETURN THE FLOATING REPRESENTATION OF  *
                 0377000 ; *  THE BITWISE OPERATION SPECIFIED            *
                 0378000 ; *  ERROR IF ANY ARGUMENT IS OUTSIDE THE       *
                 0379000 ; *  RANGE -32768 TO 32767                      *
                 0380000 ; ***********************************************
                 0381000 ;
                 0382000 ; LOGICAL OR ON (BC.L),(DE.H)
                 0383000 ; A,F,D,E,H
019E C5E5        0384000 .OR.:    SAVE    BC.L
01A0 CD1100      0385000          CALL    FIXRT
01A3 DAAF01      0386000          JC      .OR.2
01A6 78          0387000          MOV     A,B
01A7 B2          0388000          ORA     D
01A8 57          0389000          MOV     D,A
01A9 79          0390000          MOV     A,C
01AA B3          0391000          ORA     E
01AB 5F          0392000 .OR.1:   MOV     E,A
01AC CD0000      0393000          CALL    FLDT
01AF C169C1      0394000 .OR.2:   RELOAD BC.L
01B2 C9          0395000          RTN
                 0396000 ;
                 0397000 ; LOGICAL AND ON (BC.L),(DE.H)
                 0398000 ; A,F,D,E,H
01B3 C5E5        0399000 .AND.:   SAVE    BC.L
01B5 CD1100      0400000          CALL    FIXRT
01B8 DAAF01      0401000          JC      .OR.2
01BB 78          0402000          MOV     A,B
01BC A2          0403000          ANA     D
01BD 57          0404000          MOV     D,A
01BE 79          0405000          MOV     A,C
01BF A3          0406000          ANA     E
01C0 C3AB01      0407000          JMP     .OR.1
0000             0408000 .LVL     SET     .LVL-2
                 0409000 ;
                 0410000 ; LOGICAL XOR ON (BC.L),(DE.H)
                 0411000 ; A,F,D,E,H
01C3 C5E5        0412000 .XOR.:   SAVE    BC.L
01C5 CD1100      0413000          CALL    FIXRT
01C8 DAAF01      0414000          JC      .OR.2
01CB 78          0415000          MOV     A,B
01CC AA          0416000          XRA     D
01CD 57          0417000          MOV     D,A
01CE 79          0418000          MOV     A,C
01CF AB          0419000          XRA     E
01D0 C3AB01      0420000          JMP     .OR.1
0000             0421000 .LVL     SET     .LVL-2
                 0422000 ;
                 0423000 ; ****************************
                 0424000 ; * THE RELATIONAL OPERATORS *
                 0425000 ; *  RETURN -1 FOR TRUE,     *
                 0426000 ; *           0 FOR FALSE    *
                 0427000 ; ****************************
                 0428000 ;
                 0429000 ; TEST (DE.H) = (BC.L).
                 0430000 ; IF SO (DE.H):=-1, ELSE 0
                 0431000 ; A,F,D,E,H
01D3 CD0000      0432000 .EQU.:   CALL    FCMP
01D6 CAE201      0433000          JZ      .TRU;      TRUE
01D9 AF          0434000 .NTRU:   XRA     A
01DA 67          0435000          MOV     H,A;       (DE.H):=0
01DB C9          0436000          RTN
                 0437000 ;
                 0438000 ; TEST (DE.H) <> (BC.L).
                 0439000 ; IF SO (DE.H):=-1, ELSE 0
                 0440000 ; A,F,D,E,H
01DC CD0000      0441000 .NE.:    CALL    FCMP
01DF CAD901      0442000          JZ      .NTRU;     FALSE
01E2 2681        0443000 .TRU:    MVI     H,81H;     (DE.H):=-1.0
01E4 110080      0444000          LXI     D,8000H;   USE LXI D,0 FOR TRUE = +1 0, FOR
01E7 B7          0445000          ORA     A;         CLEAR ANY CARRY      PASCAL, ETC.
01E8 C9          0446000          RTN
                 0447000 ;
                 0448000 ; TEST (DE.H) <= (BC.L).
                 0449000 ; IF SO (DE.H):=-1, ELSE 0
                 0450000 ; A,F,D,E,H
01E9 CD0000      0451000 .LE.:    CALL    FCMP
01EC F2E201      0452000 .LE.1:   JP      .TRU
01EF C3D901      0453000          JMP     .NTRU
                 0454000 ;
                 0455000 ; TEST (DE.H) > (BC.L).
                 0456000 ; IF SO (DE.H):=-1, ELSE 0
                 0457000 ; A,F,D,E,H
01F2 CD0000      0458000 .GT.:    CALL    FCMP
01F5 FAE201      0459000 .GT.1:   JM      .TRU
01F8 C3D901      0460000          JMP     .NTRU
                 0461000 ;
                 0462000 ; TEST (DE.H) < (BC.L).
                 0463000 ; IF SO (DE.H):=-1, ELSE 0
                 0464000 ; A,F,D,E,H
```

```
01FB CD0000      0465000 .LT.:   CALL    FCMP
01FE CAD901      0466000         JZ      .NTRU
0201 C3EC01      0467000         JMP     .LE.1
                 0468000 ;
                 0469000 ; TEST (DE.H) >= (BC.L).
                 0470000 ; IF SO (DE.H):=-1, ELSE 0
                 0471000 ; A,F,D,E,H
0204 CD0000      0472000 .GE.:   CALL    FCMP
0207 CAE201      0473000         JZ      .TRU
020A C3F501      0474000         JMP     .GT.1
                 0475000 ;
                 0476000 ; SET (DE.H):=(DE.H)*2^15 AND PERFORM .GT.
                 0477000 ; THIS CAN BE USED TO TEST FOR A VALUE EFFECTIVELY
                 0478000 ; ZERO WITH RESPECT TO ANOTHER, FOR TERMINATION OF
                 0479000 ; ITERATION LOOPS, ETC.  THE VALUE 2^15 APPLIES TO
                 0480000 ; THIS ARITHMETIC SYSTEM, AND SHOULD BE CUSTOMIZED
                 0481000 ; TO THE PRECISION OF ANY PARTICULAR ARITHMETIC
                 0482000 ; SYSTEM FOR PROGRAM PORTABILITY.
020D 7C          0483000 .GG.:   MOV     A,H
020E C60F        0484000         ADI     15
0210 D21F02      0485000         JNC     .GG.1;    DYNAMIC ROOM
0213 E5          0486000         PUSH    H
0214 7D          0487000         MOV     A,L
0215 D60F        0488000         SUI     15
0217 6F          0489000         MOV     L,A
0218 D22102      0490000         JNC     .GG.2;    DYNAMIC ROOM
021B E1          0491000         POP     H;        NO ROOM, RETURN FALSE
021C C3D901      0492000         JMP     .NTRU
021F 67          0493000 .GG.1:  MOV     H,A
0220 E5          0494000         PUSH    H
0221 CD0000      0495000 .GG.2:  CALL    FCMP
0224 E1          0496000         POP     H
0225 C3F501      0497000         JMP     .GT.1
                 0498000 ;
0000             0499000         END.    0
PASS 2 ERRORS = 0
.AND.   IDR     001B3H      .EQU.   IDR     001D3H      .GE.    IDR     00204H
.GG.    IDR     0020DH      .GG.1   IDR     0021FH      .GG.2   IDR     00221H
.GT     IDR     001F2H      .GT.1   IDR     001F5H      .LE.    IDR     001E9H
.LE.1   IDR     001ECH      .LT.    IDR     001F8H      .LVL    RED     00000H
.NE.    IDR     001DCH      .NTRU   IDR     001D9H      .OR.    IDR     0019EH
.OR.1   IDR     001ABH      .OR.2   IDR     001AFH      .TRU    IDR     001E2H
.XOR.   IDR     001C3H      @01     RED     00005H      @ARG    RED     00004H
AERC    EXT     00002H      EX2C    IDR     00151H      EXP2    IDR     0010EH
EXP21   IDR     00120H      EXP22   IDR     00148H      EXPX    IDR     0015EH
EXPX1   IDR     0016AH      EXPX2   IDR     00175H      EXPX3   IDR     00178H
FADD    EXT     00003H      FCMP    EXT     0000FH      FDIV    EXT     00004H
FDIVR   EXT     00005H      FINT    EXT     00006H      FIXR    EXT     00007H
FIXRT   IDR     00011H      FLOAD   EXT     00010H      FLOT    EXT     0000BH
FLOTA   EXT     0000CH      FLOTD   EXT     0000DH      FMOD    IDR     0001BH
FMOD1   IDR     0003AH      FMOD2   IDR     00047H      FMUL    EXT     00008H
FRACT   IDR     00000H      FRCIP   EXT     00009H      FSUBR   EXT     0000AH
FXCHG   EXT     0000EH      LFBS    EXT     00011H      LFTS    MACRO
LOG2    IDR     00050H      LOGB    IDR     000B9H      MOVD    MACRO
MOVF    MACRO               POLY    IDR     000D9H      POLY1   IDR     000E1H
POLY2   IDR     00107H      POLY3   IDR     00108H      RELOAD  MACRO
RTN     MACRO               SAVE    MACRO               SFTS    MACRO
```

the END!

Bringing Up UCSD Pascal without CP/M

BY MIKE GABRIELSON
Box 2692
Stanford, CA

Even if you don't have CP/M or a CP/M compatible system, you can transport UCSD 8080 PASCAL to your machine by using XTRACT, a program for extracting files from UCSD floppies:

Step 1. Supply the appropriate read and write routines as described in the comments at the start of the XTRACT listing. XTRACT will be used to copy the PASCAL interpreter source code from the UCSD disk to your target machine.

Step 2. Determine where the interpreter source files are. For Release I.4b, the interpreter was contained on the disk labeled "SOURCE3:" and consisted of nine (9) source files named:

```
INTERP.TEXT
VARS.TEXT
ARITH.TEXT
FP.TEXT
SET.TEXT
PROC.TEXT
STP.TEXT
CPMIO.TEXT
BOOT.TEXT
```

Step 3. Transfer the interpreter source code files to your machine using XTRACT.

Step 4. Supply the interpreter with your own I/O drivers in lieu of the standard CP/M BIOS drivers. The Digital Research document "CP/M System Alteration Guide" gives a good description of the BIOS structure.

Step 5. Reassemble the interpreter now configured for your Pascal system as described in the portion of the interpreter named BOOT.TEXT.

Good luck!

```
0000                    ;=====
0000                    ;xtract - extract a UC San Diego Pascal file
0000                    ;
0000                    ;Mike Gabrielson  12/5/78
0000                    ;
0000                    ;ACCEPTS:  HL = @ PASCAL FILE NAME, TERMINATED BY A BLANK
0000                    ;RETURNS:  CARRY SET IF FILE NOT FOUND
0000                    ;=====
0000                        NOEJECT
0000
0000                    ;EXTERNAL ROUTINES NOT DEFINED IN THIS ASSEMBLY:
0000                            !EXTRN $READ    ;READ A SECTOR (DE = DESTINATION ADDRESS,
0000                                            ;               H = TRACK, L = SECTOR)
0000                                            ;
0000                            !EXTRN $WRITE   ;OUTPUT DATA (BC = # BYTES TO OUTPUT,
0000                                            ;             DE = ADDRESS OF DATA)
0000                                            ;
0000                    BLKSIZ = 512            ;BYTES PER PASCAL BLOCK
0000                    DIRBLK = 2              ;DIRECTORY'S 1ST PASCAL BLOCK #
0000                    DIRSIZ = 4              ;# PASCAL BLOCKS IN DIRECTORY
0000                    ENTRYS = 77             ;MAXIMUM ENTRIES PER DIRECTORY
0000                    ENTSIZ = 26             ;BYTES PER DIRECTORY ENTRY
0000                    OFFSET = 6              ;BYTES FROM START OF DIRECTORY ENTRY
0000                                            ;TO ENTRY NAME LENGTH
0000                    SECSIZ = 128            ;BYTES PER PHYSICAL SECTOR
0000
0000                    XTRACT:
0000 E5                         PUSH    H       ;SAVE @ FILE NAME
0001 CD ?? ??                   CALL    GETDIR  ;READ THE PASCAL DIRECTORY
0004 E1                         POP     H       ;GET @ FILE NAME
0005 CD ?? ??                   CALL    LOOKUP  ;FIND FILE IN DIRECTORY
0008 D8                         RC              ;QUIT IF FILE NOT FOUND
0009 CD ?? ??                   CALL    COPY    ;COPY THE FILE
000C B7                         ORA     A       ;CLEAR CARRY TO INDICATE SUCCESS
000D C9                         RET             ;XTRACT COMPLETE
000E
000E                    ;=====
000E                    ;COMPARE - COMPARE FILE NAME TO DIRECTORY ENTRY
000E                    ;ACCEPTS:  B = FILE NAME LENGTH
000E                    ;          DE = @ NAME
000E                    ;          HL = @ ENTRY
000E                    ;RETURNS:  Z SET APPROPRIATELY
000E                    ;=====
000E                    COMPARE:
000E C5                         PUSH    B
000F D5                         PUSH    D
0010 E5                         PUSH    H
0011 78                         MOV     A,B
0012 01 06 00                   LXI     B,OFFSET        ;GET OFFSET TO ENTRY NAME LENGTH
0015 09                         DAD     B               ;GET @ ENTRY NAME LENGTH
0016 47                         MOV     B,A             ;RESTORE FILE NAME LENGTH
0017 BE                         CMP     M               ;SAME AS ENTRY'S LENGTH?
0018 C2 ?? ??                   JNZ     NOTEQ           ;NO
001B 23                 CMLOOP: INX     H               ;GET @ NEXT ENTRY NAME CHARACTER
001C 1A                         LDAX    D               ;GET NEXT FILE NAME CHARACTER
001D BE                         CMP     M               ;SAME AS DIRECTORY ENTRY?
001E C2 ?? ??                   JNZ     NOTEQ           ;NO
0021 13                         INX     D               ;YES,
0022 05                         DCR     B               ;END OF NAMES?
0023 C2 1B 00                   JNZ     CMLOOP          ;NO
0026                    ;
0026                    ;FALL THROUGH TO HERE IF EQUAL, JUMP TO HERE IF NOT EQUAL
0026                    ;
0026 E1                 NOTEQ:  POP     H
0027 D1                         POP     D
0028 C1                         POP     B
0029 C9                         RET
002A
002A                    ;=====
002A                    ;COPY - COPY PASCAL FILE TO DESTINATION
002A                    ;ACCEPTS:  HL = @ PASCAL DIRECTORY ENTRY
002A                    ;=====
002A 5E                 COPY:   MOV     E,M             ;GET FILE'S FIRST PASCAL BLOCK #
002B 23                         INX     H               ;INTO DE
002C 56                         MOV     D,M
002D 23                         INX     H
002E 4E                         MOV     C,M             ;GET ENDING BLOCK # INTO BC
002F 23                         INX     H
0030 46                         MOV     B,M
0031 EB                         XCHG                    ;SAVE 1ST BLOCK # IN HL
0032 11 ?? ??                   LXI     D,BUFFER        ;GET @ BUFFER
0035 CD ?? ??           CPLOOP: CALL    GETBLK          ;READ THE PASCAL BLOCK
0038 CD ?? ??                   CALL    PUTBLK          ;AND THEN OUTPUT IT
003B 23                         INX     H               ;GET NEXT BLOCK #
```

```
 87 003C 7C                 MOV     A,H             ;ALL DONE?
 88 003D B8                 CMP     B               ;(HL = BC?)
 89 003E DA 35 00           JC      CPLOOP          ;NO, KEEP COPYING
 90 0041 7D                 MOV     A,L
 91 0042 B9                 CMP     C
 92 0043 DA 35 00           JC      CPLOOP          ;NO
 93 0046 C9                 RET
 94 0047
 95 0047                ;=====
 96 0047                ;DIVID - DIVIDE ROUTINE
 97 0047                ;=====
 98 0047 01 E6 FF       DIVID:  LXI     B,-26
 99 004A 1E FF                  MVI     E,#FF
100 004C 1C             REDIV:  INR     E
101 004D 09                     DAD     B
102 004E 7C                     MOV     A,H
103 004F B7                     ORA     A
104 0050 F2 4C 00               JP      REDIV
105 0053 01 1A 00               LXI     B,26
106 0056 09                     DAD     B
107 0057 C9                     RET
108 0058
109 0058                ;=====
110 0058                ;GETBLK - GET A PASCAL BLOCK
111 0058                ;ACCEPTS:  DE = @ BUFFER
112 0058                ;          HL = BLOCK #
113 0058                ;=====
114 0058 C5             GETBLK: PUSH    B               ;SAVE CALLER'S REGISTERS
115 0059 D5                     PUSH    D
116 005A E5                     PUSH    H
117 005B 29                     DAD     H               ;THERE ARE 4 SECTORS TO A PASCAL BLOCK
118 005C 29                     DAD     H               ;SO MULTIPLY BY 4 TO GET 1ST SECTOR
119 005D 0E 04                  MVI     C,BLKSIZ/SECSIZ ;GET NUMBER OF SECTORS TO READ
120 005F C5             GBLOOP: PUSH    B               ;SAVE SECTOR COUNT
121 0060 E5                     PUSH    H               ;SAVE SECTOR NUMBER
122 0061 D5                     PUSH    D               ;SAVE @ BUFFER
123 0062 CD ?? ??               CALL    MAP             ;CONVERT LOGICAL SECTOR TO TRACK & SECTOR
124 0065 D1                     POP     D               ;GET @ BUFFER
125 0066 D5                     PUSH    D
126 0067 CD -- --               CALL    $READ           ;READ A SECTOR
127 006A D1                     POP     D               ;GET @ BUFFER
128 006B 21 80 00               LXI     H,SECSIZ        ;GET # BYTES JUST READ
129 006E 19                     DAD     D               ;ADJUST BUFFER ADDRESS
130 006F EB                     XCHG                    ;AND LEAVE IN DE
131 0070 E1                     POP     H               ;GET SECTOR # JUST READ
132 0071 23                     INX     H               ;BUMP FOR NEXT TIME AROUND
133 0072 C1                     POP     B               ;RESTORE COUNT OF SECTORS READ
134 0073 0D                     DCR     C               ;PASCAL BLOCK DONE?
135 0074 C2 5F 00               JNZ     GBLOOP          ;NO
136 0077 E1                     POP     H               ;YES, RESTORE CALLER'S REGISTERS
137 0078 D1                     POP     D
138 0079 C1                     POP     B
139 007A C9                     RET
140 007B
141 007B                ;=====
142 007B                ;GETDIR - GET PASCAL DIRECTORY
143 007B                ;=====
144 007B 01 04 00       GETDIR: LXI     B,DIRSIZ        ;GET # PASCAL BLOCKS IN DIRECTORY
145 007E 11 ?? ??               LXI     D,BUFFER        ;GET @ DIRECTORY BUFFER
146 0081 21 02 00               LXI     H,DIRBLK        ;GET DIRECTORY'S 1ST PASCAL BLOCK #
147 0084 CD 58 00       GDLOOP: CALL    GETBLK          ;READ A BLOCK
148 0087 E5                     PUSH    H               ;SAVE CURRENT BLOCK NUMBER
149 0088 21 00 02               LXI     H,BLKSIZ        ;GET # BYTES PER BLOCK
150 008B 19                     DAD     D               ;ADJUST BUFFER ADDRESS
151 008C EB                     XCHG                    ;AND LEAVE IN DE
152 008D E1                     POP     H               ;RESTORE OLD BLOCK NUMBER
153 008E 23                     INX     H               ;PREPARE TO READ NEXT BLOCK
154 008F 0B                     DCX     B               ;ALL BLOCKS READ?
155 0090 78                     MOV     A,B             ;(BC = 0?)
156 0091 B1                     ORA     C
157 0092 C2 84 00               JNZ     GDLOOP          ;NO
158 0095 C9                     RET
159 0096
160 0096                ;=====
161 0096                ;LOOKUP - SEARCH DIRECTORY FOR FILE NAME
162 0096                ;ACCEPTS:  HL = @ FILE NAME TO SEARCH FOR, TERMINATED BY A BLANK
163 0096                ;RETURNS:  CARRY SET IF NOT FOUND
164 0096                ;          ELSE HL = @ DIRECTORY ENTRY
165 0096                ;=====
166 0096 54             LOOKUP: MOV     D,H             ;KEEP @ FILE NAME IN DE
167 0097 5D                     MOV     E,L
168 0098 23             FINDBL: INX     H               ;FIND END OF FILE NAME
169 0099 7E                     MOV     A,M
170 009A FE 20                  CPI     ' '
171 009C C2 98 00               JNZ     FINDBL          ;MARKED BY A BLANK
172 009F 7D                     MOV     A,L             ;COMPUTE LENGTH OF FILE NAME
173 00A0 93                     SUB     E               ;(HL := HL - DE)
174 00A1 6F                     MOV     L,A
175 00A2 7C                     MOV     A,H
176 00A3 9A                     SBB     D
177 00A4 67                     MOV     H,A
178 00A5 45                     MOV     B,L             ;GET ONE BYTE LENGTH INTO B FOR COMPARE
179 00A6 0E 4D                  MVI     C,ENTRYS        ;GET MAXIMUM # OF ENTRIES IN DIRECTORY
180 00A8 21 ?? ??               LXI     H,BUFFER        ;GET @ DIRECTORY
181 00AB C5             LULOOP: PUSH    B               ;SAVE FILE NAME LENGTH
182 00AC 01 1A 00               LXI     B,ENTSIZ        ;GET LENGTH OF EACH ENTRY
183 00AF 09                     DAD     B               ;POINT TO NEXT ENTRY
184 00B0 C1                     POP     B               ;(1ST ENTRY CONTAINS VOLUME LABEL)
185 00B1 CD 0E 00               CALL    COMPARE         ;DOES THIS ENTRY CONTAIN THE FILE NAME?
186 00B4 37                     STC                     ;(ASSUME SO,
187 00B5 3F                     CMC                     ; CLEAR CARRY)
188 00B6 C8                     RZ                      ;YES, ENTRY FOUND
189 00B7 0D                     DCR     C               ;NO, ANY MORE ENTRIES?
190 00B8 C2 AB 00               JNZ     LULOOP          ;YES
191 00BB 37                     STC                     ;NO,
192 00BC C9                     RET                     ;FILE NOT FOUND
193 00BD
194 00BD                ;=====
195 00BD                ;MAP - CONVERT LOGICAL SECTOR TO TRACK AND SECTOR
196 00BD                ;ACCEPTS:  HL = LOGICAL SECTOR
197 00BD                ;RETURNS:  H = PHYSICAL TRACK
198 00BD                ;          L = PHYSICAL SECTOR
199 00BD                ;=====
200 00BD CD 47 00       MAP:    CALL    DIVID
201 00C0 7D                     MOV     A,L
202 00C1 87                     ADD     A
203 00C2 47                     MOV     B,A
204 00C3 3E 0C                  MVI     A,12
205 00C5 BD                     CMP     L
206 00C6 D2 ?? ??               JNC     NEXT
207 00C9 04                     INR     B
208 00CA 4B             NEXT:   MOV     C,E
209 00CB AF                     XRA     A
210 00CC 57                     MOV     D,A
211 00CD 67                     MOV     H,A
212 00CE 68                     MOV     L,B
213 00CF 3E 06                  MVI     A,6
214 00D1 19             REMAP:  DAD     D
215 00D2 3D                     DCR     A
216 00D3 C2 D1 00               JNZ     REMAP
217 00D6 C5                     PUSH    B
218 00D7 CD 47 00               CALL    DIVID
219 00DA C1                     POP     B
220 00DB 2C                     INR     L
221 00DC 61                     MOV     H,C
222 00DD 24                     INR     H
223 00DE C9                     RET
224 00DF
225 00DF                ;=====
226 00DF                ;PUTBLK - OUTPUT PASCAL BLOCK
227 00DF                ;ACCEPTS:  DE = @ BUFFER
228 00DF                ;=====
229 00DF C5             PUTBLK: PUSH    B
230 00E0 D5                     PUSH    D
231 00E1 E5                     PUSH    H
232 00E2 01 00 02               LXI     B,BLKSIZ        ;GET # BYTES PER PASCAL BLOCK
233 00E5 CD -- --               CALL    $WRITE          ;WRITE THE DATA
234 00E8 E1                     POP     H
235 00E9 D1                     POP     D
236 00EA C1                     POP     B
237 00EB C9                     RET
238 00EC
239 08EC                BUFFER: DS      DIRSIZ*BLKSIZ   ;BLOCK BUFFER, ALSO HOLDS DIRECTORY
```

```
0000 XTRACT
0007 SP
0080 SECSIZ
00D1 REMAP
004C REDIV
00DF PUTBLK
0006 PSW
0026 NOTEQ
00CA NEXT
00BD MAP
00AB LOOKUP
0096 LOOKUP
0005 L
0004 H
007B GETDIR
0058 GETBLK
005F GBLOOP
0084 GDLOOP
0098 FINDBL
001A ENTSIZ
004D ENTRYS
0003 E
0047 DIVID
0004 DIRSIZ
0002 DIRBLK
0002 D
0035 CPLOOP
002A COPY
000E COMPARE
001B CMLOOP
0001 C
0200 BUFFER
0000 B
0200 BLKSIZ
0007 A
----- $WRITE
----- $READ
```

Unspecified 8085 Op Codes Enhance Programming

BY WOLFGANG DEHNHARDT AND VILLY M. SORENSEN
GSI, Darmstadt, and Sorensen Software
Seeheim, West Germany

Ten operating codes and two flag bits previously unknown to most users of the 8085 microprocessor will enable programmers to write more efficient routines. The new members of the instruction set, which were stumbled upon during the testing of an assembler-disassembler module, include seven op codes that involve the processing of register pairs, two that involve jump operations with one new flag bit, and one that performs a conditional restart on the overflow indication of the other flag bit.

The seven register pair instructions (all with 16-bit operands) consist of a double subtraction, a rotate, a shift, indirect loading and storing of a word, and two offset operations. Either BC, DE, HL, or SP are the designated register pairs used in these op codes.

The mnemonic names of the instructions have been selected to be compatible with the 8085's existing mnemonics. In the double subtraction (DSUB), register pair BC is subtracted from HL. This instruction thus performs the opposite task of DAD B, a well-known instruction. The instruction RDEL rotates register pair DE left 1 bit through the carry. ARHL is an arithmetic shift to the right of HL. It serves to divide HL by 2, except in cases where HL is -1.

All 16 bits of register pair HL can be stored indirectly at the address contained in the DE pair by specifying instruction SHLX. To load HL, LHLX must be employed.

As an example of how this instruction can be used to cut instruction steps, consider the common sequence used for a routine table jump shown in part (a) of the figure. By assigning the register DE for HL and using the LHLX instruction, this sequence can be replaced by the much simpler arrangement shown at the bottom of part (a).

As for adding the contents of register pairs with an additional byte (offset), DE can be loaded with HL plus the byte by selecting the instruction LDHI, which simplifies array addressing. Usually, the architecture of 8080-type systems dictate the addressing of arrays in what are called pages of 256 bytes. This restriction means that the starting address of an array must be placed near the beginning of a page. A typical call is as shown in part (b) of the figure.

The page limitation is bypassed using the LDHI instruction code and constant indexes. The starting address of the array can now be placed anywhere, and addressing occurs as shown at the bottom of part b.

Any additional byte can be combined with register-pair SP in DE if instruction LDSI is specified. This instruction is designed for operating system routes that transfer arguments on the stack. An example sequence, shown in (c), stores HL into the 16-bit word located as the second item below the top of the stack.

The jump and restart instructions work in conjunction with the two discovered flag bits, X5 and V. Op codes JX5 and JNX5 jump depending on the state of the X5 flag. Op code RSTV makes a restart call to hexadecimal address 40 if the V flag is set; otherwise it functions as a no-operation instruction.

Flag bit V indicates a 2's complement overflow condition for 8- and 16-bit arithmetical operations. Flag bit X5 has been named for its position in the condition code byte and not for its function. It does not resemble any normal flag bit. The only use for this bit found thus far are as an unsigned overflow indicator resulting from a data change of FFFF to 0000 on executing the instruction of INX and as an unsigned underflow indicator from a data change of 0000 to FFFF on executing DCX.

The new 8085 instructions are outlined in the table.

(a)

Source statement		Comments
MOV	E, M	;ROUTINE ADR LOW BYTE
INX	H	;HL = TABLE ADR
MOV	D, M	;ROUTINE ADR HIGH BYTE
XCHG		;DE = ROUTINE ADR
PCHL		;GO TO ROUTINE ADR

⬇

LHLX		;DE = TABLE ADR
PCHL		;HL = ROUTINE ADR

(b)

Source statement		Comments
LXI	H, ARRAY	;ARRAY BASE ADR
MVI	L, INDEX	;8-BIT INDEX, HL = ARRAY ADR

⬇

LXI	H, ARRAY	;ARRAY BASE ADR
LDHI	INDEX	;8-BIT INDEX, DE = ARRAY ADR

(c)

Source statement		Comments
LDSI	2	;DE = SP + 2
SHLX		;REPLACE 2. ITEM ON STACK

Reprinted from *Electronics*, January 18, 1979
©McGraw Hill, Inc., 1979

NEW CONDITION CODES: V = bit 1
X5 = bit 5

Condition code format							
S	Z	X5	AC	0	P	V	C

DSUB (double subtraction)

(H) (L) = (H) (L) − (B) (C)

The contents of register pair B and C are subtracted from the contents of register pair H and L. The result is placed in register pair H and L. All condition flags are affected.

0 0 0 0 1 0 0 0

(08)

cycles: 3
states: 10
addressing: register
flags: Z, S, P, CY, AC, X5, V

ARHL (arithmetic shift of H and L to the right)

(H7=H7); (Hn−1) = (Hn)
(L7=Ho); (Ln−1) = (Ln); (CY) = (Lo)

The contents of register pair H and L are shifted right one bit. The uppermost bit is duplicated and the lowest bit is shifted into the carry bit. The result is placed in register pair H and L. Note: only the CY flag is affected.

0 0 0 1 0 0 0 0

(10)

cycles: 2
states: 7
addressing: register
flags: CY

RDEL (rotate D and E left through carry)

(Dn+1) = (Dn); (Do) = (E7)
(CY) = (D7); (En+1) = (En); (Eo) = (CY)

The contents of register pair D and E are rotated left one position through the carry flag. The low-order bit is set equal to the CY flag and the CY flag is set to the value shifted out of the high-order bit. Only the CY and the V flags are affected.

0 0 0 1 1 0 0 0

(18)

cycles: 3
states: 10
addressing: register
flags: CY, V

LDHI (load D and E with H and L plus immediate byte)

(D) (E) = (H) (L) + (byte 2)

The contents of register pair H and L are added to the immediate byte. The result is placed in register pair D and E. Note: no condition flags are affected.

0 0 1 0 1 0 0 0
data

(28)

cycles: 3
states: 10
addressing: immediate register
flags: none

LDSI (load D and E with SP plus immediate byte)

(D) (E) = (SPH)(SPL) + (byte 2)

The contents of register pair SP are added to the immediate byte. The result is placed in register pair D and E. Note: no condition flags are affected.

0 0 1 1 1 0 0 0
data

(38)

cycles: 3
states: 10
addressing: immediate register
flags: none

2's complement overflow
Underflow (DCX) or overflow (INX)
X5 = 01·02 + 01·R + 02·R, where
01 = sign of operand 1, 02 = sign of operand 2,
R = sign of result. For subtraction and comparisons, replace 02 with $\overline{02}$.

RSTV (restart on overflow)

If (V):
((SP)−1) = (PCH)
((SP)−2) = (PCL)
(SP) = (SP)−2
(PC) = 40 hex

If the overflow flag V is set, the actions specified above are performed; otherwise control continues sequentially.

1 1 0 0 1 0 1 1

(C8)

cycles: 1 or 3
states: 6 or 12
addressing: register indirect
flags: none

SHLX (store H and L indirect through D and E)

((D)(E)) = (L)
((D)(E)+1) = (H)

The contents of register L are moved to the memory location whose address is in register pair D and E. The contents of register H are moved to the succeeding memory location.

1 1 0 1 1 0 0 1

(D9)

cycles: 3
states: 10
addressing: register indirect
flags: none

JNX5 (jump on not X5)

If (not X5):
(PC) = (byte 3) (byte 2)

If the X5 flag is reset, control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction; otherwise control continues sequentially.

1 1 0 1 1 1 0 1
low-order address
high-order address

(DD)

cycles: 2 or 3
states: 7 or 10
addressing: immediate
flags: none

LHLX (load H and L indirect through D and E)

(L) = ((D)(E))
(H) = ((D)(E)+1)

The content of the memory location whose address is in D and E, are moved to register L. The contents of the succeeding memory location are moved to register H.

1 1 1 0 1 1 0 1

(ED)

cycles: 3
states: 10
addressing: register indirect
flags: none

JX5 (jump on X5)

If (X5):
(PC) = (byte 3) (byte 2)

If the X5 flag is reset, control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction; otherwise control continues sequentially.

1 1 1 1 1 1 0 1
low-order address
high-order address

(FD)

cycles: 2 or 3
states: 7 or 10
addressing: immediate
flags: none

CONVERSION:
8-bit Memories to 16-bits

BY GEORGE LYONS
280 Henderson St.
Jersey City, NJ 07302

To use 16-bit size processors such as the TI9900 and the new generation of chips including the Intel 8086, Z8000 etc., in the S-100 bus, the proposed S-100 bus standards from the IEEE suggest combining the original 8-bit DATA IN and DATA OUT S-100 lines into a single bi-directional 16-bit wide data bus.

Within the same computer system, the same lines could also be used in the original, split eight bit mode, as long as only one kind of transfer were attempted at one time.

John Walker of Marinchip Systems describes a similar bi-directional system in the January, 1979 *Byte*, and his company actually supplies a TI9900 cpu card implementing that system on the S-100 bus, the first product of its kind.

Although the M9900 board is equipped with multiplexing circuitry to support existing 8-bit memories on the S-100 bus, it also supports the wider path and this is necessary if the increased power available from the bigger cpu is to be realized. At first glance this would appear to limit the potential of the new cpu as a means of enhancing existing S-100 bus systems, inasmuch as all the memory in a system would have to be replaced by a new memory organized as 16 bits wide and following the 16 bit transfer protocol—a much more expensive proposition than merely changing one component, the cpu. It turns out, however, that the regular memories might easily be converted to a 16 bit organization with a technique described here.

To begin with, memory boards are to be dealt with in pairs; a single board cannot be converted. The first step is to set the address allocation switches or jumpers on two memories so they are both assigned to the same place in memory and occupy a single block of contiguous bytes. To illustrate, suppose there were originally two 16K boards, A and B, assigned respectively to hex addresses 000-3FFF and 4000-7FFF; B would be reset to also occupy 0000-3FFF. As step two cut traces and install jumpers to rearrange the connections to address bus lines A0 and A14 (for 16K boards). Disconnect A0 where it enters the boards and instead connect A14 to that point. Where A14 originally entered the board tie the input to the logic level voltage that occurs when addresses are in the range of the board, i.e., the level required for board select to occur (ground in the example here).

With the A14 input thus tied active, the boards will be selected throughout the 32K byte range 0000-7FFF, which corresponds to the 16K word range 0000-3FFF. Each of the two boards which are simultaneously selected by 15-bit word addresses (A0 dropped) will hold one half the word and below be modified to use separate data lines. A14 fills in for the internal board function performed by A0 before; bytes in the memories are no longer mapped onto the address space sequentially. The first byte in memory corresponds to address 0000, the second to address 4000, the third to 0001, the fourth to 4001, and so on—which should not affect memory performance but would affect memory diagnostic programs.

As step three, separate the data lines of the two memories and convert to a bidirectional configuration. One board will use the former DATA IN bus lines exclusively while the other uses the former DATA OUT lines. Suppose board A is to use the OUT lines; the input lines are then already in place, and it remains only to disconnect the data output tristate buffers from the DATA IN lines and jumper their outputs over to the corresponding DATA OUT lines. The fact that the output from the memory is attached to the memory's own inputs has no effect upon it. On board B the output lines are already in place so just disconnect the DATA OUT lines where they enter the board and jumper those inputs over to the corresponding DATA IN lines instead.

Essentially that is all there is to a basic conversion. Some additional changes may be needed depending on the particular new cpu to be used. For instance, the above assumes that the new cpu will generate all the standard S-100 bus status and control signals during the 16-bit transfers. Individuals constructing their own cpu's may choose to meet that requirement, but whether any commercially available product does so is a question only the manufacturers can answer. At least one additional requirement, among an unknown number of others, has been described by Marinchip Systems—a signal to be pulled low by the memories to identify their 16-bit organization to the cpu, which handles both types of memories. Whether this requirement could be simply met by permanently grounding this new bus line, thereby avoiding more changes to memory boards, is not known.

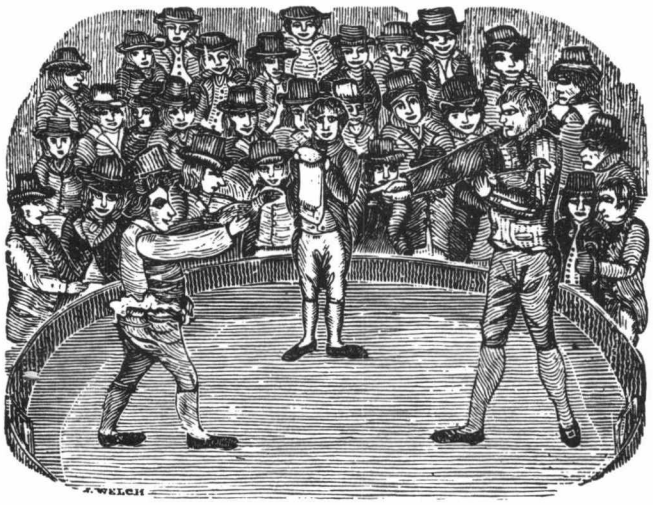
One generally applicable problem with the simple technique above is permanently fixing the memories in the 16-bit mode. They can thus no longer work with either 8-bit cpu's, or more importantly, 8-bit DMA devices for the S-100 bus (although there really are not many of those). This problem can be solved by elaborating the basic modification described to include a new input to the memories from the bus, MODE, which would make the memories operate in 16-bit mode when low and in 8-bit mode when high. To accommodate DMA devices MODE could simply be taken as PHLDA already on the bus. More complex systems could accommodate both 16-bit and 8-bit cpu's by having the cpu generate MODE as a combination of PHLDA and cpu type.

New parts must be added to memories to respond to MODE, which may not be practical with some units, since there are no solder pads provided for them and in some cases the original circuit's structure may be unfavorable to the new requirements. At least one memory, the Vandenberg Data Products 16K static, is ideally structured for fairly easy modification because it employs an 8212 latch for the DATA IN buffer and a tristate 74368 buffer on the DATA OUT lines. Instead of permanently disconnecting these data lines, it is then only necesary to control the enable input of the buffers with the MODE signal, and the presence of *two* enable inputs on the 8212 helps eliminate any need to provide additional gates to, say, combine MODE with signals internal to the board. To reconnect the data lines to the revised configuration a second buffer must be added to each original buffer, enabled by the complement of the original buffer's enable input, providing either an input selector or data distributor, as the case may be. These second add-on buffers might be installed by "piggy-backing" them on top of the existing ones, with which they share several common circuit connections anyway. The fact that all the new components needed on the Vandenberg unit can be mounted this way—and consist of only one 8212 and one 74368 for board A and two 74368's for board B—may make this approach of interest to do-it-yourselfers.

Full details for this memory, which incidentally, may no longer be manufactured, are available from the author; they are as yet untested. Note that in addition to the treatment of the DATA IN and OUT lines the address line inputs must be dealt with. A14 should continue to be permanently reconnected to the original A0 input, while MODE controls the input where A14 was originally connected; in 16-bit mode this is locked to enable both boards of the pair simultaneously, while in 8-bit mode A0 must determine the input at this point separately on boards A and B, enabling one when A0 is low and the other when A0 is high. Circuitry for this would be simplified if the cpu floated the A0 line while in 16-bit mode. Finally, in configuring the A0 control care must be taken to be consistent with the order in which the 16-bit cpu divides a word into two bytes, and whether A0 should be considered low for the low order byte of the word or should be considered high. Not all 16-bit cpu's might be consistent in this regard.

Should the dual 8-/16-bit mode elaboration prove impractical, all is not lost as far as DMA compatibility is concerned. A small area of memory, say 8K, might be left unconverted and thus available for DMA operations in 8-bit mode. The cpu could then copy data from that area to the location in 16-bit memory where the DMA was actually supposed to occur—at least a dual mode cpu like the Marinchip could do this. Granted, the full power of the DMA device would be lost, but at least it would still be accessible.

In closing, a basic principle of the modification technique, both fixed and dual mode, should be noted. The method assumes that the 16-bit cpu will address memory words with 15-bit addresses represented on the 16 S-100 bus address lines by the *top* fifteen lines, not the lower fifteen lines; A0 is reserved to distinguish bytes within words, and if generated by the 16-bit cpu, is to be ignored. Should conversion methods other than the one presented here be used, an entire system might be configured with only fifteen address lines, naturally taking the lowest fifteen on the bus; this should be avoided.



CONVERSATION
on the origins of Dr. Dobb's Journal

One recent winter afternoon, I sat down with Bob Albrecht, Dennis Allison and Rick Bakalinsky, the three people who started DDJ. *I wanted them to talk about the very early days of the magazine, and I had a few questions to ask, like: Where in the* world *did they get that name?*

Dennis Allison is a private consultant, a lecturer at Stanford University and a technical editor for Computer *magazine; Bob Albrecht is the infamous "dragon" of* Recreational Computing *née* People's Computers *fame, and the author of a seemingly limitless number of books, articles and pamphlets; Rick Bakalinsky is a man of many disguises, but at last sighting was a graphic artist.*

–SMR

Suzie: I'll get the ball rolling, I guess: how did *DDJ* get started?

Dennis: Well, it's a long story. Bob had been hassling me to write a book review, and I finally did it. When I brought it to him, he was reading a magazine. He said "look at this wonderful thing," and showed me the *Popular Electronics* issue that had the Altair on the cover. . .

Bob: January 1975, I think it was.

Dennis: We read through it and discovered that there was an *enormous* amount of memory provided with the machine–256 bytes, which is much too small to do anything useful. Bob said, well what are we going to do? These people are going to go out and buy this wonderful machine for $500 and it's going to revolutionize the world, but they won't be able to do anything with it. I said, well, I guess we could do a really small Basic. He said, well, why don't you write something? So we made a two-page thing about it to run in *People's Computers*. . .

Bob: Build Your Own Basic, in the March 1975 issue.

Dennis: Right, Build Your Own Basic, and somewhere in it, Bob said, well, what about a Tiny Basic–we'll call it Tiny Basic?

Bob: It starts out saying, suppose you're a kid about seven years old, you don't care about logarithms, sines and stuff like that, and we outlined just a little of a tiny, tiny Basic.

Dennis: So anyway, we put that together. Rick at that point was the entire production department at PCC . . . he was the best typist and layout person we had, and he was doing everything else too.

Bob: So we published that, and nothing happened, nobody responded. So we said we're going to drop it–we published that, saying we'd drop it. . .

Dennis: And then the letters came pouring in–dozens of letters.

Bob: Then you did the interpreter within the interpreter.

Dennis: Yeah. That was a single article, but it was a long article, a lot of stuff in it, and it really was almost a complete Tiny Basic but without the interpreter part written. We were going to write the interpreter; we had acquired an Altair and also a programmer named Bernard. Within a month of publication, Dick Whipple sent us a copy of a program run–which ran!

Bob: Whipple and Arnold.

Dennis: That's right, Whipple and Arnold. We wrote them back and got a listing of about 2,000 bytes of octal code which we published. The issue after that they sent us a complete listing and some documentation. But in the interim, having published this 2,000 byte octal program, we got a number of bug reports and corrections which said a substantial number of people had actually toggled the entire 2,000 bytes into the front panel, run it, and disassembled it to the point of understanding it so they could fix it! It was really quite spectacular. That was perhaps the first of a dozen Tiny Basics.

Bob: That toggling was . . . boggling! Then we said in an issue of *PC* that we were going to do a short term Tiny Basic newsletter which we would xerox–3 issues and that was it.

Dennis: Three issues for $3, a dollar an issue, that was it. We got 300 subscriptions immediately. The presumption was we ought to start a Journal, and that's what happened. At the end of the third issue, Jim Warren came on board. Jim was a graduate student at Stanford, and he became the first editor and carried on for quite a while.

Bob: In many ways, the style of the Journal has been consistent from Day Zero, and the original design was done by Rick, here, who named it.

Suzie: Let's talk about that, about the name.

Rick: I guess Bob and/or Dennis told me about their new journal and asked me if I would design it and think of a name for it. I knew absolutely nothing about computers, so I started asking around for some terms, trying to understand what they meant. I finally decided "byte" was a nice word. . .

Suzie: Oh, that's great.

Bob: That was in use though.

Suzie: Oh, byte:bite. So that's where orthodontia come from.

Bob: Of course.

Rick: I was thinking of a remedy, like a patent medicine, so "Doctor." At first I thought Dennis' name was Don, so I got "Dobb's" from Don and Bob combined . . . I thought it was perfectly symmetrical.

Dennis: Well, at least you got the "D" right.

Rick: Then the subtitle was "Running Light Without Overbyte." For the first few issues I had a picture of nude runner—running light—then I got a picture of Kirk Douglas displaying his perfect choppers, and I grafted his head onto the runner, so we had perfect dentures —without overbyte—running light (see box). What's funny is, initially we got several subscriptions from dentists.

Suzie: When I first took this job, my mother thought I was editor of a dental magazine.

Dennis: Yeah, there's a long series of letters in *Volume 1* about this dentist who bought *DDJ* because of the orthodonture thing and he got into computers—he *really* got into computers—fascinating.

Bob: People in this business are just amazing.

Rick: Anyway, after that I spent many hours xeroxing the first few issues and doing the layout.

Dennis: Well, the front cover idea is basically yours, the *DDJ* with the expanded Old English letters. . .

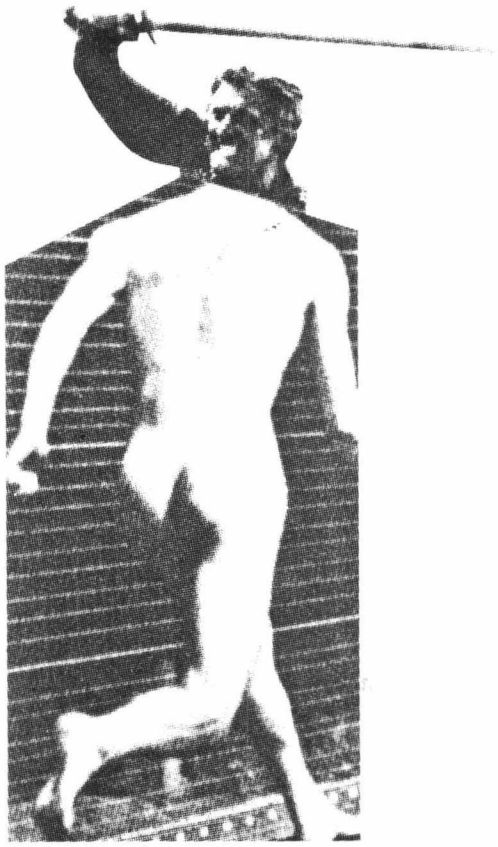
Suzie: Did you envision it would still be going strong three years later?

Bob: Not at all. We initially thought we'd do it for three issues and then it just took off.

Dennis: We decided after a while to have a real publication, and today *DDJ* is the largest-circulation magazine published by PCC. You showed me a letter the other day where somebody was comparing it to the old, relatively informal and unreviewed journals of some time ago, scientific journals. There's a lot of good work—good computer science. There's a lot of opinion which is something that many scientific journals don't publish, and a feeling of community which you don't get out of the heavily referred journals. It seems we have to put a good deal of effort into ensuring the *Journal* doesn't change its character and continues to serve that community. Our subscriber list is a very interesting one, ranging from high school kids to some of the most eminent computer scientists in the world. About 10-12% of the subscription base is outside of the United States.

Rick: You know, although I knew very little about computing at the time, I found it very exciting that so many people in the early days wrote *DDJ* to use it as a forum for sharing problems, sending in their own programs, finding bugs, and so on.

Suzie: They still do that.



Dennis: Well, Suzie, let's turn the tables. What do you think is going to happen to the magazine?

Suzie: I think basically stay the same, but I want to expand certain areas, involve the readers more. A Reader Exchange column, say, where readers can help each other. I think we should take a more active role in

consumer complaints—an action line, so to speak. I've had great response to that so far.

Dennis: What responses have you had to the questionnaire?

Suzie: As far as volume, it's been overwhelming. Basically, people like *DDJ* the way it is. Some want more of one thing, less of something else. Then there are just as many people who want the reverse: you know, six of one, half a dozen of the other.

Bob: I know the problem well.

Suzie: But definitely more tutorials, more reviews. I'm working on it.

But let's get back to the origins of *DDJ*. Have you noticed any change in the mood of people interested in small computers from three years ago? Since *DDJ* was started?

Dennis: Oh, yes. A very substantial change. For one thing, there's a much broader group of people involved. People who wouldn't have *dreamed* of buying computers three years ago are buying them now. Go into Kepler's (a Menlo Park bookstore) and they have a TRS-80 with games, and people are playing games, while other people are standing by writing little programs.

Bob: The Kepler's thing came out of a Communityquest meeting held at PCC. The idea was generated there. The underlying stategy is that—well, I feel that one of the next economically viable kinds of things will be, not a computer store, but a combination computer center and bookstore. Bookstore, software store, computer center where people can rent time on machines, try out software, buy software.

Dennis: Kepler's is an interesting example. You go into Kepler's and you discover that their computer science book selection is only *marginally* poorer than that of Stanford University.

Bob: Oh, yeah, it's a great idea. It's just getting started, but it's certainly something that should take off. It's a prototype of a bookstore, computer center bookstore. I heard that Nolan Bushnell (founder and former president of Atari, Inc.) is now split from Atari and is setting up a chain of pizza parlors with electronic games. So you're going to start seeing this sort of thing, computers where people hang out.

Dennis: The thrust of the computer store has changed substantially. Computer stores began to take care of the hobby computerist. But companies discovered they could buy parts from computer stores—albeit at a slight premium price—and get instant delivery. So the major business of computer stores turned out not to be the personal hobbyist, but businesses: IBM, Xerox, Varian, Smith Kline, etc., would go in and buy supplies at the local Byte Shop. Businesses would buy machines designed for the hobby market and put them to work in laboratories or accounting departments. More and more of the actual dollar volume of personal computing moved away from the original hobbyist to the low end of industry.

Bob: Is this what happened to computer fairs?

Dennis: Oh, yes.

Suzie: Well, how did you get involved in all this to begin with, Dennis?

Dennis: Another long story. I met Bob. He was doing some stuff for the ACM National Conference in San Francisco. So I saw him from time to time and when it came time to set up the People's Computer Company, which was his idea, he asked if I would be on the original board. I had very little to do with the actual production of the magazine.

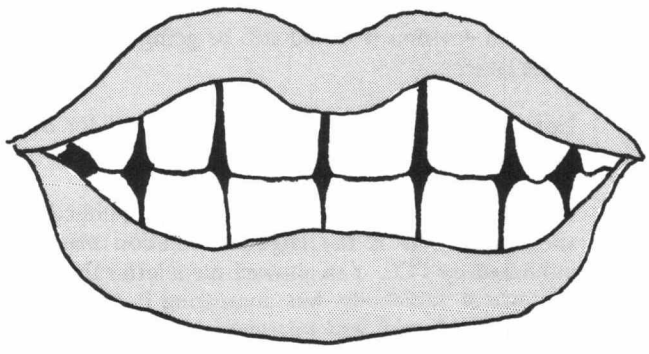
Suzie: Do you guys think people will use computers widely soon? Since they're getting cheaper and better all the time—computers, I mean.

Dennis: I don't know. I think it's a love/hate relationship for a lot of people with computers.

Rick: I think people who think you need a lot of math skills to run a computer shy away from them. Children find it easier.

Dennis: My students at Stanford—it's hard to tell the Master's students from the freshmen. The Master's or PhDs really enter from a different generation. The freshmen are much, much more knowledgeable. I was talking to a charming, charming freshman today, and it turns out that she's had three years of programming in Basic, with some experience in Cobol, Fortran and she's a freshman at Stanford!

Here the phone rang with a call for Bob. Dennis and Nancy began getting ready to go to The City to see a play. Suddenly it was all quite hectic and so we broke up and went our separate ways. Maybe there will be future conversations.



A Disk for Frits

BY PAC DALE PUCKETT
163 Farm Acre Road
Syracuse, NY 13210

Dear Dr. Dobb's:

I am sending you some software I know your 6800 fans will be glad to have. Enclosed you will find disk drivers which work with Frits Van Der Wateren's LISP for the 6800 (*DDJ* #28). They run in FLEX. They could however, be easily modified to run in any disc operating system. I could supply the object on source to anyone who sent a disc with return postage.

To write to a disc use the following sequence: (OPEN 3 FILENAME) (CR) (PRINT (NAME OF S-EXPRESSION) 3) (CR).

To read from a disk use the following: (OPEN 2 FILE-NAME) (CR) (READ 2) (CR).

As soon as I learn enough about LISP to write professional (SAVE and LOAD) functions in LISP I'll try to pass them along to *Dr. Dobb's.*

Frits Van Der Wateren should be thanked many times over for making such an exciting language available to novice micro owners. I find the language fascinating and quite powerful.

Hope your readers will find the enclosed helpful.

ADDENDUM

Dear Suzanne,

Thanks for your letter of January 20.

Here is some more information which will make the disk drivers I supplied earlier even more valuable to your 6800 readers.

First, in the LISP interpreter itself: Change HEX location 0376 from 09 HEX to 0A HEX. This change is necessary because the TERPRI function outputs both a carriage return and a line feed. This would be OK, but, LISP also recognizes a line feed as an atom and when you try to read a file that has been built using TERPRI you go into an infinite read loop. The change above will make it work fine.

After the disk drivers have been installed, and the enclosed files are located on a disk and LISP is in memory, the following steps will allow the user to bootstrap as many functions as necessary off of a disk.

* (OPEN 2 RESTORE)
* (EVAL (READ 2))
* (CLOSE 2)
* (RESTORE UTILITY)
* is the LISP prompt.

RESTORE IS THE FILENAME OF A FILE THAT CONTAINS THE FUNCTION . . . RESTORE. UTILITY IS THE FILENAME OF A FILE THAT CONTAINS THE FUNCTIONS . . . SAVE, DEFINE AND DEFINEQ. UTILITY also contains CHANGE and EQUAL. The latter two are not necessary to the bootstrap operation so they may be excluded if you are short on space.

Once the functions are in the LISP system and you generate a few more of your own you can save them by using the SAVE function. Here is the format:

* (SAVE function1 function2 function3 . . . function filename)

Note: filename is any length FLEX filename, i.e., up to eight characters. The drivers default to a .TXT extension.

The attached LISP code is a direct listing of the files, UTILITY, RESTORE and MORUTIL. ONLY THE FUNCTIONS: SAVE, RESTORE, DEFINE and DEFINEQ are essential to bootstrapping. The others may save a few readers some time.

Thanks for your interest in my LISP disk drivers. I think a lot of your 6800 FLEX users are going to have some fun with the language.

Sincerely,
Dale L. Puckett

```
UTILITY  FILE

    SAVE    (PUTPROP SAVE (LAMBDA (L A) (PROG (FILE FUNCTION TYPE) (SETQ
            FILE (CAR L)) (OPEN 3 FILE) LOOP (SETQ L (CDR L)) (COND
            ((NULL L) (GO DONE))) (SETQ FUNCTION (CAR L)) (SETQ TYPE
            (COND ((GET FUNCTION (QUOTE FEXPR)) (QUOTE FEXPR)) ((GET
            FUNCTION (QUOTE EXPR)) (QUOTE EXPR)) (T (GO LOOP)))) (PRINT
            (LIST (QUOTE PUTPROP) FUNCTION (GET FUNCTION TYPE) TYPE)
            3) (TERPRI 3) (PRINT (LIST FUNCTION (QUOTE SAVED))) (GO
            LOOP) DONE (PRINT NIL 3) (TERPRI 3) (CLOSE 3) (RETURN (QUOTE
            DONE)))) FEXPR)
    CHANGE  (PUTPROP CHANGE (LAMBDA (FROM TO IN) (COND ((EQUAL
            FROM IN) TO) ((ATOM IN) IN) (T (CONS (CHANGE FROM TO (CAR
            IN)) (CHANGE FROM TO (CDR IN)))))) EXPR)
    equal   (PUTPROP EQUAL (LAMBDA (A B) (COND ((EQ A
            B) T) ((ATOM A) NIL) ((ATOM B) NIL) ((EQUAL (CAR A) (CAR
            B)) (EQUAL (CDR A) (CDR B))))) EXPR)
    DEFINE  (PUTPROP DEFINE (LAMBDA (L A) (PUTPROP (CAR
            L) (CONS LAMBDA (CDR L)) EXPR)) FEXPR)
    defineq (PUTPROP DEFINEQ (LAMBDA (L A) (PUTPROP (CAR
            L) (CONS LAMBDA (CDR L)) FEXPR)) FEXPR)
            NIL

RESTORE  FILE

    RESTORE (PUTPROP (QUOTE RESTORE) (QUOTE (LAMBDA (FILE) (PROG (L)
            (OPEN 2 FILE) LOOP (SETQ L (READ 2)) (COND ((NULL L) (RETURN
            (QUOTE DONE)))) (PRINT (LIST (CAR (CDR L)) (QUOTE RESTORED)))
            (TERPRI) (SETQ L (LIST (CAR L) (LIST (QUOTE QUOTE) (CAR
            (CDR L))) (LIST (QUOTE QUOTE) (CAR (CDR (CDR L)))) (LIST
            (QUOTE QUOTE) (CAR (CDR (CDR (CDR L)))))) (EVAL L) (GO
            LOOP)))) EXPR)
            NIL


MORUTIL FILE
    REVERSE (PUTPROP REVERSE (LAMBDA (L) (COND ((ATOM L) L EXPR (LAMBDA))
            (T (APPEND (REVERSE (CDR L)) (LIST (CAR L)))))) EXPR)
    NOT     (PUTPROP NOT (LAMBDA (X) (COND ((NULL X)
            T) (T NIL))) EXPR)
    CADR    (PUTPROP CADR (LAMBDA (X) (COND (T (CAR (CDR
            X))))) EXPR)
    HAS     (PUTPROP HAS (LAMBDA (ATM LIS) (COND ((NULL
            LIS) NIL) ((EQ ATM (CAR LIS)) T) (T (HAS ATM (CDR LIS)))))
            EXPR)
            NIL

HAS determines whether a particular atom is in a list.  To use the function type:
(HAS (QUOTE SAM) NAMEOFLIST)

CADR returns the CAR of the CDR or the second item in a list.

NOT returns True if the statement is false.

REVERSE  completely reverses a list.

CHANGE allows you to change one atom within a list

EQUAL determines if an S-expression is equal.
```

Listing

```
                    NAM    LISPDRV
                * January 4, 1978
                * Routines to provide disk
                * drivers for LISP interpreter written
                * by Frits Van Der Wateren
                * and featured in Dr. Dobbs Journal,
                * Number 28.

                * By Dale L. Puckett
                * Chief Photojournalist
                * U. S. Coast Guard

                * Equates

7740            FCB     EQU    $7740
7806            FMS     EQU    $7806
7803            FMSCLS  EQU    $7803
7103            WARMS   EQU    $7103
7118            PSTRNG  EQU    $7118
712D            SETEXT  EQU    $712D
713C            RPTERR  EQU    $713C
002E            ARG2    EQU    $2E
06EB            TRUE    EQU    $06EB
0000            NIL     EQU    0

3D00                    ORG    $3D00

3D00 20 08      LISPD   BRA    OPNRED
3D02 01         VN      FCB    1
3D03            RDORWR  RMB    1
3D04            LSPNAM  RMB    2
3D06            FILNAM  RMB    2
3D08            XSAV    RMB    2

3D0A 37         OPNRED  PSH B  Save      B-reg
3D0B 86 01              LDA A  #1        Open for read
3D0D B7 3D 03           STA A  RDORWR    Set flag
3D10 8D 02              BSR    GETNAM    Move filename
3D12 20 46              BRA    OPNFIL    Go finish

3D14 DE 2E      GETNAM  LDX    ARG2      Name in LISP
3D16 FF 3D 04           STX    LSPNAM    Pointer
3D19 CE 77 44           LDX    #FCB+4    Filename in FCB
3D1C FF 3D 06           STX    FILNAM    Another pointer
3D1F C6 08              LDA B  #8        Filename count
3D21 FE 3D 04   RDNAM   LDX    LSPNAM
3D24 A6 00              LDA A  0,X       Get character
3D26 08                 INX              Bump
3D27 FF 3D 04           STX    LSPNAM    Update
3D2A FE 3D 06           LDX    FILNAM    Point to target
3D2D 81 29              CMP A  #$29      a ")"
3D2F 27 0B              BEQ    CONTIN    That's end of name
3D31 A7 00              STA A  0,X       else put in FCB
3D33 08                 INX              bump
3D34 FF 3D 06           STX    FILNAM    update
3D37 5A                 DEC B            Decrement count
3D38 26 E7              BNE    RDNAM     Loop till done
3D3A 20 07              BRA    FILEXT
3D3C 4F         CONTIN  CLR A            Clear extension
3D3D A7 00      CONT1   STA A  0,X
3D3F 08                 INX
3D40 5A                 DEC B
3D41 26 FA              BNE    CONT1

3D43 CE 77 4C   FILEXT  LDX    #FCB+12   Point to extension
3D46 6F 00              CLR    0,X
3D48 6F 01              CLR    1,X
3D4A 6F 02              CLR    2,X
3D4C CE 77 40           LDX    #FCB
3D4F B6 70 8C           LDA A  $708C     Get work drive number
3D52 A7 03              STA A  3,X       Put in FCB
3D54 86 01              LDA A  #1        Set extension as .TXT
3D56 BD 71 2D           JSR    SETEXT
3D59 39                 RTS

3D5A CE 77 40   OPNFIL  LDX    #FCB
3D5D B6 3D 03           LDA A  RDORWR    Get flag
3D60 A7 00              STA A  0,X       Put in FCB
3D62 BD 78 06           JSR    FMS       Do it
3D65 26 04              BNE    ERROR     Go on error
3D67 33                 PUL B            Restore B-Reg
3D68 7E 06 EB           JMP    TRUE      Report success to LISP

3D6B BD 71 3C   ERROR   JSR    RPTERR    Report it
3D6E BD 78 03           JSR    FMSCLS    Close files
3D71 CE 00 00           LDX    #0        Tell LISP you failed
3D74 33                 PUL B            Restore B-reg
3D75 39                 RTS

3D76 37         DISKIN  PSH B            Save B-reg
3D77 FF 3D 08           STX    XSAV      and X-reg
3D7A CE 77 40           LDX    #FCB
3D7D BD 78 06           JSR    FMS       Get char
3D80 26 05              BNE    EOFCHK    Go on error
3D82 FE 3D 08   RET10   LDX    XSAV      Restore X
3D85 33                 PUL B            and B-Reg
3D86 39                 RTS              return

3D87 E6 01      EOFCHK  LDA B  1,X       Get error code
3D89 C1 08              CMP B  #8        Is is EOF?
3D8B 26 18              BNE    RERROR    go if not
3D8D C6 04              LDA B  #4        If so,
3D8F E7 00              STA B  0,X       Close file
3D91 BD 78 06           JSR    FMS
3D94 26 0F              BNE    RERROR
3D96 CE 3D DD           LDX    #MSG1     report EOF error
3D99 BD 71 18           JSR    PSTRNG
3D9C FE 3D 08   RETLSP  LDX    XSAV      Restore X-reg
3D9F 33                 PUL B            and B-reg
3DA0 86 0D              LDA A  #$0D      a CR is a standard
3DA2 7E 01 9C           JMP    $019C     This is LISP.

3DA5 BD 71 3C   RERROR  JSR    RPTERR    Report it
3DA8 20 F2              BRA    RETLSP

3DAA 36         DISKOU  PSH A            Save A-reg
3DAB FF 3D 08           STX    XSAV      and X-reg
3DAE CE 77 40           LDX    #FCB
3DB1 BD 78 06           JSR    FMS       Send Char
3DB4 26 05              BNE    WERROR    go on error
3DB6 FE 3D 08   WRET10  LDX    XSAV      Else restore X
3DB9 32                 PUL A            and A-reg
3DBA 39                 RTS              then return.

3DBB BD 71 3C   WERROR  JSR    RPTERR    Go report it
3DBE 20 F6              BRA    WRET10    and return to LISP

3DC0 37         DISKCL  PSH B            Same routine whether
3DC1 CE 77 40           LDX    #FCB      reading or writing
3DC4 86 04              LDA A  #4        Get function code
3DC6 A7 00              STA A  0,X       put in FCB
3DC8 BD 78 06           JSR    FMS       and do it
3DCB 26 9E              BNE    ERROR
3DCD 33                 PUL B            restore B-reg
3DCE 7E 06 EB           JMP    TRUE      report success

3DD1 37         OPNWRT  PSH B            Save B-reg
3DD2 86 02              LDA A  #2
3DD4 B7 3D 03           STA A  RDORWR
3DD7 BD 3D 14           JSR    GETNAM
3DDA 7E 3D 5A           JMP    OPNFIL

3DDD 45         MSG1    FCC    'END OF FILE REACHED'
3DDE 4E 44
3DE0 20 4F
3DE2 46 20
3DE4 46 49
3DE6 4C 45
3DE8 20 52
3DEA 45 41
3DEC 43 48
3DEE 45 44
3DF0 04                 FCB    4
```

Continued on pg. 39

OSI BASIC for the KIM-1

BY JONATHAN M. PRIGOT
35 Cummings Road, #6
Boston, Massachusetts 02146

For some time, now, if a KIM-1 user wished to run Microsoft's 8K BASIC on their system, they had their choice of two cassettes. Unfortunately, since both versions seem to use self-modifying code, if the KIM crashes, then the tape must be reloaded. Putting it into ROM would prevent this, but, of course, this cannot be done.

Ohio Scientific Inc. (OSI) makes a line of 6502-based machines that contain their BASIC in four ROMs. If these ROMs could be used on other 6502 systems, the effects of a system crash would be less catastrophic. The problem, however, is to link the ROM BASIC with other machines. The purpose of this article is to present a way to do just that.

The OSI chip set consists of five chips: four 2316 ROMs which contain the 8K BASIC and reside at A000-BFFF, and a 1702 "support PROM" which resides at FF00-FFFF, and contains initialization routines, jumps to I/O routines, and the 6502's NMI, RST, and IRQ vectors. These support routines must be customized for the system the BASIC is to run on.

Some Important Addresses

```
0000-00FF: Zero page pointers and routines (USR vector=$B,$C
                                                    [L,H])
0100-01FF: Stack Area
0200-02FF: Normally used by OSI's I/O routines, otherwise
                                                    unused
0300-up  : Source file storage
A000-BFFF: BASIC ROMs
FF00-FFFF: OSI Support ROM which contains the following jumps:
FFEB-FFED: (4C) (Console In Low) (Console In High)
FFEE-FFF0: (4C) (Console Out Low) (Console Out High)
FFF1-FFF3: (4C) (Ctl/C Handler Low) (Ctl/C Handler High)
FFF4-FFF6: (4C) (Cassette Load Low) (Cassette Load High)
FFF7-FFF9: (4C) (Cassette Save Low) (Cassette Save High)
```

Console I/O Routines

The Console In and Out routines must preserve the Y-register. The BASIC also assumes the use of a full-duplex system. If your system, like the KIM, is half-duplex, then the echoed character must be suppressed. Both these routines must end with an RTS (60).

The action of the Control-C Handler depends on the action of the Console In routine. If the Console In routine does not halt and wait for a character input, then simply JMP to $A633. The BASIC will then input a character and test it. If the Console In routine does wait for input, then some means of polling the input port for a character must be used. Upon detection of a character, it must be passed to the BASIC by a JMP to $A636.

Cassette Routines

I decided to use the KIM's resident cassette save/load routines due to their accuracy. Unfortunately, both these routines exit to location $0000. Re-writing these routines to end with an RTS (60) so as to return to the BASIC would take too much space. I therefore decided to set up the save/load routines for two phases each. Phase one gets a file number (01-FE) from the user and saves/loads the BASIC's pointers to the source file. Phase two saves/loads the actual source file.

Since the save/load routines exit to location $0000, each phase sets up this location to point to the next phase. After each phase exits, one simply enters a "G" to go to the next phase. Phase two sets up a jump to the BASIC's warm-start point.

Source File Storage Relocation

The BASIC assumes uninterrupted memory from $0000 on up. If the bulk of your RAM is located elsewhere (like $2000-up), then the BASIC's pointers must be changed to reflect this. For example, if we assume a 4K RAM at $2000-$2FFF, then the locations to change are:

```
$79,$7A-Start of RAM + 1 (01,20)
$7B,$7C-Start of RAM + 4 (04,20)
$7D,$7E-  "    "  "   "      "
$7F,$80-  "    "  "   "      "
$81,$82-End of RAM + 1 (00,30)
$85,$86-  "    "  "   "      "
$8F,$90-Start of RAM (00,20)
$C3,$C4-  "    "  "      "
```

In addition, the first three locations in the text area must be set to zero. Initialize, do a warm-start, and type NEW. This resets all the links to the source file.

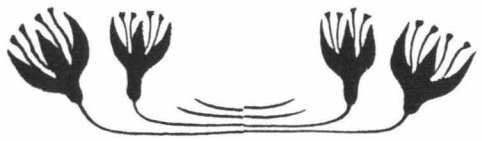
I have found that the most useful way to use this program is to set the KIM's NMI vector to point to this monitor. All that is then necessary is to press the ST key to activate the monitor.

```
                        0010  ;SIMPLE OSI/MICROSOFT BASIC SUPPORT MONITOR
                        0020  ;FOR THE KIM-1, VERSION 1.1
                        0030  ;WRITTEN 12/29/78 BY J.M. PRIGOT
                        0040  ;ALL RIGHTS RESERVED
                        0050  ;
                        0060        .OR FE00
                        0070  ;INITIALIZE BY CLEARING BCD FLAG, AND SETING
                        0080  ;STACK POINTER TO $0128
                        0090  ;
FE00  D8        0100  STAR CLD
FE01  A228      0110        LDX 28
FE03  9A        0120        TXS
                        0130  ;
                        0140  ;CLEAR SCREEN AND PRINT MENU, THEN GET USER'S
                        0150  ;CHOICE AND BRANCH APPROPRIATELY.
                        0160  ;
FE04  AD35FE    0170        LDA HEDR          ;LOAD TEXT POINTER
FE07  85EC      0180        STA *TEXT
FE09  AD36FE    0190        LDA HEDR+01
FE0C  85ED      0200        STA *TEXT+01
FE0E  20C9FE    0210        JSR PMPT          ;PRINT MENU
FE11  20E6FE    0220        JSR GETC          ;GET USER'S CHOICE
FE14  20FAFE    0230        JSR ZERO          ;ZERO Y-REG. AND ECHO-SUPPRESS FLAG
                        0240  ;
FE17  C94D      0250        CMP 'M            ;MONITOR?
FE19  D003      0260        BNE WARM
FE1B  4C001C    0270        JMP $1C00         ;JUMP TO KIM MONITOR
FE1E  C957      0280  WARM  CMP 'W            ;WARM-START THE BASIC?
FE20  D003      0290        BNE COLD
FE22  4C0000    0300        JMP $0000         ;BASIC WARM-START ENTRY
FE25  C943      0310  COLD  CMP 'C            ;COLD-START THE BASIC?
FE27  D0D7      0320        BNE STAR          ;INVALID ENTRY, SO RESTART
FE29  4C11BD    0330        JMP $BD11         ;BASIC COLD-START ENTRY
                        0340  ;
FE2C  1AFF      0350  MENU  .HS 1AFF          ;CLEAR SCREEN AND DELAY
FE2E  432F57    0360        .AS 'C/W/M?'      ;MESSAGE TEXT
FE31  2F4D3F
FE34  00        0370        .HS 00            ;END-OF-MESSAGE INDICATOR
FE35  2CFE      0380  HEDR  .SA MENU          ;POINTER TO TEXT
                        0390  ;
                        0400  ;CONTROL-C HANDLER
                        0410  ;
FE37  2C4017    0420  CNTC  BIT $1740         ;POLL TTY INPUT PORT
FE3A  3005      0430        BMI SKIP          ;IF NOTHING THERE, LEAVE
FE3C  A903      0440        LDA 03            ;LOAD CTL/C CHARACTER,
FE3E  4C36A6    0450        JMP $A636         ;AND JUMP TO BASIC HANDLER
FE41  60        0460  SKIP  RTS
                        0470  ;
                        0480  ;CASSETTE SAVE/LOAD ROUTINES
                        0490  ;
FE42  209AFE    0500  SAVE  JSR FILE          ;GET/STORE FILE NUMBER
FE45  A979      0510        LDA 79            ;STORE POINTER AREA ADDRESSES
FE47  8DF517    0520        STA $17F5
FE4A  A991      0530        LDA 91
FE4C  8DF717    0540        STA $17F7
FE4F  A900      0550        LDA 00
FE51  8DF617    0560        STA $17F6
FE54  8DF817    0570        STA $17F8
FE57  AD7FFE    0580        LDA SVAD          ;REDIRECT WARM-START VECTOR
FE5A  8501      0590        STA $01           ;TO PHASE-2 OF SAVE ROUTINE
FE5C  AD80FE    0600        LDA SVAD+01
FE5F  8502      0610        STA $02
FE61  4C..18    0620        JMP $1800         ;STORE POINTER AREA ON CASSETTE
                        0630  ;
FE64  D8        0640  PH2S  CLD               ;SAVE ROUTINE PHASE-2
FE65  A58F      0650        LDA $8F           ;LOAD SOURCE FILE POINTERS
FE67  8DF517    0660        STA $17F5
FE6A  A590      0670        LDA $90
FE6C  8DF617    0680        STA $17F6
FE6F  A57F      0690        LDA $7F
FE71  8DF717    0700        STA $17F7
FE74  A580      0710        LDA $80
FE76  8DF817    0720        STA $17F8
FE79  20C0FE    0730        JSR RSTR          ;RESTORE THE WARM-START VECTOR
FE7C  4C0018    0740        JMP $1800         ;SAVE THE SOURCE FILE ON CASSETTE
                        0750  ;
FE7F  64FE      0760  SVAD  .SA PH2S          ;POINTER TO SAVE PHASE-2 ROUTINE
                        0770  ;
                        0780  ;
FE81  209AFE    0790  LOAD  JSR FILE          ;GET FILE NUMBER
FE84  AD98FE    0800        LDA LOAD          ;REDIRECT WARM-START VECTOR
FE87  8501      0810        STA $01           ;TO PHASE-2 OF LOAD ROUTINE
FE89  AD99FE    0820        LDA LOAD+01
FE8C  8502      0830        STA $02
FE8E  4C7318    0840        JMP $1873         ;RETRIEVE POINTERS FROM CASSETTE
                        0850  ;
FE91  D8        0860  PH2L  CLD               ;LOAD ROUTINE PHASE-2
FE92  20C0FE    0870        JSR RSTR          ;RESTORE THE WARM-START VECTOR
FE95  4C7318    0880        JMP $1873         ;RETRIEVE THE SOURCE FILE
                        0890  ;
FE98  91FE      0900  LOAD  .SA PH2L          ;POINTER TO LOAD PHASE-2 ROUTINE
                        0910  ;
FE9A  D8        0920  FILE  CLD               ;GET/STORE FILE NUMBER ROUTINE
FE9B  ADBEFE    0930        LDA GETF          ;LOAD TEXT POINTERS
FE9E  85EC      0940        STA *TEXT
FEA0  ADBFFE    0950        LDA GETF+01
FEA3  85ED      0960        STA *TEXT+01
FEA5  20C9FE    0970        JSR PMPT          ;PRINT MESSAGE
FEA8  20F2FE    0980        JSR GETB          ;GET THE FILE NUMBER
FEAB  8DF917    0990        STA $17F9         ;STORE IN KIM CASSETTE FILE LOC.
FEAE  60        1000        RTS
                        1010  ;
FEAF  0D0A      1020  FLNM  .HS 0D0A          ;CR/LF
FEB1  46494C    1030        .AS 'FILE NUMBER='
FEB4  45204E
FEB7  554D42
FEBA  45523D
FEBD  00        1040        .HS 00
FEBE  AFFE      1050  GETF  .SA FLNM          ;POINTER TO TEXT
                        1060  ;
FEC0  A974      1070  RSTR  LDA 74            ;RESTORE WARM-START VECTOR
FEC2  8501      1080        STA $01
FEC4  A9A2      1090        LDA 0A2
FEC6  8502      1100        STA $02
FEC8  60        1110        RTS
                        1120  ;
                        1130  ;CONSOLE I/O ROUTINES
                        1140  ;
FEC9  20FAFE    1150  PMPT  JSR ZERO          ;ZERO Y-REG. AND ECHO-SUPP. FLAG
FECC  B1EC      1160  FTCH  LDA (TEXT),Y     ;GET A CHARACTER
FECE  F006      1170        BEQ DONE          ;IF END-OF-MESSAGE, LEAVE
FED0  20D7FE    1180        JSR PRNT          ;PRINT THE CHARACTER
FED3  C8        1190        INY               ;INCREMENT THE POINTER
FED4  D0F6      1200        BNE FTCH          ;GET NEXT CHARACTER
FED6  60        1210  DONE  RTS
                        1220  ;
                        1230  ;
FED7  84F4      1240  PRNT  STY *YSAV         ;SAVE Y-REGISTER
FED9  A4EE      1250        LDY *PFLG         ;GET THE ECHO-SUPPRESS FLAG
FEDB  D003      1260        BNE RSET          ;IF SET, DO NOT PRINT
FEDD  20A01E    1270        JSR $1EA0         ;OTHERWISE, PRINT THE CHARACTER
FEE0  20FAFE    1280  RSET  JSR ZERO          ;RESET ECHO-SUPPRESS FLAG
FEE3  A4F4      1290        LDY *YSAV         ;RESTORE Y-REGISTER
FEE5  60        1300        RTS
                        1310  ;
                        1320  ;
FEE6  84F4      1330  GETC  STY *YSAV         ;SAVE Y-REGISTER
FEE8  A0FF      1340        LDY 0FF           ;SET THE ECHO-SUPPRESS FLAG
FEEA  84EE      1350        STY *PFLG
FEEC  205A1E    1360        JSR $1E5A         ;GET CHARACTER FROM TERMINAL
FEEF  A4F4      1370        LDY *YSAV         ;RESTORE Y-REGISTER
FEF1  60        1380        RTS
                        1390  ;
                        1400  ;
```

```
                    * The third pointer opens a file on a
                    * file oriented device
                    * The fourth closes a file on a file
                    * oriented device.
                    * Location $02C5 is a RTS

                    * Device 2 first

010E                        ORG     $010E
010E 3D 76                  FDB     DISKIN
0110 02 C5                  FDB     $02C5
0112 3D 0A                  FDB     OPNRED
0114 3D C0                  FDB     DISKCL

                    * Device Three

0116 02 C5                  FDB     $02C5
0118 3D AA                  FDB     DISKOU
011A 3D D1                  FDB     OPNWRT
011C 3D C0                  FDB     DISKCL

                    * Device Four
                    * pointers to my IBM routines

011E 02 C5                  FDB     $02C5
0120 3E B0                  FDB     $3EB0
0122 3E 80                  FDB     $3E80
0124 02 C5                  FDB     $02C5

                    * Device Five is set up
                    * to return to FLEX

0126 71 03                  FDB     $7103
0128 71 03                  FDB     $7103
012A 71 03                  FDB     $7103
012C 71 03                  FDB     $7103

0140                        ORG     $0140
0140 12 78                  FDB     $1278

0149                        ORG     $0149
0149 3C FE                  FDB     $3CFE

                            END

NO ERROR(S) DETECTED

* Device 1 is terminal
* Its pointers remain the
* same.  We change the
* pointers to the other devices
* in the following code.

* The first pointer returns a character in A.
* The second pointer takes a character from A

   SYMBOL TABLE:
ARG2   002E   CONT1  3D3D   CONTIN 3D3C   DISKCL 3DC0   DISKIN 3D76
DISKOU 3DAA   EOFCHK 3D87   ERROR  3D6B   FCB    7740   FILEXT 3D43
FILNAM 3D06   FMS    7806   FMSCLS 7803   GETNAM 3D14   LISPD  3D00
LSPNAM 3D04   MSG1   3DDD   NIL    0000   OPNFIL 3D5A   OPNRED 3D0A
OPNWRT 3DD1   PSTRNG 7118   RDNAM  3D21   RDORWR 3D03   RERROR 3DA5
RET10  3D82   RETLSP 3D9C   RPTERR 713C   SETEXT 712D   TRUE   06EB
VN     3D02   WARMS  7103   WERROR 3DBB   WRET10 3DB6   XSAV   3D08
```

Continued from pg. 36

```
FEF2  84F4      1410  GETB STY *YSAV          #SAVE Y-REGISTER
FEF4  209D1F    1420       JSR $1F9D          #GET BYTE FROM TERMINAL
FEF7  A4F4      1430       LDY *YSAV          #RESTORE Y-REGISTER
FEF9  60        1440       RTS
                1450  ;
                1460  ;
FEFA  A000      1470  ZERO LDY 00             #ZERO Y-REGISTER
FEFC  84EE      1480       STY *PFLG          #RESET ECHO-SUPPRESS FLAG
FEFE  60        1490       RTS
                1500  ;
                1510  #SYSTEM EQUATES
                1520  ;
                1530  TEXT .DL 00EC
                1540  PFLG .DL 00EE
                1550  YSAV .DL 00F4
                1560  ;
                1570  #JUMP VECTORS FOR BASIC
                1580  ;
                1590       .OR FFEB
FFEB  4CE6FE    1600  GETV JMP GETC
FFEE  4CD7FE    1610  PUTV JMP PRNT
FFF1  4C37FE    1620  CTLC JMP CNTC
FFF4  4C81FE    1630  LODV JMP LOAD
FFF7  4C42FE    1640  CSAV JMP SAVE
                1650  ;
                1660  #6502/KIM VECTORS
                1670  ;
FFFA  1C1C      1680  NMIV .HS 1C1C
FFFC  221C      1690  RSTV .HS 221C
FFFE  1F1C      1700  IRQV .HS 1F1C
                1710       .EN

SYMBOL TABLE
STAR    FE00
WARM    FE1E
COLD    FE25
MENU    FE2C
HEDR    FE35
CNTC    FE37
SKIP    FE41
SAVE    FE42
PH2S    FE64
SVAD    FE7F
LOAD    FE81
PH2L    FE91
LDAD    FE98
FILE    FE9A
FLNM    FEAF
GETF    FEBE
RSTR    FEC0
PMPT    FEC9
FTCH    FECC
DONE    FED6
PRNT    FED7
RSET    FEE0
GETC    FEE6
GETB    FEF2
ZERO    FEFA
TEXT    00EC
PFLG    00EE
YSAV    00F4
GETV    FFEB
PUTV    FFEE
CTLC    FFF1
LODV    FFF4
CSAV    FFF7
NMIV    FFFA
RSTV    FFFC
IRQV    FFFE

:
```



Increase your Poly 88 Pleasure

BY DAVID L. JOHNSON
4106 Montreal Avenue
Prince George, Virginia 23875

Tired of interrupting your train of thought to stop and load yet another short program? Combine those programs into one big one.

The following idea should prove useful to owners of Poly 88 computers who use the Poly cassette interface, A00 or A01 BASIC, and the 4.0 monitor ROM.

When I first got my Poly, I had only 5656 bytes of free memory, so my programs tended to be short. Now I have almost 14K of free memory and find it useful to load several related programs at one time. For example, I have one series of five number guessing games; I have another of word and letter games. I also have one program of number conversions and various memory dump routines. The desired application can be chosen by use of a simple menu at the beginning of the program series. The menu provides a GOTO or GOSUB to the routine of choice. In most cases I just STOP a program when finished with it and then type RUN again to get back to the menu. If I had initialized all variables at the beginning of all programs, this would not be necessary. I could then just jump back to the menu each time. But in Poly A00 BASIC all variables are initialized when RUN is used, so I didn't always bother to do my own initialization when I originally wrote the short programs.

The idea of having several programs or routines loaded at once has been my desire for a long time. But I refused to sit and retype all the programs just to achieve the desired convenience. The problem is that Poly BASIC apparently executes a SCRatch when a new program is LOADed. I didn't and still don't know how to get around that. But I have found a roundabout way to accomplish what I wanted. It involves using the Poly printer driver software called BPRINT. What I have done is modified BPRINT to write to and read from tape instead of to and from a terminal device. To combine multiple programs into one program in memory, do exactly as this procedure says. It's not at all difficult.

1. Load A00 or A01 BASIC and BPRINT as outlined in your manuals.
2. Change 5 bytes in BPRINT as follows:

Location in A00	(in A01)	Byte is	Change to	Comment
4A7D	(4B81)	3E	C9	RET. Do not put out any pad characters.
4A98	(4B9C)	16	06	Changes output port to 0, normally the cassette port.
4A9B	(4B9F)	DA	CE	Changes setup of USART for tape.
4AEF	(4BF3)	0D	7F	Check for DEL instead of carriage return. The USART puts out DELs when it has no character to send and special carriage return processing is not needed anyway.
4AF1	(4BF5)	C1 C5	AC B0	Jump to CEXIT instead of to CCR.

3. Load one of the BASIC programs you wish to combine.
4. Make the first 15 lines of the program REMarks, like 1REM, 2REM, etc. RENumber as necessary to free up the first 15 line numbers for this.
5. RENumber 1,1 to make the line numbers throughout the entire program as small as possible.
6. Type PRINT CHR$(17) to turn on the "printer," which in this case is the USART to the cassette interface.
7. Type LIST (do *not* press the RETURN key yet).
8. Start a blank tape to recording.
9. After ten seconds press the RETURN key.

Continued on pg. 42

Product Evaluation: PERCOM LFD-400

BY DOUGLAS K. BECK
995 Lundy Lane
Los Altos, CA 94022

Z80/S100/CP-M folk have it rather simple. All the choices have been made for them with respect to hardware and operating systems. Nonconformists have a little more trouble. For instance, the SWTP-6800 has at least three disc drive/operating system combinations available. To weigh the potential and capabilities of any of them is difficult at best. After living with my SWTP-6800 for a year it became obvious that the operating system and supporting hardware would have to be expanded far beyond the original cassette-based equipment.

My major concern was to provide myself with more time to work on projects. Software support and a modular approach to system configuration also received strong consideration. The time element came from the fact that tape entry with a binary format could not bring the load time for BASIC or the Assembler below four minutes.

I initially considered the SWTP MF-68 disc system; however the price of the dual drive plus an additional 8K memory board was more than a hobby budget could absorb all at once.

About this time PERCOM Data Co. announced the LFD-400. It was a single drive system (expandable to three drives) which had a ROM based operating system featuring named files called MINI-DOS Plus X. Even though a single drive is a bit limiting for some processing operations, it looked promising, and I could use an existing 4K memory board to support the DOS.

Other software support was lean but growing, and most of my current programs could be reconfigured to accomodate the disc. To further check my initial impression I bought the LFD-400 manual and spent several evenings studying the listings and procedures shown in it. The hardware portion was quite simple and made use of a hard sectored disc format, 10 sectors per track, 35 tracks, 256 bytes per sector for a capacity of 89,600 bytes.

Percom specifies 3 week delivery and was as good as its word. Installation was quite simple, requiring you to bolt the disc in its housing and install the interface card in the C.P.U. chassis. No diagnostic procedures are supplied with the disc so verification is a matter if making in-service checks.

Percom supplies some fine patches to adapt such programs as SWTP BASIC (2.0 or 2.2) to their disc system. As an "all or nothing" test I loaded BASIC and followed the save to disc procedure. After a cold start the program was read in again and tested.

The Percom patches (now available as the "Bandaid" patch) were applied and I found that BASIC was now a disc file handling program with substantially more power than before. It also loads in two seconds, thus reaching my goal of program load time savings.

Since then the Percom patches to the T.S.C. Text Editor and Text Processor have been applied with equal success. Percom people have done no violence to good software.

Percom offers their own Symbolic Assembler. The Percom Assembler permits you to specify assembly options from a parameter line rather than in the header of the program. This more closely follows current practice in permitting an assembly to be run without having to re-edit the source code each time you want a variation in the listing procedure. The T.S.C. Editor with the "Touch Up" patch applied provides the nucleus for disc copy procedures that permit disc-to-disc transfers without the need for a second disc drive.

Percom provides a number of transient and utility programs. The utilities consist of:

- A Hex Loader which creates a binary memory load from the 'S1' format assembler output.
- Print, which lists the contents of ASCII files to a hardcopy device or to console.
- A Disc Map program permitting you to examine the contents of a disc sector.

The transient programs are:

- Copy, which provides the ability to copy disc-to-disc or from one sector to another on the same disc. It also provides the means for copying from a sector numbered file to a named file with a directory listing.
- Pack, a routine which provides the means for recovering unused space on a disc.
- Backup, which creates mirror image disc-to-disc copies to preserve important files.
- Create, which will create a file named in the directory having arbitrary length, to preserve a block of disc space.

Each Transient or Utility program comes with annotated source code, as does the Mini-DOS Plus X file handler and Mini-DOS disc driver.

Mini-DOS Plus X (TM) provides the essential commands through a series of single letter mnemonics. Mini-DOS Plus X supports up to 31 named files per disc. My fullest disc contains under twenty files and is the backup for the entire system. System Commands are:

Continued on pg. 42

Continued from pg. 40

10. After all of the program has listed on the screen (at 30 characters per second) wait at least ten more seconds and then stop the tape recorder.
11. Type PRINT CHR$(19) to turn off the "printer" port. This is very important. If you fail to do this, you will have to restart BASIC as explained in your BPRINT manual.
12. LOAD the program to be combined with the first program.
13. RENumber this program so the first line numbers falls *above* the highest line number in the previous program, e.g., use REN 8000,1 for safety.
14. Rewind the tape you LISTed on in steps 8 through 10.
15. Type PRINT CHR$(17). Type PRINT CHR$(18). This enables the "keyboard" or in this case, the playback facility.
16. Play back the program you LISTed on tape in steps 8 through 10. You may get some extraneous characters at first, but soon the computer should synchronize on the 15 REMarks you typed at the beginning of the program and then read in correctly thereafter.
17. After the entire program is LISTed on the screen, stop the recorder. If there are extraneous characters following the last line, type CNTL-X to erase them.
18. Type PRINT CHR$(20). Type PRINT CHR$(19). This disables the "printer" and the "keyboard" facilities.
19. You now have both programs in memory. You can replace the unwanted REMarks at the beginning of the program with the menu or guide to the two programs. You can also RENumber as desired and SAVE the combined programs in the normal way.

Of course, the above procedure can be repeated from step 3 as many times as you like (until you run out of memory) to combine as many programs as you like.

While it will take several minutes to combine your programs this way, you will find it much quicker than having to retype even one lengthy program. And once it's done, you will find you don't have to spend nearly as much time sorting and loading cassettes as you have in the past, since several programs related to the one you're using will already be loaded and instantly available to you. Here's one last thought. As your programs get longer, you should try the LOAD and SAVE speed-up routines mentioned in the Poly Newsletter which is available from Roger Lewis, 1477 Barrington, Suite 17, Los Angeles, CA 90025. The techniques work great and allow normal SAVE, LOAD, and dump operations at around 15K bytes per minute using a medium to high quality tape recorder. Using the higher speed, A00 BASIC loads in 42 seconds instead of the normal 75 seconds.

Continued from pg. 41

- L(oad) which reads the desired program from disc into memory and returns control to the DOS. Invoking the program name such as 'BASIC' will bring the program into memory and begin execution.
- S(ave) creates a directory entry by name for a program with specified memory limits and execution address. Files are identified in the disc header as to type (binary, BASIC, Editor, Assembler output, etc.) and may also be assigned a protected attribute by prefixing the commercial 'at' (@) symbol to the (up to six character) file name.
- R(ename) permits changes in file name and also a method of changing the protect status of the file.
- X(it) returns control to the ROM monitor.
- A(ppend) provides for allocation of 10 additional sectors to a file allowing room for expansion in the future.
- J(ump) causes an unconditional transfer to the entered location. This command permits the execution of programs without the need to return to the ROM monitor.
- M(inidos) will transfer control to the most primitive portion of the operating system which permits you to operate on files by drive, track and sector number.
- F(iles) creates a listing of all files in the directory.
- I(nitialize) clears the first ten sectors of a disc to provide room for the directory.
- D(elete) deletes the named file and makes the space available to the preceding file.

I doubt that an ideal operating system has ever been constructed. Certainly some inefficiency can be traded for a subjective friendliness or operator tolerance. The Percom system appears to have that quality. Certainly the Percom LFD-400 system has a good spot to fill where a modular approach to building a disc based operating system is desired.

Its main weakness lies in that it is not yet a true named file system. At some point in most procedures you must revert to referencing your file by its drive, track and sector number. This is a consequence of having to support the most restricted version of the disc system. It does appear to have the capability of conversion to a pure named file system if my reading of the listings is correct.

In the remainder of the Percom software there have been no surprises. Each patch or program worked as advertised.

Letters

WINTER FUN IN CHICAGO

Dear People,

Why is 'TUFIVE' or six an improvement over hexadecimal? Quick, off the top of your head, what is 57 base 32, in hex? Five bits into 8 leaves 3, into 16 with one left over isn't bad, now with six bits. Eight remainder of two, sixteen with a remainder of four, not so good. As long as we're at it let us have TUSIXTYFOUR and all those of us who don't have an IBM can approximate the results with an analogue machine.

For one of the larger machines Mr. Simons might have a point. However I think for a small machine unless the user is really desperate for space or something to do TUFIVE will be ignored until perhaps they wish to write you in code.

Now if all this snow will melt and it gets up to thirty degrees and the streets get eventually plowed maybe I will have something to do rather than figuring out things like this article.

Yours truly, Chicago, Illinois
L. Barker

```
NUMBER? FIVE
BASE? 32
TO BASE? 10
510958
ANOTHER? Y

NUMBER? 510958
BASE? 10
TO BASE? 32
FIVE
ANOTHER? Y

NUMBER? NUMBER
BASE? 10
TO BASE? 70
N is not base 10

NUMBER? NUMBER
BASE? 64
TO BASE? 10
Overflow in conversion.

NUMBER? 65535
BASE? 10
TO BASE? 16
FFFF
ANOTHER? Y

NUMBER? FFFF
BASE? 16
TO BASE? 8
177777
ANOTHER? Y

NUMBER? 177777
BASE? 8
TO BASE? 10
65535
ANOTHER? Y

NUMBER? 240
BASE? 10
TO BASE? 2
11110000
ANOTHER? Y

NUMBER? SIX
BASE? 64
TO BASE? 16
1C4A1
ANOTHER? Y

NUMBER? 1C4A1
BASE? 16
TO BASE? 25
7A9N
ANOTHER? Y

NUMBER? 7A9N
BASE? 25
TO BASE? 64
SIX
ANOTHER? Y

NUMBER? 10
BASE? 16
TO BASE? 8
20
ANOTHER? Y

NUMBER? FFFFF
BASE? 16
TO BASE? 2
11111111111111111111
ANOTHER? Y

NUMBER? 241
BASE? 2
TO BASE? 1
Base 2 through base 70 only.

NUMBER? 12345
BASE? 10
TO BASE? 64
30p
ANOTHER? N
DISK RUNNING? Y
```

IN RESPONSE TO DAVID CHAPMAN

Dear Editor:

David Chapman's article, "Programming Languages and Standards" (*DDJ* 30) contains a flagrant misunderstanding: "[Pascal's limited control structures] make the source code easier to understand and debug, but [they] restrict the programmer and may discourage creativity."

Programming languages are aids to the programmer. They allow algorithms to be expressed in symbols like those we already know. If a language makes the program "easier to understand and debug," then it's doing its job: no more, no less. Pascal's control structures may not be ideal, but they are relatively simple, powerful and easy to implement. Criticizing a language for having such structures is like criticizing a piano for having keys.

"Creativity" is an overused word. If it means the generation of new things, especially ideas, then it is best exercised during the design of a program before coding is done. If a programming language truly restricts a programmer's creativity, then he has waited too long to do his thinking.

Sincerely, 734 S. W. Westwood Drive
David Rowland Portland, OR 97201

SS-50 BUS BETTER

Dear Editor!

In the last issue of your *Journal,* which I enjoy very much, author K. Britton published a "Critical Look at a Proposed New Standard Bus for Microprocessors." I would like to make some comments on this topic—despite the fact that you may consider the Europeans living in the sticks with respect to microprocessors. And of course I must beg your pardon for my imperfect English!

I feel that the design of the bus is a vital aspect of the further development in the microcomputer field. As far as I can see there are at the moment two trends: first, all components of a microcomputer system are located on a single (large) pc board e.g. PET, APPLE, AIM 65, CHALLENGER II etc.; second, there is a well defined mother board bus with plug-onto boards each representing a well defined portion of a computer e.g. SWTPC, IMSAI, HORIZON etc.! Due to the increasing complexity and integration of the digital ICs the first attempt will lead in the near future to some microprocessor systems with 64k bytes of RAM/ROM, keyboard, video interface, cassette and floppy I/O, a 40-columns printer and a standard interface connector (IEEE-bus) all on one single pc board, which will not need any external expansion. So the problem of the bus design will be completely inadequate for this type of "personal computers."

The bus design will therefore offend only the design of larger systems which will probably be non-byte oriented! I think these "microcomputers" will represent the next generation of the current "minicomputers"—and they will reproduce their architecture, including the bus structure. Even today many micros compete very well with the last generation of PDPs, NOVAs and HP21s! These trends however call for a bus design—if at all a new bus should be established—which is based upon an address path which is broader than 16 bits and a data path which is at least 16 bits wide.

Everybody involved in microprocessors knows that the now formally standardized S-100 bus was the incidental definition of the first design with all the questionable definitions of such a design. The number of unnecessary signals on the bus is excessive increasing cross-talk and general noise. The arrangement of the different lines does in many cases not take into account the interaction of neighbouring lines: signals coming out of and going into the CPU are adjacent to each other, data and address lines are partly intermixed and especially in systems with extended memory spread over the whole bus etc. The use of direct contacts to the maximum of 100 fingers makes the design of cards and their fixation within the connectors difficult and some more. Any attempt to expand the bus to 16 bits of data and/or 24 bits of addresses makes the situation even worse. So I personally feel completely discouraged to use a system based upon this bus—despite the fact, that the bus is now firmly established as an IEEE standard, as I was informed.

Unfortunately the newly proposed standard seems not to be much better—as far as one could learn from the above-mentioned article. It will introduce however a new, un-standardized card format, for which you will probably get no accessories (pc boards etc.) at least in the next future. I think the proposed card format is better than the European standard "EUROCARD" which we here have to struggle with. As was pointed out by the author the usable area of the Eurocard is too limited and the spacing of the (indirect) 64-pin connectors is very close. This is a significant disadvantage for the home brew designer and experimenter(!) because the artwork upon the pc board design is nearly impossible without photographic size reduction techniques i.e. by expensive methods. Nevertheless the European industry is succeeding in designing up-systems using the normal 100 X 160 mm Eurocard (e.g. KONTRON, PEP, SIEMENS). Unfortunately there is no standard for the bus and as a consequence, there are at the moment at least four different busses, all based upon the standardized 64-pin connector and the above-mentioned card format. All cards will plug together-but will not work together!

Working in the computer field for more than 20 years I'm interested in a private system for semi-professional applications which must therefore be expandable beyond the 16+8 bit frontiers and which gives the possibility to design special cards without excessive costs i.e. have a large enough usable area and an indirect connector system. Though using a 12-bit system based on the Intersil 6100 chip (very nice and quite powerful and quite familiar to an old PDP and HP hacker) I decided to use the well established SS-50 bus instead of designing my own bus. And my intention is to point out that this already existing bus is a much better design than most people seem to be aware of. It uses sufficient large pc boards, an indirect connector and—most important—an orderly separation of data, addresses and control signals! The spacing between adjacent trays allows for broad copper lines on the mother board and the bus may be easily expanded to a maximum of 24 bits for data and/or data by multiplying addresses and data (as the Intersil chip does).

There are only a few objections against this bus—besides the fact that the connectors are not easily to obtain here in Germany. First of all there is no "wait for slow memory device" line defined. Second: the feeding of the different baud rate signals along the main bus (!) is unnecessary—only I/O devices will use them—and may cause noise. Third: some signals on the bus are used only for the connection of the processor pc board and the front cover. Eliminating these signals from the bus would make it possible to maintain the 50-pin design together with some expansion signals which may be necessary for the adaption of dynamic or bubble memories and the multiplexing of data and addresses along the bus. This however would allow an easy adaption of the new processor chips like the M68000 or Z8000!

I feel that a thorough and careful modification of the already established SS-50 bus would be a better solution than creating a completely new bus. Perhaps *Dr. Dobb's Journal* can pay some attention to this question. Personally I am a little bit astonished that the SS-50 bus didn't get the popularity which the S-100 bus gained.

For—I must repeat it—the SS-50 bus is the better design.

Yours truly,
K. Dieter Schaefer

Im Langenstrich 11
D 6333 Braunfels/Lahn
Germany

Thanks for your comments—and your English is much better than my German! We would like to hear much more from our European readers.

ERRATA

Dear *Dr. Dobb's*,

I was pleased to see my article in the Feb '79 issue of *DDJ* —thanks for the free issues.

However, the changes required to the article did not get made in the published version.

Perhaps you can publish an errata in a future issue.

Thanks,
R. J. Long

19 Cedarleigh Rd.
Kenmore, Qld. 4069
Australia

Here are the changes mentioned by Mr. Long:

On page 5 of the text, 5th line from the bottom,

use
"5009: C3 06 F0 C3 39 F8 Jump to memory read"

instead of
"5009: C3 06 F0 C3 F8 39 Jump to memory read".

On page 6, 10th line from the top,

use
"83AB: CD 06 40 CD 3F F8 Calls RESETW."

instead of
"83AB: CD 06 04 CD 3F F8 Calls RESETW.".

SIMPLE PATCH FOR SWTBUG

Dear Dr. Dobbs:

This is just a short letter to describe a simple patch to SWTPC 8K Basic V2 for use with SWTBUG and an ACIA on the control port. The SWTBUG documentation says that 8K V2 Basic can't be modified for the ACIA. Listed below are the original and new versions of the BREAK routine for 8K V2 Basic.

ORIGINAL BREAK ROUTINE

```
$03A3   LDAA $8004    86 8004  Check status bit of control port
$03A6   BMI→$038A     2B 12
$03A8   STX 008A      DF 8A
$03AA   LDX #$8004    CE 8004
$03AD   JSR INCH      BD E1AC  Get character from control port
$03B0   LDX $008A     DE 8A
$03B2   CMPA $0156    81 0156  Check if BREAK character
$03B5   BNE→$038A     26 03    No,goto $038A and resume BASIC
$03B7   JMP $08AB     7E 08AB  Yes,goto $08AB (BREAK sequence)
$038A   PULA          32
$038B   PULB          33
$038C   RTS           39
```

NEW BREAK ROUTINE

```
$03A3   LDAA #1       86 01    Check RDRF bit on ACIA
$03A5   BITA $8004    B5 8004
$03A8   BEQ→$038A     27 10    RDRF=False,goto $038A and resume BASIC
$03AA   LDAA $8005    86 8005  RDRF=True,get character from control port
$03AD   NOP           01       and then skip down to $0382 to
$03AE   NOP           01       check for BREAK character.
$03AF   NOP           01
$03B0   NOP           01
$03B1   NOP           01
```

```
86
01
B5
80
04
27
10
B6
80
05
01
01
01
01
01
PV
0010 FOR I=931 TO 945
0020 LET X=PEEK(I)
0030 GOSUB 100
0040 NEXT I
0045 LIST #7
0050 END
0100 REM DECIMAL TO HEX CONVERSION
0110 LET A$="0123456789ABCDEF"
0120 LET X1=INT(X/16)
0130 LET X2=X-(X1*16)
0140 PRINT #7,MID$(A$,X1+1,1);
0150 PRINT #7,MID$(A$,X2+1,1)
0160 RETURN
```

As far as a machine generated program listing goes, I am enclosing a BASIC generated listing of the memory areas involved in the patch as proof. To quote Dr. Dobbs, "If the computer typed it, the computer would probably accept it . . . " The listing was generated with the attached BASIC program and was printed on an old model 28 TTY driven by an ASCII-Baudot conversion which is linked to the BASIC interpreter on port #7.

I suspect that a similar fix will also work on 4K BASIC and SWTPC Microbasic as well as other programs for the SWTPC machine.

Sincerely yours,
Noel M. Moss

7760 Bellstone Rd.
St. Louis, MO 63119

A USEFUL MEMORY CLEAR ROUTINE

Dear Dr. Dobbs:

Here is a hint I have found useful, and it should apply to most 8080 or Z80 systems if the RST-7 location is available and a Monitor program of some sort is in memory. Clear unused RAM to all high instead of all low, that is, fill the RAM with RST-7 instructions, and at the RST-7 location (octal 70, hex 38) poke in a jump instruction to return to your Monitor. Even a "jump to self" loop would be ok. This way, if you have a programming or memory error which causes the computer to execute either non-programmed or non-existant memory, it will immediately be vectored back to your Monitor, hopefully before anything is wiped out.

The listing here is the memory clear routine the way I have it implemented in my Monitor, which is in EPROM located in high memory. The clearing starts at address 0 and continues through all contiguous RAM until it finds a location that won't zero. Notice that as it is written here, it requires that the stack be pre-loaded with the starting address of the Monitor, and that it should be JMPed to, not CALLed. Of course, the POP could be replaced with a LXI instruction instead. Also, any of the restarts could be used, but the advantage of the RST-7 is that it is good for non-existant memory as well as the non-programmed.

```
CLRMEM  LXI H Ø       ;starting address to clear
        MVI B 377Q    ;RST 7 instruction
        XRA A         ;clear acc.
CLRM    MOV M,B       ;RST 7 into memory
        INR L         ;next location
        JNZ CLRM      ;loop for full page
        INR H         ;next page
        MOV M,A       ;zero into memory
        CMP M         ;did it zero?
        JZ CLRM       ;then clear another page
```

```
POP H          ;else we're done, get Monitor address
SHLD 71Q       ;poke it into RST 7 location,
MVI A 303Q     ;
STA 70Q        ;along with a JMP instruction
PCHL           ;return to Monitor.
```

Sincerely,
Larry Hudson

2232 Alameda Ave.
Ventura, CA 93003

IMPROVE YOUR OSI RESIDENT EDITOR

Dear Dr. Dobbs,

I've been reading your journal since it came out and I thought that it was about time I got a subscription. So, here is my fifteen dollars. (Well, inflation does hit *all* of us!)

Here is a simple program to fix up a problem with OSI's resident editor (supplied with 65D). The editor has a nice feature which allows you to append a line to a previously entered line. You can do this by first printing the line you wish to append with the 'P' command and then typing the line to be appended with a line number of zero. The only thing bad about this is that it retains the carriage return and line feed from the original line. BASIC and the Assembler both give you a variety of errors when they encounter them. To remedy this, after you have typed in the lines to be appended in the normal way (with 0 as a line number), load this program through DOS with the command:

```
A*C2130=tt/s
```

where tt is the track number that it was stored on, and s is the sector (usually 1). Then run it and return to the assembler with:

A*RE	get into the extended monitor
:G2130	run the program
:D	back to DOS
A*RA	return to editor/assembler
INTZ?N	don't initialize memory
.	you're done, continue with work

The program resides from $2130 to $2230 and uses $BO and $B1 of page zero. None of this will affect the operating system. The program should be assembled with an .A3 assembly with *no* offset (M0000 in DOS sets the offset to zero). Then store it in core image on disk with:

```
A*Stt,s=2130/1
```

where tt is the track number (up to you) and s is the sector (usually 1). Now the program will be ready to use with the above procedure.

This program inserts three blanks between the original and the appended line, so keep this in mind when you are typing the appended line.

I hope that this program proves to be useful.

Yours truly,
Mark Lentczner

445 Third Street
Brooklyn, NY 11215

```
INTZ?NO
•ASSEMBLE

 10 0000          ;COMMENT ADDER
 20 0000          ;    FOR USE WITH OSI EDITOR/ASSEMBLER
 30 0000          ;BY MARK LENTCZNER
 40 0000          ;    WRITTEN: NOVEMBER 27, 1978
 50 0000          ;
 60 2130                *=$2130
 70 2130                WSB=$12C9    WORK SPACE BEGINING POINTER
 80 2130                WSE=$12FE    WORK SPACE END POINTER
 90 2130                EXTD=$1702   EXTENDED MONITOR ENTRY POINT
100 2130                COM=$B0      COMMENT TEST BYTE POINTER
110 2130          ;GET ADDRESS OF FIRST BYTE
120 2130 ADC912   START   LDA WSB
130 2133 85B0            STA COM
140 2135 ADCA12          LDA WSB+1
150 2138 85B1            STA COM+1
160 213A          ;LOOK FOR APPEND SEQUENCE: 0D 00 00
170 213A A900     TEST    LDA #$00    OFFSET=0
180 213C B1B0             LDA (COM),Y    FIRST BYTE
190 213E C90D             CMP #$0D
200 2140 D015             BNE NEXT
210 2142 C8               INY            OFFSET=1
220 2143 B1B0             LDA (COM),Y    SECOND BYTE
230 2145 C900             CMP #$00
240 2147 D00E             BNE NEXT
250 2149 C8               INY            OFFSET=2
260 214A B1B0             LDA (COM),Y    THIRD BYTE
270 214C C900             CMP #$00
280 214E D007             BNE NEXT
290 2150          ;SEQUENCE IS FOUND!
300 2150          ;LOAD FILL CHR• IF YOU DON'T WANT IT TO INSERT
310 2150          ;SPACES, CHANGE LINE 320 TO: LDA #$00
320 2150 A920             LDA #$20
330 2152 91B0     BLANK   STA (COM),Y
340 2154 88               DEY
350 2155 10FB             BPL BLANK    LOOP FOR ALL THREE BYTES
360 2157          ;NEXT BYTE TO TEST
370 2157 E6B0     NEXT    INC COM
380 2159 D0DF             BNE TEST     BRANCH IF NO PAGE CHANGE
390 215B A5B1             LDA COM+1
400 215D E6B1             INC COM+1    INC HIGH ADDRESS
410 215F C0FF12           CMP WSE+1    EQUAL TO END PAGE?
420 2162 D0D6             BNE TEST     IF NOT TEST NEXT BYTE
430 2164 4C0217           JMP EXTD     FINISHED, RETURN TO MONITER
440 2167                  •END

•A3       ASSEMBLE INTO MEMORY

•END 0    RETURN TO DOS
A*S18,1=2130/1
A*RA


INTZ?Y
•10         THIS IS A LINE
•20         THIS IS A LINE TOO
•0, IT HAS A COMMENT
•PRINT
   10         THIS IS A LINE
   20         THIS IS A LINE TOO
     , IT HAS A COMMENT

•END0     RETURN TO DOS
A*C2130=18,1
A*RE
:G2130
:D
A*RA


INTZ?NO     DON'T CLEAR MEMORY
•PRINT

   10         THIS IS A LINE
   20         THIS IS A LINE TO    , IT HAS A COMMENT

•END
A*U09
A*U01
```

A COUPLE OF THINGS . . .

Dear Doc:

I was very interested in the Kim editor you just published. (*DDJ* No. 30) I have been doing a similar project on an OSI machine. The problem in designing such a program is covered very well by the program you published. There are only two things I'd suggest. One, make the line numbers automatic. If you would like to change your position in the file, have another command. Two, with a serial terminal all the nice features of video or typing in the local mode are lost. These features would include: delete, insert, tab, and others and

would add less than 400 bytes to the program.

Enclosed is a partial renumbering program for Microsoft's Basic running on the OSI machine. If a Rom version is used, locations 574 and 575 should be changed to locations 11 and 12.

The more I learn about all of this, the more I enjoy your magazine.

Yours truly,
L. Barker — Chicago, Ill. 60660

```
A
 10 0000          ;        Re number and pretty print
 20 0000          ;        for OSI's 65-D  (6502 CPU)
 30 0000          ;        Written by  L. Barker
 40 0000                NXTLOW=80
 50 0000                NXTHI=81
 60 6000                  *=24576
 70 6000          ;
 80 6000 A97F             LDA #127
 90 6002 8550             STA NXTLOW
100 6004 A931             LDA #49
110 6006 8551             STA NXTHI
120 6008 A900             LDA #0    ; start numbering with zero
130 600A 8552             STA 82    ; line count low
140 600C A900             LDA #0
150 600E 8553             STA 83    ; line count high
160 6010 A002       LOOP  LDY #2
170 6012 A552             LDA 82
180 6014 9150             STA (80),Y
190 6016 C8               INY
200 6017 A553             LDA 83
210 6019 9150             STA (80),Y
220 601B C8               INY
230 601C B150             LDA (80),Y
240 601E C921             CMP #33
250 6020 D004             BNE TENMOR
260 6022 A920             LDA #32   ; pretty print
270 6024 9150             STA (80),Y
280 6026 18         TENMOR CLC
290 6027 A90A             LDA #10
300 6029 6552             ADC 82
310 602B 8552             STA 82
320 602D A900             LDA #0
330 602F A8               TAY
340 6030 6553             ADC 83
350 6032 8553             STA 83
360 6034 B150             LDA (80),Y
370 6036 48               PHA
380 6037 C8               INY
390 6038 B150             LDA (80),Y
400 603A 8551             STA NXTHI
410 603C 68               PLA
420 603D 8550             STA NXTLOW
430 603F D0CF             BNE LOOP
440 6041 A551             LDA NXTHI
450 6043 D0CB             BNE LOOP
460 6045 A920             LDA #32
470 6047 8D3E02           STA 574
480 604A A92E             LDA #46
490 604C 8D3F02           STA 575
500 604F 60               RTS
510 6050            .END
```

```
LIST
 0 POKE 574,0 :POKE 575,96 :X=USR(X) :END
 2 PRINT "THIS IS A TEST
 3 !
 4 FOR I=1 TO 3
 5 ! PRINT "TEST #";I
 6 NEXT
 7 !
 9 REM  will not renumber goto's
 11 GOTO 4
 12 GOSUB 14
 13 END
 14 RETURN
OK
RUN
OK
LIST
 0 POKE 574,0 :POKE 575,96 :X=USR(X) :END
 10 PRINT "THIS IS A TEST
 20
 30 FOR I=1 TO 3
 40   PRINT "TEST #";I
 50 NEXT
 60
 70 REM   will not renumber goto's
 80 GOTO 4
 90 GOSUB 14
 100 END
 110 RETURN
OK
0
RUN
THIS IS A TEST
TEST # 1
TEST # 2
TEST # 3
?US ERROR IN  80
OK
```

SEARCH in SMAL/80

Ma'am:

I wonder if you and your readers would be interested in seeing what Mike Gabrielson's SEARCH routine (*DDJ*, No. 30, p. 16) looks like in SMAL/80, a compiled, structured 8080 language that was developed and is being sold by Chromod Associates (I am one of the Associates).

The SMAL/80 version looks longer but it will compile into exactly the same object code (almost) as Mr. Gabrielson's Intel version. The big difference to a reader is in the understandability of the program, which is due to the symbolic notation used as well as the structured constructs and indentation. The reader can see immediately that the routine consists of one long loop, with an inner loop that compares the characters in the caller's string with the characters in the node's string, plus three IF-THEN-ELSE's that test for different conditions within the loops as the routine is traversed.

Yours,
Morris Krieger — 37 Eighth Avenue, Brooklyn, N.Y. 11217

```
/* at entry, BC holds addr of string to look for, terminated
by NUL.  DE holds address of ROOT pointer.  HL holds contents
of ROOT pointer.  At exit, if CY = 1, string has not been
found; if CY = 0, string has been found. */

NUL EQU 0                       /* end-of-string char */;


SEARCH:                         /* SEARCH routine */
   LOOP;
      A = H                     /* does pointer indicate end of tree? */;
      A = A OR L;
      IF CY = 1 ZERO            /* assume yes and if TRUE, end of tree */
         RETURN                 /* and exit SEARCH */;
      PUSH BC                   /* else (if FALSE), save addr of
                                   caller's string */:
      PUSH HL                   /* save addr of node */;
      ++HL                      /* skip past node's l. and r. subtree
                                   pointers and stop at first char of
                                   node's string */;
      ++HL;
      ++HL;
      ++HL;
      LOOP;
         A = M(BC)              /* get char from caller's string */;
         IF A : M(HL) ZERO      /* if same as char in node's string */
            BREAK               /* leave loop */;
         ++BC                   /* else, point to next char pair */;
         ++HL;
      REPEAT WHILE A : NUL NOT ZERO     /* while not at end of char string */;
      HL = POP                  /* else, restore node addr */;
      BC = POP                  /* and addr of caller's string */;
      IF ZERO                   /* if strings match */;
         RETURN                 /* exit SEARCH */;
      ELSE IF NOT CY THEN       /* if caller's string is high */
         [
         ++HL                   /*get addr of r. substring pointer */;
         ++BC;
         ]
      ELSE                      /* if caller's string is low */
         [
         D = M(HL)              /* get addr of l. subtree in DE */;
         ++HL;
         E = M(HL);
         HL <=> DE              /* and then into HL */;
         --DE                   /* let DE point to last node accessed */;
         ]
   REPEAT                       /* keep searching down the tree */;
```

ON FORTH

Dear Editor:

One point seems to have been missed in your previously published discussions about FORTH and the inconvenience of writing in reverse polish notation instead of algebraic notation. Rather than justify post-fix format as having no disadvantages, I would point out that, as a language very close to machine language, using FORTH does not preclude writing FORTH functions to parse expressions entered in algebraic format instead of true FORTH code. Processing such algebraic input would be at the application level, and limited to those problems involving extensive encoding of formulas obtained from printed documents, as with a mathematical simulation model or formulas for financial reports. With a custom coded translator, such input can be as fancy as the material requires instead of being restricted by the constraints on designing a general purpose compiler. Since it would only need to generate FORTH source code as the output to be processed further by the FORTH compiler, it could be quite simple, and the stack oriented environment simplifies it further.

Such a facility is not included in the basic FORTH system and would have to be supplied by the user. Perhaps an important reason why examples of it have not already been presented is the expectation that the applications where it is really needed normally require a large-scale computer. It is thus irrelevant given the equipment for which FORTH is intended. The do-it-yourself computer hobbyist, however, brings more ambitious demands for performance from mini-computer systems than would be relevant to commercial users.

Yours truly,
George B. Lyons

280 Henderson Street
Jersey City, NJ 07302

TEN 8085 INSTRUCTIONS

Dear *Dr. Dobb's*,

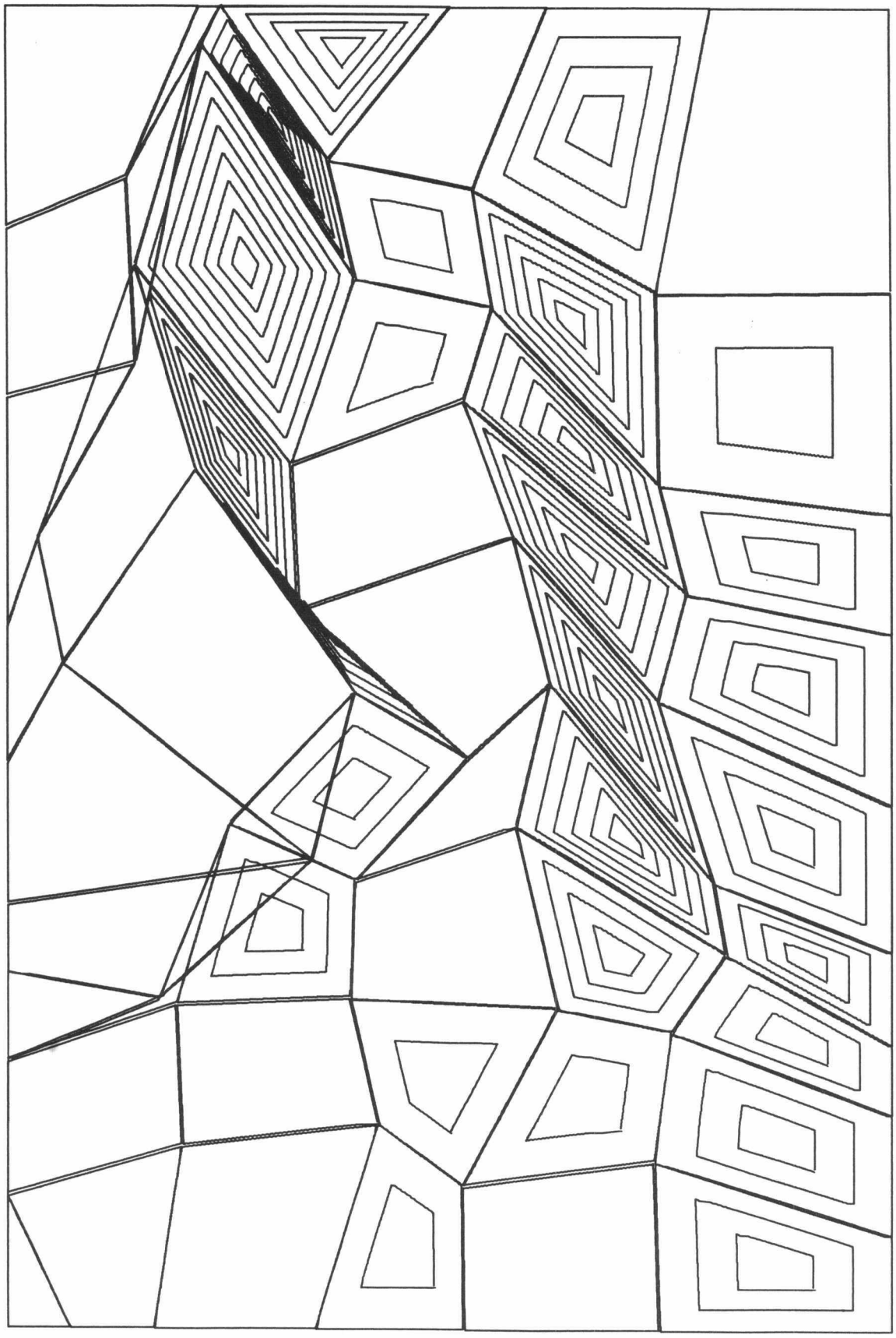
Your readers might be interested in an article by W. Dehnhardt and V. Sorensen on page 144 of the January 18th issue of *Electronics*, which describes ten previously undocumented instructions for the 8085:

DSUB - subtract BC from HL
ARHL - arithmetic shift of HL right
RDEL - rotate DE left through carry
LDHI - load DE with HL + immediate byte
LDSI - load DE with SP + immediate byte
RSTV - restart on overflow flag
SHLX - store HL at location addressed by DE
LHLX - load HL from location addressed by DE
JNX - jump on not X flag
JX - jump on X flag

Sincerely,
Mike Gabrielson

Box 2692
Stanford, CA 94305

Coincidentally, we thought so too – Electronics *had given us permission to reprint the article when we received your letter. Look for it in this issue.* – *SR*



© David Dameron 1979

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 35 | May 1979 | Volume 4, Issue 5

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

A Z80-ZAP Disassembler 16
B.W. Lee

The Software Tools FORMAT Program: An 8080 Version 4
Mike Gabrielson

Patching the SWTPC Co-Resident Editor-Assembler 28
Geoffrey A. Gass

ODT User's Manual 40
Gary D. Gaugler

Tiny-c Interpreter on CDOS 37
Ray Duncan

Programming Pastimes and Pleasures 26
Charles Wetherell

And Much More ...

The Software Tools FORMAT Program: An 8080 Version

BY MIKE GABRIELSON
Box 2692
Stanford, Calif. 94305

FORMAT is a program for neatly formatting and printing arbitrary text. It can output documents to terminals or line printers, and provide automatic justification, page numbering, margin control on all four page borders, centering, underlining, indenting, interline spacing, and titles at page tops and bottoms.

FORMAT was originally published as a Ratfor (Rational Fortran) program in the excellent book *Software Tools*, by Brian W. Kernighan and P.J. Plauger. The version presented here is virtually the same program, but it has been translated by hand into 8080 assembly language. Because the Ratfor version is so thoroughly documented in the book, and since all variable and subroutine names have been retained to allow easy comparison with the original Ratfor code, the comments in the listing of the 8080 version are a bit sparse.

How To Use FORMAT

FORMAT inputs text as a continuous stream of ASCII characters, with carriage returns (the NEWLINE character) separating lines. As FORMAT copies the input to its ASCII output device, it will adjust the text to cause default pagination (skipping over the fold in continuous paper) and justification. FORMAT will do a reasonable job of printing a document without any intervention or special preparation on the part of the user. Sophisticated formatting can be performed by interspersing the text to be formatted with special commands that tell FORMAT how it should adjust the text. A command consists of a period (which must be the first character on a line) followed only by a command keyword and occasionally an optional number or string argument. For example,

.center

tells FORMAT to center the next line of output, and

.MARGIN 50

sets the right margin to 50 for all subsequent output. Commands can be any mixture of upper and lower case characters, and may be arbitrarily abbreviated. For example,

.center
.CEN
.c

will all have the same effect. Command names end (and any following arguments begin) with the first non-alphabetic character following the period at the beginning of the command. Illegal commands or arguments are ignored.

Any command allowing a numeric argument will also allow the same argument to be preceded by a + or - to indicate a relative change from the current value. For example,

.margin -10

will pull in the right margin 10 places from its current value.

Certain commands implicitly cause what is called a "break," i.e., cause a new line to be started.

FORMAT Command Summary

The complete set of FORMAT commands given here should be compared with the summary given on page 223 of *Software Tools*.

COMMAND	BREAK?	DEFAULT	FUNCTION
.break	yes		cause a break
.center n	yes	n = 1	center next n lines
.footer title	no	empty	put title on bottom of following pages
.hang n	yes	n = 0	temporary indent of n spaces
.header title	no	empty	put title on top of following pages
.indent n	no	n = 0	indent n spaces
.jagged	yes		stop justification
.justify	yes		start justification
.length n	no	n = 66	set page length to n lines
.margin n	no	n = 60	set right margin to n
.page n	yes	n = +1	begin page number n
.skip n	yes	n = 1	skip down n lines
.space n	no	n = 1	line spacing is n
.under n	no	n = 1	underline words in next n lines

When a command is found, FORMAT searches the above list. If an abbreviated command can match more than one command, the first matching command in the list is used. When no argument is specified in a command, FORMAT will use the default value.

Differences From The Ratfor Version

The COMAND routine in the Ratfor version relied on a long chain of ELSE-IF constructs to perform command decoding. The 8080 COMAND instead calls a new SEARCH routine that searches a table of valid commands and allows abbreviated commands as previously described.

The UNDERL routine has been modified because of an apparent bug in the Ratfor version: on page 243 of *Software Tools*, "j < size-1" does not appear to be a sufficient test. For example, if (buf(i) = 'A') and (j=size-2) at the start of the for-loop, then the original code will perform as follows . . .

```
(1)  tbuf(j) := 'A'
(2)  j := size-1
(3)  tbuf(size-1) := BACKSPACE
(4)  tbuf(size) := UNDERLINE
(5)  j := size+1
(6)  for-loop exits
(7)  SUBSCRIPT ERROR:  tbuf(size+1) := NEWLINE
(8)  SUBSCRIPT ERROR:  tbuf(size+2) := EOS
```

Note also that the symbol "max" in the LEADBL routine on page 236 happens to be unreferenced.

To allow output on non-continuous paper, a variable named WAITUP has been added. The PHEAD routine has been modified so that if WAITUP contains the value YES, then FORMAT will wait for the operator to type any character at the console before beginning to print each page. This gives the operator a chance to insert a new sheet of paper in the printer. Currently, INIT always sets WAITUP to NO.

How To Make FORMAT Work On Your Computer

FORMAT requires four support routines configured for the local operating environment:

ROUTINE	FUNCTION
SGETC	Returns with carry set if the end of the input file has been reached, else returns the next ASCII input character in register A. Other registers are not modified.
SPUTC	Outputs the ASCII character in register A and leaves all registers unmodified.
SQUIT	Called when FORMAT wishes to terminate and never regain control.
SWAIT	Waits for any operator input at the console to allow printer paper to be changed. Leaves registers unmodified. Only called if the variable WAITUP contains the value YES.

The cross reference listing can be used to find where the support routines are called from. FORMAT has no other special requirements.

Final Notes

Since a one pass assembler was used to generate the listing, all forward references appear as question marks in the object code field of each source line. The symbol table listing should be used to determine the ultimate value of all labels. A hex/ASCII dump of the entire program is also provided.

```
Line Addr Contents

   1 0000                ;------
   2 0000                ; format ... the Software Tools text formatter
   3 0000                ;
   4 0000                ; Mike Gabrielson   12/2/78
   5 0000                ;
   6 0000                ; This program is a hand compiled and somewhat modified version
   7 0000                ; of the format program in the book "Software Tools".  Don't let
   8 0000                ; the absence of comments scare you!  The book contains complete
   9 0000                ; documentation for this code.
  10 0000                ;------
  11 0000
  12 0000                        NOEJECT
  13 0000
  14 0000                ; External routines not defined in this assembly:
  15 0000
  16 0000                        !EXTRN $WAIT            ;wait for console input
  17 0000                        !EXTRN $QUIT            ;end execution
  18 0000                        !EXTRN $GETC            ;get next character into A
  19 0000                        !EXTRN $PUTC            ;output character in A
  20 0000
  21 0000                BACKSPACE = 8
  22 0000                COMMAND = '.'
  23 0000                EOS = 0
  24 0000                HUGE = 1000
  25 0000                LINSIZ = 80 + 2         ; +2 = ROOM FOR NEWLINE AND EOS
  26 0000                NEWLINE = #D
  27 0000                NO = 0
  28 0000                PAGELEN = 66
  29 0000                PAGENUM = '#'
  30 0000                PAGEWIDTH = 60
  31 0000                TAB = 9
  32 0000                TRIPLE = LINSIZ*3       ;UNDERLINING TRIPLES THE LENGTH
  33 0000                YES = -1
  34 0000
  35 0000 CD ?? ??      FORMAT: CALL    INIT
  36 0003
  37 0003                MAIN_LOOP:
  38 0003 21 ?? ??              LXI     H,INBUF
  39 0006 CD ?? ??              CALL    GETLIN
  40 0009 DA ?? ??              JC      FLUSH
  41 000C 3E 2E                 MVI     A,COMMAND
  42 000E BE                    CMP     M
  43 000F C2 ?? ??              JNZ     ITS_TEXT
  44 0012 CD ?? ??              CALL    COMAND
  45 0015 C3 03 00              JMP     MAIN_LOOP
  46 0018
  47 0018                ITS_TEXT:
  48 0018 CD ?? ??              CALL    TEXT
  49 001B C3 03 00              JMP     MAIN_LOOP
  50 001E
  51 001E 2A ?? ??      FLUSH:  LHLD    LINENO
  52 0021 7C                    MOV     A,H
  53 0022 B5                    ORA     L
  54 0023 CC -- --              CZ      $QUIT
  55 0026 21 E8 03              LXI     H,HUGE
  56 0029 CD ?? ??              CALL    SPACE
  57 002C CD -- --              CALL    $QUIT
  58 002F
  59 0081               FOOTER: DS      LINSIZ
  60 00D3               HEADER: DS      LINSIZ
  61 01C9               INBUF:  DS      TRIPLE
  62 02BF               OUTBUF: DS      TRIPLE
  63 03B5               WRDBUF: DS      TRIPLE
  64 03B5
  65 03B6               BACKUP: DS      1
  66 03B8               BOTTOM: DS      2
  67 03BA               CEVAL:  DS      2
  68 03BC               CURPAG: DS      2
  69 03BC 00            DIR:    0
  70 03BE               FILL:   DS      1
  71 03C0               INVAL:  DS      2
  72 03C2               LAST:   DS      2
  73 03C4               LINENO: DS      2
  74 03C6               LLVAL:  DS      2
  75 03C8               LSVAL:  DS      2
  76 03CA               M1VAL:  DS      2
  77 03CC               M2VAL:  DS      2
  78 03CE               M3VAL:  DS      2
  79 03D0               M4VAL:  DS      2
  80 03D2               NB:     DS      2
  81 03D4               NE:     DS      2
  82 03D6               NEWPAG: DS      2
  83 03D8               NEXTRA: DS      2
  84 03DA               NHOLES: DS      2
  85 03DC               OUTP:   DS      2
  86 03DE               OUTW:   DS      2
  87 03E0               OUTWDS: DS      2
  88 03E2               PLVAL:  DS      2
  89 03E4               RMVAL:  DS      2
  90 03E6               SPVAL:  DS      2
  91 03E8               TIVAL:  DS      2
  92 03EA               ULVAL:  DS      2
  93 03EB               WAITUP: DS      1       ;YES IF FORMAT SHOULD WAIT AT START OF EACH
  94 03EB                                                                       PAGE
  95 03EB               COMMANDS:
  96 03EB
  97 03EB 42 52 45              'BREAK', EOS, @ BREAK!
  98 03F3 43 45 4E              'CENTER', EOS, @ CENTER!
  99 03FC 46 4F 4F              'FOOTER', EOS, @ FOOTER!
 100 0405 48 41 4E              'HANG', EOS, @ HANG!
 101 040C 48 45 41              'HEADER', EOS, @ HEADER!
 102 0415 49 4E 44              'INDENT', EOS, @ INDENT!
 103 041E 4A 41 47              'JAGGED', EOS, @ JAGGED!
 104 0427 4A 55 53              'JUSTIFY', EOS, @ JUSTIFY!
 105 0431 4C 45 4E              'LENGTH', EOS, @ LENGTH!
 106 043A 4D 41 52              'MARGIN', EOS, @ MARGIN!
 107 0443 50 41 47              'PAGE', EOS, @ PAGE!
 108 044A 53 4B 49              'SKIP', EOS, @ SKIP!
 109 0451 53 50 41              'SPACE', EOS, @ SPACE!
 110 0459 55 4E 44              'UNDER', EOS, @ UNDER!
 111 0461
 112 0461               ENDCOM = *
 113 0461
 114 0461               ;------
 115 0461               ; ALPHA - TEST CHARACTER FOR LETTER
 116 0461               ; ACCEPTS:  A = CHARACTER
 117 0461               ; RETURNS:  CARRY SET IF A = A-Z OR a-z
 118 0461               ;------
 119 0461 FE 61         ALPHA:  CPI     'a'
 120 0463 DA ?? ??              JC      NOTLC
 121 0466 FE 7B                 CPI     'z'+1
 122 0468 D8                    RC
 123 0469
 124 0469 FE 5B         NOTLC:  CPI     'Z'+1
 125 046B D0                    RNC
 126 046C FE 41                 CPI     'A'
 127 046E 3F                    CMC
 128 046F C9                    RET
 129 0470
 130 0470
 131 0470               ; BRK - END CURRENT FILLED LINE
 132 0470               ;------
 133 0470 F5            BRK:    PUSH    PSW
 134 0471 C5                    PUSH    B
 135 0472 E5                    PUSH    H
 136 0473 11 00 00              LXI     D,0
 137 0476 2A DA 03              LHLD    OUTP
 138 0479 CD ?? ??              CALL    COMPARE
 139 047C D2 ?? ??              JNC     EMPTY
 140 047F 11 C9 01              LXI     D,OUTBUF
 141 0482 19                    DAD     D
 142 0483 36 00                 MVI     M,EOS
 143 0485 2B                    DCX     H
 144 0486 36 0D                 MVI     M,NEWLINE
 145 0488 EB                    XCHG
 146 0489 CD ?? ??              CALL    PUT
 147 048C 21 00 00      EMPTY:  LXI     H,0
 148 048F 22 DA 03              SHLD    OUTP
 149 0492 22 DC 03              SHLD    OUTW
 150 0495 22 DE 03              SHLD    OUTWDS
 151 0498 E1                    POP     H
 152 0499 C1                    POP     B
 153 049A F1                    POP     PSW
 154 049B C9                    RET
 155 049C
 156 049C               ;------
 157 049C               ; CENTER - CENTER A LINE BY SETTING TIVAL
 158 049C               ; ACCEPTS:  HL = @ LINE
 159 049C               ;------
 160 049C E5            CENTER: PUSH    H
 161 049D CD ?? ??              CALL    WIDTH
 162 04A0 2A E6 03              LHLD    TIVAL
 163 04A3 EB                    XCHG
 164 04A4 CD ?? ??              CALL    DELESSHL
 165 04A7 EB                    XCHG
 166 04A8 2A E2 03              LHLD    RMVAL
 167 04AB 19                    DAD     D
```

```
                        CALL    DIVBY2
                        LXI     D,0
                        CALL    MAX
                        SHLD    TIVAL
                        POP     H
                        RET

                ;------
                ; CHRBAK - PUSH CHARACTER BACK ONTO INPUT
                ; ACCEPTS:  B = CHARACTER
                ;------
CHRBAK: MOV     A,B
                STA     BACKUP
                RET

                ;------
                ; COMAND - PERFORM FORMATTING COMMAND
                ;------
COMAND: LXI     B,ENDCOM
                LXI     D,COMMANDS
                INX     H               ;GET @ KEYWORD (SKIP .)
                CALL    SEARCH          ;COMMAND IN TABLE?
                RC                      ;NO, IGNORE
                PUSH    D               ;STACK DISPATCH ADDRESS
                CALL    GETVAL
                RET                     ;JUMP TO COMMAND HANDLER
                                        ;DE = @ END OF COMMAND,
                                        ;HL = ARGUMENT VALUE
BREAK!: CALL    BRK
                RET

CENTER!:
                CALL    BRK
                LXI     B,0
                PUSH    B
                INX     B
                LXI     D,HUGE
                PUSH    D
                XCHG
                LHLD    CEVAL
                CALL    SET
                SHLD    CEVAL
                RET

FOOTER!:
                LXI     H,FOOTER
                CALL    GETTL
                RET

HANG!:  LXI     B,0
                PUSH    B
                XCHG
                LHLD    RMVAL
                PUSH    H
                LHLD    TIVAL
                CALL    SET
                SHLD    TIVAL
                RET

HEADER!:
                LXI     H,HEADER
                CALL    GETTL
                RET

INDENT!:
                LXI     B,0
                PUSH    B
                XCHG
                LHLD    RMVAL
                DCX     H
                PUSH    H
                LHLD    INVAL
                CALL    SET
                SHLD    INVAL
                SHLD    TIVAL
                RET

JAGGED!:
                CALL    BRK
                MVI     A,NO
                STA     FILL
                RET

JUSTIFY!:
                CALL    BRK
                MVI     A,YES
                STA     FILL
                RET

LENGTH!:
                PUSH    H
                LHLD    M1VAL
                XCHG
                LHLD    M2VAL
                DAD     D
                XCHG
                LHLD    M3VAL
                DAD     D
                XCHG
                LHLD    M4VAL
                DAD     D
                INX     H
                XTHL
                LXI     B,HUGE
                PUSH    B
                LXI     B,PAGELEN
                XCHG
                LHLD    PLVAL
                CALL    SET
                SHLD    PLVAL
                XCHG
                LHLD    M3VAL
                CALL    DELESSHL
                XCHG
                LHLD    M4VAL
                CALL    DELESSHL
                SHLD    BOTTOM
                RET

MARGIN!:
                XCHG
                LHLD    TIVAL
                INX     H
                PUSH    H
                LXI     H,HUGE
                PUSH    H
                LXI     B,PAGEWIDTH
                LHLD    RMVAL
                CALL    SET
                SHLD    RMVAL
                RET

PAGE!:  PUSH    H
                PUSH    PSW
                LXI     D,0
                LHLD    LINENO
                CALL    COMPARE
                LXI     H,HUGE
                CC      SPACE
                POP     PSW
                POP     D
                LXI     B,-HUGE
                PUSH    B
                LXI     B,HUGE
                PUSH    B
                LHLD    CURPAG
                MOV     B,H
                MOV     C,L
                INX     B
                CALL    SET
                SHLD    CURPAG
                SHLD    NEWPAG
                RET

SKIP!:  LXI     B,0             ;STACK MINIMUM
                PUSH    B
                INX     B               ;FORM DEFAULT
                LXI     D,HUGE          ;STACK MAXIMUM
                PUSH    D
                XCHG
                LHLD    SPVAL
                CALL    SET
                SHLD    SPVAL
                CALL    SPACE
                RET

SPACE!: LXI     B,1             ;GET MINIMUM AND DEFAULT
                PUSH    B               ;STACK MINIMUM
```

```
05C9 11 E8 03           LXI     D,HUGE          ;STACK MAXIMUM
05CC D5                 PUSH    D
05CD EB                 XCHG                    ;DE := ARGUMENT VALUE
05CE 2A C6 03           LHLD    LSVAL
05D1 CD ?? ??           CALL    SET
05D4 22 C6 03           SHLD    LSVAL
05D7 C9                 RET
05D8
05D8 01 01 00   UNDER!: LXI     B,1
05DB C5                 PUSH    B
05DC 0B                 DCX     B
05DD 11 E8 03           LXI     D,HUGE
05E0 D5                 PUSH    D
05E1 EB                 XCHG
05E2 2A E8 03           LHLD    ULVAL
05E5 CD ?? ??           CALL    SET
05E8 22 E8 03           SHLD    ULVAL
05EB C9                 RET
05EC
05EC                    ;------
05EC                    ; COMPARE - COMPARE TWO SIGNED NUMBERS
05EC                    ; ACCEPTS:  NUMBERS IN DE, HL
05EC                    ; RETURNS:  Z SET IF DE = HL
05EC                    ;           C SET IF DE < HL
05EC                    ;           C CLEAR IF DE >= HL
05EC                    ; DO THE USUAL SUBTRACT, BUT COMPLEMENT THE CARRY
05EC                    ; IF THE ORIGINAL SIGNS ARE NOT EQUAL
05EC                    ;------
05EC                    COMPARE:
05EC 7B                         MOV     A,E
05ED 95                         SUB     L
05EE 7A                         MOV     A,D
05EF 9C                         SBB     H
05F0 F5                         PUSH    PSW
05F1 7A                         MOV     A,D
05F2 AC                         XRA     H
05F3 E6 80                      ANI     #80
05F5 C2 ?? ??                   JNZ     SIGNEQ
05F8 F1                         POP     PSW
05F9 C9                         RET
05FA F1                 SIGNEQ: POP     PSW
05FB 3F                         CMC
05FC C9                         RET
05FD
05FD                    ;------
05FD                    ; DELESSHL - SUBTRACT HL FROM DE
05FD                    ; RETURNS:  HL = DE - HL
05FD                    ;------
05FD                    DELESSHL:
05FD 7B                         MOV     A,E
05FE 95                         SUB     L
05FF 6F                         MOV     L,A
0600 7A                         MOV     A,D
0601 9C                         SBB     H
0602 67                         MOV     H,A
0603 C9                         RET
0604
0604                    ;------
0604                    ; DIVBY2 - DIVIDE SIGNED NUMBER IN HL BY TWO
0604                    ; (SHIFT RIGHT BUT PROPAGATE THE SIGN)
0604                    ;------
0604 7C                 DIVBY2: MOV     A,H
0605 17                         RAL
0606 7C                         MOV     A,H
0607 1F                         RAR
0608 67                         MOV     H,A
0609 7D                         MOV     A,L
060A 1F                         RAR
060B 6F                         MOV     L,A
060C C9                         RET
060D
060D                    ;------
060D                    ; DIVIDE - DIVIDE DE BY HL
060D                    ; ACCEPTS:  HL = DIVISOR
060D                    ;           DE = DIVIDEND
060D                    ; RETURNS:  HL = QUOTIENT
060D                    ;------
060D 44                 DIVIDE: MOV     B,H             ;MOVE DIVISOR TO BC
060E 4D                         MOV     C,L
060F 21 00 00                   LXI     H,0             ;FORM INITIAL QUOTIENT
0612 7B                 DVLOOP: MOV     A,E             ;DIVIDE BY SUBTRACTION
0613 91                         SUB     C               ;(DE := DE - BC)
0614 5F                         MOV     E,A
0615 7A                         MOV     A,D
0616 98                         SBB     B
0617 57                         MOV     D,A
0618 D8                         RC
0619 23                         INX     H
061A C3 12 06                   JMP     DVLOOP
061D
061D                    ;------
061D                    ; GETCHR - GET NEXT CHARACTER
061D                    ; RETURNS:  CARRY SET IF EOF
061D                    ;           ELSE A = CHARACTER
061D                    ;------
061D 3A B5 03   GETCHR: LDA     BACKUP          ;ANY CHARACTER BACKED UP?
0620 FE 0D              CPI     NEWLINE         ;(NEWLINE CAN'T BE BACKED UP,
0622                                            ; IT MEANS BACKUP IS EMPTY)
0622 CA ?? ??           JZ      GETC            ;NO, READ NEXT CHARACTER
0625 47                 MOV     B,A             ;YES, SAVE CHARACTER IN B
0626 3E 0D              MVI     A,NEWLINE       ;GET NEWLINE
0628 32 B5 03           STA     BACKUP          ;EMPTY BACKUP
062B 78                 MOV     A,B             ;RESTORE CHARACTER
062C B7                 ORA     A               ;CLEAR CARRY
062D C9                 RET
062E
062E CD ~~ ~~   GETC:   CALL    $GETC           ;GET NEXT CHARACTER
0631 C9                 RET
0632
0632                    ;------
0632                    ; GETLIN - FILL BUFFER WITH NEXT INPUT LINE
0632                    ; ACCEPTS:  HL (SAVED) = @ BUFFER
0632                    ; RETURNS:  CARRY SET IF EOF
0632                    ;           ELSE LINE TERMINATED BY EOL, EOS
0632                    ;------
0632 54                 GETLIN: MOV     D,H             ;SAVE @ BUFFER IN DE
0633 5D                         MOV     E,L
0634 0E 00              GETNXT: MVI     C,0             ;INITIALIZE LINE LENGTH
0636 CD 1D 06           GETNXT: CALL    GETCHR          ;GET NEXT CHARACTER
0639 DA ?? ??           GETNXT: JC      LINEND          ;HANDLE EOF
063C 47                         MOV     B,A             ;SAVE CHARACTER IN B
063D FE 0D              GETNXT: CPI     NEWLINE         ;END OF LINE?
063F CA ?? ??           GETNXT: JZ      LINFIN          ;YES
0642 3E 52              GETNXT: MVI     A,LINSIZ        ;NO, GET MAXIMUM CHARACTERS PER LINE
0644 B9                 GETNXT: CMP     C               ;LINE BUFFER FULL?
0645 CA ?? ??           GETNXT: JZ      CHOP            ;YES, FORCE NEW LINE
0648 70                 GETNXT: MOV     M,B             ;NO, SAVE CHARACTER IN BUFFER
0649 23                 GETNXT: INX     H               ;POINT TO NEXT CHARACTER POSITION
064A 0C                 GETNXT: INR     C               ;ADJUST # OF CHARACTERS SAVED
064B C3 36 06           GETNXT: JMP     GETNXT          ;GET NEXT CHARACTER
064E
064E CD BA 04   CHOP:   CALL    CHRBAK          ;PUSH CHARACTER BACK ONTO INPUT
0651 36 0D      LINFIN: MVI     M,NEWLINE       ;END LINE WITH CARRIAGE RETURN
0653 23                 INX     H               ;AND
0654 36 00              MVI     M,EOS           ;END OF STRING
0656 EB                 XCHG                    ;RESTORE HL
0657 B7                 ORA     A               ;CLEAR CARRY
0658 C9                 RET
0659
0659 79         LINEND: MOV     A,C             ;BUFFER EMPTY?
065A B1                 ORA     C               ;(COUNT ZERO?)
065B C2 51 06           JNZ     LINFIN          ;NO, PROVIDE EOL
065E 37                 STC                     ;YES, TRUE EOF
065F C9                 RET
0660
0660                    ;------
0660                    ; GETTL - COPY TITLE
0660                    ; ACCEPTS:  DE = @ END OF COMMAND
0660                    ;           HL = @ DESTINATION
0660                    ;------
0660 13         GETTL:  INX     D
0661 1A                 LDAX    D
0662 FE 20              CPI     ' '
0664 CA 60 06           JZ      GETTL
0667 FE 09              CPI     TAB
0669 CA 60 06           JZ      GETTL
066C 13                 INX     D
066D FE 27              CPI     ''''
066F CA ?? ??           JZ      COPYTL
0672 FE 22              CPI     '"'
0674 CA ?? ??           JZ      COPYTL
0677 1B                 DCX     D
0678 CD ?? ??   COPYTL: CALL    SCOPY
067B C9                 RET
067C
067C                    ;------
067C                    ; GETVAL - EVALUATE OPTIONAL NUMERIC ARGUMENT
067C                    ; ACCEPTS:  HL = @ END OF COMMAND
```

```
                        ; RETURNS:  A = ARGTYP
                        ;           HL = ARGUMENT VALUE
                        ;           DE = @ END OF COMMAND
                        ;------
067C E5                 GETVAL: PUSH    H
067D                    FINDARG:
067D 23                         INX     H               ;GET @ NEXT CHARACTER
067E 7E                         MOV     A,M             ;GET NEXT CHARACTER
067F FE 20                      CPI     ' '             ;BLANK?
0681 CA 7D 06                   JZ      FINDARG         ;YES, KEEP SEARCHING FOR ARGUMENT
0684 FE 09                      CPI     TAB             ;NO, TAB?
0686 CA 7D 06                   JZ      FINDARG         ;YES, KEEP SEARCHING
0689 23                         INX     H               ;ASSUME CHARACTER IS '+' OR '-'
068A FE 2B                      CPI     '+'             ;IS IT '+'?
068C CA ?? ??                   JZ      CTOI            ;YES
068F FE 2D                      CPI     '-'             ;NO, IS IT '-'?
0691 CA ?? ??                   JZ      CTOI            ;YES
0694 2B                         DCX     H               ;NO, RESTORE @ ARGUMENT
0695 F5                 CTOI:   PUSH    PSW             ;SAVE ARGTYP
0696 EB                         XCHG                    ;KEEP @ ARGUMENT IN DE
0697 21 00 00                   LXI     H,0             ;INITIALIZE ARGUMENT VALUE
069A 1A                 VALOOP: LDAX    D               ;GET NEXT ARGUMENT CHARACTER
069B 13                         INX     D               ;GET @ NEXT ARGUMENT CHARACTER
069C FE 30                      CPI     '0'             ;< '0'?
069E DA ?? ??                   JC      GOTVAL          ;YES, END OF ARGUMENT
06A1 FE 3A                      CPI     '9'+1           ;NO, > '9'?
06A3 D2 ?? ??                   JNC     GOTVAL          ;YES, END OF ARGUMENT
06A6 44                         MOV     B,H             ;NO, SAVE CURRENT VALUE
06A7 4D                         MOV     C,L             ;IN BC
06A8 29                         DAD     H               ;DOUBLE CURRENT VALUE
06A9 29                         DAD     H               ;QUADRUPLE
06AA 09                         DAD     B               ;QUINTUPLE
06AB 29                         DAD     H               ;DECTUPLE
06AC D6 30                      SUI     '0'             ;CONVERT NEW CHARACTER TO BINARY
06AE 85                         ADD     L               ;ADD TO NEW VALUE
06AF 6F                         MOV     L,A
06B0 D2 9A 06                   JNC     VALOOP
06B3 24                         INR     H
06B4 C3 9A 06                   JMP     VALOOP
06B7
06B7 F1                 GOTVAL: POP     PSW             ;RESTORE ARGTYP
06B8 D1                         POP     D
06B9 C9                         RET
06BA
                        ;------
                        ; GETWRD - GET NEXT WORD
                        ; ACCEPTS:  HL = @ TEXT
                        ; RETURNS:  CARRY SET IF NONE FOUND
                        ;           ELSE HL = @ WORD, EOS
                        ;                DE = @ TEXT FOLLOWING WORD
                        ;------
06BA 7E                 GETWRD: MOV     A,M
06BB 23                         INX     H
06BC FE 20                      CPI     ' '
06BE CA BA 06                   JZ      GETWRD
06C1 FE 09                      CPI     TAB
06C3 CA BA 06                   JZ      GETWRD
06C6 2B                         DCX     H
06C7 54                         MOV     D,H
06C8 5D                         MOV     E,L
06C9 7E                 GWLOOP: MOV     A,M
06CA 23                         INX     H
06CB FE 00                      CPI     EOS
06CD CA ?? ??                   JZ      WRDEND
06D0 FE 0D                      CPI     NEWLINE
06D2 CA ?? ??                   JZ      WRDEND
06D5 FE 09                      CPI     TAB
06D7 CA ?? ??                   JZ      WRDEND
06DA FE 20                      CPI     ' '
06DC C2 C9 06                   JNZ     GWLOOP
06DF 2B                 WRDEND: DCX     H
06E0 36 00                      MVI     M,EOS
06E2 CD EC 05                   CALL    COMPARE
06E5 EB                         XCHG
06E6 3F                         CMC
06E7 13                         INX     D
06E8 C9                         RET
06E9
                        ;------
                        ; INIT - INITIALIZE VARIABLES
                        ;------
06E9 3E 00              INIT:   MVI     A,NO
06EB 32 EA 03                   STA     WAITUP          ;SAVE WAIT OPTION FLAG
06EE 3E 0D                      MVI     A,NEWLINE       ;INITIALIZE
06F0 32 B5 03                   STA     BACKUP          ;BACKUP CHARACTER
06F3 32 81 00                   STA     HEADER
06F6 32 2F 00                   STA     FOOTER
06F9 3E 00                      MVI     A,EOS
06FB 32 82 00                   STA     HEADER+1
06FE 32 30 00                   STA     FOOTER+1
0701 3E FF                      MVI     A,YES
0703 32 BD 03                   STA     FILL
0706 21 00 00                   LXI     H,0
0709 22 BE 03                   SHLD    INVAL
070C 22 E6 03                   SHLD    TIVAL
070F 22 B8 03                   SHLD    CEVAL
0712 22 E8 03                   SHLD    ULVAL
0715 22 C2 03                   SHLD    LINENO
0718 22 BA 03                   SHLD    CURPAG
071B 22 DA 03                   SHLD    OUTP
071E 22 DC 03                   SHLD    OUTW
0721 22 DE 03                   SHLD    OUTWDS
0724 23                         INX     H
0725 22 C6 03                   SHLD    LSVAL
0728 22 D4 03                   SHLD    NEWPAG
072B 23                         INX     H
072C 22 CA 03                   SHLD    M2VAL
072F 22 CC 03                   SHLD    M3VAL
0732 23                         INX     H
0733 22 C8 03                   SHLD    M1VAL
0736 22 CE 03                   SHLD    M4VAL
0739 21 3C 00                   LXI     H,PAGEWIDTH
073C 22 E2 03                   SHLD    RMVAL
073F 21 42 00                   LXI     H,PAGELEN
0742 22 E0 03                   SHLD    PLVAL
0745 EB                         XCHG
0746 2A CC 03                   LHLD    M3VAL
0749 CD FD 05                   CALL    DELESSHL
074C EB                         XCHG
074D 2A CE 03                   LHLD    M4VAL
0750 CD FD 05                   CALL    DELESSHL
0753 22 B6 03                   SHLD    BOTTOM
0756 C9                         RET
0757
                        ;------
                        ; LEADBL - "DELETE" LEADING BLANKS, SET TIVAL
                        ; ACCEPTS:  HL = @ LINE
                        ; RETURNS:  HL = @ FIRST NON-BLANK
                        ;------
0757 CD 70 04           LEADBL: CALL    BRK
075A 11 00 00                   LXI     D,0
075D 7E                 LBLOOP: MOV     A,M
075E 13                         INX     D
075F 23                         INX     H
0760 FE 20                      CPI     ' '
0762 CA 5D 07                   JZ      LBLOOP
0765 2B                         DCX     H
0766 FE 0D                      CPI     NEWLINE
0768 C8                         RZ
0769 EB                         XCHG
076A 22 E6 03                   SHLD    TIVAL
076D EB                         XCHG
076E C9                         RET
076F
                        ;------
                        ; LENGTH - COMPUTE LENGTH OF STRING
                        ; ACCEPTS:  HL = @ STRING
                        ; RETURNS:  DE = LENGTH
                        ;------
076F 11 00 00           LENGTH: LXI     D,0
0772 7E                 LENEXT: MOV     A,M
0773 FE 00                      CPI     EOS
0775 C8                         RZ
0776 13                         INX     D
0777 23                         INX     H
0778 C3 72 07                   JMP     LENEXT
077B
                        ;------
                        ; MAX - DETERMINE LARGER OF TWO NUMBERS
                        ; ACCEPTS:  NUMBERS IN DE, HL
                        ; RETURNS:  HL = LARGER
                        ;           DE = SMALLER
                        ;------
077B CD ?? ??           MAX:    CALL    MIN
077E EB                         XCHG
077F C9                         RET
0780
                        ;------
                        ; MIN - DETERMINE SMALLER OF TWO NUMBERS
                        ; ACCEPTS:  NUMBERS IN DE, HL
```

```
                        ; RETURNS:  DE = LARGER
                        ;           HL = SMALLER
                        ;------
0780 CD EC 05           MIN:    CALL    COMPARE
0783 D0                         RNC
0784 EB                         XCHG
0785 C9                         RET
                        ;------
                        ; PFOOT - PUT OUT PAGE FOOTER
                        ;------
0786 2A CC 03           PFOOT:  LHLD    M3VAL
0789 CD ?? ??                   CALL    SKIP
078C 11 00 00                   LXI     D,0
078F 2A CE 03                   LHLD    M4VAL
0792 CD EC 05                   CALL    COMPARE
0795 D0                         RNC
0796 11 2F 00                   LXI     D,FOOTER
0799 2A BA 03                   LHLD    CURPAG
079C CD ?? ??                   CALL    PUTTL
079F 2A CE 03                   LHLD    M4VAL
07A2 2B                         DCX     H
07A3 CD ?? ??                   CALL    SKIP
07A6 C9                         RET
                        ;------
                        ; PHEAD - PUT OUT PAGE HEADER
                        ;------
07A7 D5                 PHEAD:  PUSH    D
07A8 E5                         PUSH    H
07A9 3A EA 03                   LDA     WAITUP          ;GET W: OPTION FLAG
07AC FE FF                      CPI     YES             ;WAIT FOR OPERATOR?
07AE CC --- ---                 CZ      $WAIT           ;YES, WAIT FOR CONSOLE INPUT
07B1 2A D4 03                   LHLD    NEWPAG
07B4 22 BA 03                   SHLD    CURPAG
07B7 23                         INX     H
07B8 22 D4 03                   SHLD    NEWPAG
07BB 2A C8 03                   LHLD    M1VAL
07BE E5                         PUSH    H
07BF 11 00 00                   LXI     D,0
07C2 CD EC 05                   CALL    COMPARE
07C5 D2 ?? ??                   JNC     SKPHDR
07C8 2B                         DCX     H
07C9 CD ?? ??                   CALL    SKIP
07CC 11 81 00                   LXI     D,HEADER
07CF 2A BA 03                   LHLD    CURPAG
07D2 CD ?? ??                   CALL    PUTTL
07D5 2A CA 03           SKPHDR: LHLD    M2VAL
07D8 E5                         PUSH    H
07D9 CD ?? ??                   CALL    SKIP
07DC E1                         POP     H
07DD D1                         POP     D
07DE 19                         DAD     D
07DF 23                         INX     H
07E0 22 C2 03                   SHLD    LINENO
07E3 E1                         POP     H
07E4 D1                         POP     D
07E5 C9                         RET
                        ;------
                        ; PUT - PUT OUT LINE WITH PROPER SPACING AND INDENTING
                        ; ACCEPTS:  HL = @ LINE
                        ;------
07E6 E5                 PUT:    PUSH    H
07E7 2A C2 03                   LHLD    LINENO
07EA 7C                         MOV     A,H
07EB B5                         ORA     L
07EC C2 ?? ??                   JNZ     NOT0
07EF CD A7 07                   CALL    PHEAD
07F2 C3 ?? ??                   JMP     HEADED
07F5
07F5 EB                 NOT0:   XCHG
07F6 2A B6 03                   LHLD    BOTTOM
07F9 EB                         XCHG
07FA CD EC 05                   CALL    COMPARE
07FD D2 ?? ??                   JNC     HEADED
0800 CD A7 07                   CALL    PHEAD
0803 2A E6 03           HEADED: LHLD    TIVAL
0806 7C                 TITEST: MOV     A,H
0807 B5                         ORA     L
0808 CA ?? ??                   JZ      TIZERO
080B 3E 20                      MVI     A,' '
080D CD --- ---                 CALL    $PUTC
0810 2B                         DCX     H
0811 C3 06 08                   JMP     TITEST
0814
0814 2A BE 03           TIZERO: LHLD    INVAL
0817 22 E6 03                   SHLD    TIVAL
081A E1                         POP     H
081B CD ?? ??                   CALL    PUTLIN
081E 2A B6 03                   LHLD    BOTTOM
0821 EB                         XCHG
0822 2A C2 03                   LHLD    LINENO
0825 CD FD 05                   CALL    DELESSHL
0828 EB                         XCHG
0829 2A C6 03                   LHLD    LSVAL
082C 2B                         DCX     H
082D CD 80 07                   CALL    MIN
0830 CD ?? ??                   CALL    SKIP
0833 2A C6 03                   LHLD    LSVAL
0836 EB                         XCHG
0837 2A C2 03                   LHLD    LINENO
083A 19                         DAD     D
083B 22 C2 03                   SHLD    LINENO
083E EB                         XCHG
083F 2A B6 03                   LHLD    BOTTOM
0842 EB                         XCHG
0843 CD EC 05                   CALL    COMPARE
0846 DC 86 07                   CC      PFOOT
0849 C9                         RET
084A
                        ;------
                        ; PUTDEC - OUTPUT DECIMAL NUMBER
                        ; ACCEPTS:  HL = NUMBER
                        ;------
084A D5                 PUTDEC: PUSH    D
084B E5                         PUSH    H
084C 11 ?? ??                   LXI     D,TENTAB
084F D5                         PUSH    D
0850 0E 01                      MVI     C,1
0852 E3                 DOTPOS: XTHL
0853 5E                         MOV     E,M
0854 23                         INX     H
0855 56                         MOV     D,M
0856 23                         INX     H
0857 E3                         XTHL
0858 06 FF                      MVI     B,-1
085A 04                 DOTDIV: INR     B
085B 7D                         MOV     A,L
085C 93                         SUB     E
085D 6F                         MOV     L,A
085E 7C                         MOV     A,H
085F 9A                         SBB     D
0860 67                         MOV     H,A
0861 D2 5A 08                   JNC     DOTDIV
0864 19                         DAD     D
0865 AF                         XRA     A
0866 B0                         ORA     B
0867 C2 ?? ??                   JNZ     DOTPRI
086A B1                         ORA     C
086B C2 ?? ??                   JNZ     DOTSKP
086E F6 30              DOTPRI: ORI     '0'
0870 0E 00                      MVI     C,0
0872 CD --- ---                 CALL    $PUTC
0875 7B                 DOTSKP: MOV     A,E
0876 FE 01                      CPI     1
0878 C2 52 08                   JNZ     DOTPOS
087B D1                         POP     D
087C E1                         POP     H
087D D1                         POP     D
087E C9                         RET
087F
087F 10 27              TENTAB: @ 10000
0881 E8 03                      @ 1000
0883 64 00                      @ 100
0885 0A 00                      @ 10
0887 01 00                      @ 1
0889
                        ;------
                        ; PUTLIN - OUTPUT LINE
                        ; ACCEPTS:  HL = @ LINE
                        ;------
0889 7E                 PUTLIN: MOV     A,M
088A FE 00                      CPI     EOS
088C C8                         RZ
088D CD --- ---                 CALL    $PUTC
0890 23                         INX     H
0891 C3 89 08                   JMP     PUTLIN
```

```
0894                    ;------
0894                    ; PUTTL - PUT OUT TITLE LINE WITH OPTIONAL PAGE NUMBER
0894                    ; ACCEPTS:  DE = @ LINE
0894                    ;           HL = PAGE #
0894                    ;------
0894 1A         PUTTL:  LDAX    D
0895 13                 INX     D
0896 FE 00              CPI     EOS
0898 C8                 RZ
0899 FE 23              CPI     PAGENUM
089B C2 ?? ??           JNZ     TLCOPY
089E CD 4A 08           CALL    PUTDEC
08A1 C3 94 08           JMP     PUTTL
08A4
08A4 CD --- ---  TLCOPY: CALL   $PUTC
08A7 C3 94 08           JMP     PUTTL
08AA
08AA                    ;------
08AA                    ; PUTWRD - PUT A WORD IN OUTBUF
08AA                    ; ACCEPTS:  HL = @ WORD
08AA                    ;------
08AA D5         PUTWRD: PUSH    D
08AB E5                 PUSH    H
08AC CD ?? ??           CALL    WIDTH
08AF 42                 MOV     B,D             ;BC := W
08B0 4B                 MOV     C,E
08B1 E1                 POP     H
08B2 E5                 PUSH    H
08B3 CD 6F 07           CALL    LENGTH
08B6 2A DA 03           LHLD    OUTP
08B9 19                 DAD     D
08BA 23                 INX     H
08BB 22 C0 03           SHLD    LAST
08BE 2A E2 03           LHLD    RMVAL
08C1 EB                 XCHG
08C2 2A E6 03           LHLD    TIVAL
08C5 CD FD 05           CALL    DELESSHL
08C8 22 C4 03           SHLD    LLVAL
08CB EB                 XCHG
08CC 2A DC 03           LHLD    OUTW
08CF 09                 DAD     B
08D0 CD EC 05           CALL    COMPARE
08D3 DA ?? ??           JC      MAYFIT
08D6 2A C0 03           LHLD    LAST
08D9 EB                 XCHG
08DA 21 F6 00           LXI     H,TRIPLE
08DD CD EC 05           CALL    COMPARE
08E0 DA ?? ??           JC      FITS
08E3 11 00 00   MAYFIT: LXI     D,0
08E6 2A DA 03           LHLD    OUTP
08E9 CD EC 05           CALL    COMPARE
08EC D2 ?? ??           JNC     FITS
08EF EB                 XCHG
08F0 2A C0 03           LHLD    LAST
08F3 EB                 XCHG
08F4 CD FD 05           CALL    DELESSHL
08F7 22 C0 03           SHLD    LAST
08FA 2A C4 03           LHLD    LLVAL
08FD EB                 XCHG
08FE 2A DC 03           LHLD    OUTW
0901 CD FD 05           CALL    DELESSHL
0904 23                 INX     H
0905 22 D6 03           SHLD    NEXTRA
0908 C5                 PUSH    B
0909 E5                 PUSH    H
090A CD ?? ??           CALL    SPREAD
090D E1                 POP     H
090E C1                 POP     B
090F 11 00 00           LXI     D,0
0912 CD EC 05           CALL    COMPARE
0915 D2 ?? ??           JNC     FLUSHIT
0918 13                 INX     D
0919 2A DE 03           LHLD    OUTWDS
091C CD EC 05           CALL    COMPARE
091F D2 ?? ??           JNC     FLUSHIT
0922 2A D6 03           LHLD    NEXTRA
0925 EB                 XCHG
0926 2A DA 03           LHLD    OUTP
0929 19                 DAD     D
092A 22 DA 03           SHLD    OUTP
092D            FLUSHIT:
092D CD 70 04           CALL    BRK
0930 11 C9 01   FITS:   LXI     D,OUTBUF
0933 2A DA 03           LHLD    OUTP
0936 19                 DAD     D
0937 D1                 POP     D
0938 CD ?? ??           CALL    SCOPY
093B 2A C0 03           LHLD    LAST
093E 22 DA 03           SHLD    OUTP
0941 11 C9 01           LXI     D,OUTBUF
0944 19                 DAD     D
0945 2B                 DCX     H
0946 36 20              MVI     M,' '
0948 2A DC 03           LHLD    OUTW
094B 09                 DAD     B
094C 23                 INX     H
094D 22 DC 03           SHLD    OUTW
0950 2A DE 03           LHLD    OUTWDS
0953 23                 INX     H
0954 22 DE 03           SHLD    OUTWDS
0957 D1                 POP     D
0958 C9                 RET
0959
0959                    ;------
0959                    ; SCOPY - COPY STRINGS
0959                    ; ACCEPTS:  DE = @ SOURCE
0959                    ;           HL = @ DESTINATION
0959                    ;------
0959 1A         SCOPY:  LDAX    D
095A 77                 MOV     M,A
095B FE 00              CPI     EOS
095D C8                 RZ
095E 13                 INX     D
095F 23                 INX     H
0960 C3 59 09           JMP     SCOPY
0963
0963                    ;------
0963                    ; SEARCH - SEARCH TABLE FOR A COMMAND
0963                    ; ACCEPTS:  BC = @ TABLE END
0963                    ;           DE = @ TABLE
0963                    ;           HL = @ COMMAND
0963                    ; RETURNS:  CARRY SET IF COMMAND NOT IN TABLE
0963                    ;           ELSE DE = ADDRESS FROM TABLE
0963                    ;                HL = @ END OF COMMAND
0963                    ;------
0963 E5         SEARCH: PUSH    H               ;SAVE @ COMMAND
0964 1A         NXTCHR: LDAX    D               ;GET NEXT CHARACTER FROM TABLE
0965 BE                 CMP     M               ;SAME AS COMMAND CHARACTER?
0966 CA ?? ??           JZ      MATCH           ;YES
0969 F6 20              ORI     ' '             ;NO, TRY LOWER CASE
096B BE                 CMP     M               ;SAME AS COMMAND?
096C 1A                 LDAX    D               ;(RESTORE UPPER CASE CHARACTER)
096D CA ?? ??           JZ      MATCH           ;YES
0970
0970
0970
0970 FE 00              CPI     EOS             ;END OF TABLE ENTRY?
0972 C2 ?? ??           JNZ     NXTENT          ;NO, REALLY NOT EQUAL, TRY NEXT TABLE
0975 7E                 MOV     A,M             ;GET COMMAND CHARACTER          ENTR
0976 CD 61 04           CALL    ALPHA           ;IS IT A LETTER?
0979 DA ?? ??           JC      NXTENT          ;YES, COMMAND IS LONGER
097C
097C
097C
097C E3         FOUND:  XTHL                    ;SAVE @ COMMAND TERMINATOR ON STACK
097D EB                 XCHG                    ;MOVE TABLE POINTER TO HL
097E 23                 INX     H               ;POINT PAST EOS
097F 5E                 MOV     E,M             ;GET ADDRESS FROM TABLE INTO DE
0980 23                 INX     H
0981 56                 MOV     D,M
0982 E1                 POP     H               ;RESTORE @ COMMAND TERMINATOR
0983 2B                 DCX     H               ;GET @ END OF COMMAND
0984 B7                 ORA     A               ;CLEAR CARRY
0985 C9                 RET
0986
0986
0986
0986 13         MATCH:  INX     D               ;POINT TO NEXT
0987 23                 INX     H               ;PAIR OF CHARACTERS
0988 7E                 MOV     A,M             ;GET NEXT COMMAND CHARACTER
0989 CD 61 04           CALL    ALPHA           ;END OF COMMAND?
098C DA 64 09           JC      NXTCHR          ;NO, CONTINUE COMPARING
098F 1A         GETEOS: LDAX    D               ;YES,
0990 13                 INX     D               ;FIND TABLE ENTRY'S
0991 FE 00              CPI     EOS             ;END OF STRING
0993 C2 8F 09           JNZ     GETEOS
0996 1B                 DCX     D
0997 C3 7C 09           JMP     FOUND
```

```
1012 099A
1013 099A                ; IGNORE REST OF THIS TABLE ENTRY.  MOVE DE TO START OF NEXT ENTRY
1014 099A
1015 099A E1             NXTENT: POP     H               ;RESTORE @ COMMAND
1016 099B 1A             NXLOOP: LDAX    D               ;GET CURRENT TABLE CHARACTER
1017 099C 13                     INX     D               ;POINT TO NEXT CHARACTER
1018 099D FE 00                  CPI     EOS             ;END OF CURRENT TABLE STRING?
1019 099F C2 9B 09               JNZ     NXLOOP          ;NO, KEEP LOOKING
1020 09A2 13                     INX     D               ;YES, SKIP PAST ADDRESS FIELD
1021 09A3 13                     INX     D
1022 09A4 78                     MOV     A,B             ;REACHED END OF TABLE?
1023 09A5 BA                     CMP     D               ;(BC = DE?)
1024 09A6 C2 63 09               JNZ     SEARCH          ;NO, RESUME SEARCH
1025 09A9 79                     MOV     A,C             ;MAYBE...
1026 09AA BB                     CMP     E
1027 09AB C2 63 09               JNZ     SEARCH          ;NO, RESUME SEARCH
1028 09AE 37                     STC                     ;YES, COMMAND NOT FOUND
1029 09AF C9                     RET
1030 09B0
1031 09B0                ;------
1032 09B0                ; SET - SET PARAMETER AND CHECK RANGE
1033 09B0                ; ACCEPTS:  A = ARGTYP
1034 09B0                ;           BC = DEFAULT VALUE
1035 09B0                ;           DE = ARGUMENT VALUE
1036 09B0                ;           HL = PARAMETER VALUE
1037 09B0                ;           TOP OF STACK = MAXIMUM VALUE
1038 09B0                ;           NEXT STACKED = MINIMUM VALUE
1039 09B0                ; RETURNS: HL = NEW PARAMETER VALUE
1040 09B0                ;------
1041 09B0 FE 0D          SET:    CPI     NEWLINE         ;IS THERE AN ARGUMENT?
1042 09B2 CA ?? ??               JZ      DEFAULTED       ;NO
1043 09B5 FE 2B                  CPI     '+'             ;YES, INCREASE?
1044 09B7 CA ?? ??               JZ      RELPLUS         ;YES
1045 09BA FE 2D                  CPI     '-'             ;NO, DECREASE?
1046 09BC EB                     XCHG                    ;(ASSUME NOT: USE ABSOLUTE ARGUMENT)
1047 09BD CA ?? ??               JZ      RELMINUS        ;YES
1048 09C0 C1             MINMAX: POP     B               ;GET RETURN ADDRESS
1049 09C1 D1                     POP     D               ;HL = PARAM, GET MAXVAL
1050 09C2 CD 80 07               CALL    MIN             ;USE SMALLER OF MAXVAL AND PARAM
1051 09C5 D1                     POP     D               ;GET MINVAL
1052 09C6 CD 7B 07               CALL    MAX             ;USE LARGER OF MINVAL AND PARAM
1053 09C9 C5                     PUSH    B
1054 09CA C9                     RET
1055 09CB
1056 09CB                DEFAULTED:
1057 09CB 60                     MOV     H,B             ;USE DEFVAL
1058 09CC 69                     MOV     L,C             ;FOR PARAM
1059 09CD C3 C0 09               JMP     MINMAX
1060 09D0
1061 09D0                RELPLUS:
1062 09D0 19                     DAD     D               ;INCREASE PARAM
1063 09D1 C3 C0 09               JMP     MINMAX
1064 09D4
1065 09D4                RELMINUS:
1066 09D4 CD FD 05               CALL    DELESSHL        ;DECREASE PARAM
1067 09D7 C3 C0 09               JMP     MINMAX
1068 09DA
1069 09DA                ;------
1070 09DA                ; SKIP - OUTPUT BLANK LINES
1071 09DA                ; ACCEPTS:  HL = # OF LINES TO SKIP
1072 09DA                ;------
1073 09DA 7C             SKIP:   MOV     A,H
1074 09DB B5                     ORA     L
1075 09DC C8                     RZ
1076 09DD 3E 0D                  MVI     A,NEWLINE
1077 09DF CD -- --               CALL    $PUTC
1078 09E2 2B                     DCX     H
1079 09E3 C3 DA 09               JMP     SKIP
1080 09E6
1081 09E6                ;------
1082 09E6                ; SPACE - SPACE SOME LINES OR TO BOTTOM OF PAGE
1083 09E6                ; ACCEPTS: HL = # OF LINES
1084 09E6                ;------
1085 09E6 CD 70 04       SPACE:  CALL    BRK.
1086 09E9 44                     MOV     B,H
1087 09EA 4D                     MOV     C,L
1088 09EB 2A B6 03               LHLD    BOTTOM
1089 09EE EB                     XCHG
1090 09EF 2A C2 03               LHLD    LINENO
1091 09F2 CD EC 05               CALL    COMPARE
1092 09F5 D8                     RC
1093 09F6 C5                     PUSH    B
1094 09F7 7C                     MOV     A,H
1095 09F8 B5                     ORA     L
1096 09F9 CC A7 07               CZ      PHEAD
1097 09FC 13                     INX     D
1098 09FD CD FD 05               CALL    DELESSHL
1099 0A00 C1                     POP     B
1100 0A01 50                     MOV     D,B
1101 0A02 59                     MOV     E,C
1102 0A03 CD 80 07               CALL    MIN
1103 0A06 CD DA 09               CALL    SKIP
1104 0A09 2A C2 03               LHLD    LINENO
1105 0A0C 09                     DAD     B
1106 0A0D 22 C2 03               SHLD    LINENO
1107 0A10 EB                     XCHG
1108 0A11 2A B6 03               LHLD    BOTTOM
1109 0A14 EB                     XCHG
1110 0A15 CD EC 05               CALL    COMPARE
1111 0A18 DC 86 07               CC      PFOOT
1112 0A1B C9                     RET
1113 0A1C
1114 0A1C                ;------
1115 0A1C                ; SPREAD - SPREAD WORDS TO JUSTIFY RIGHT MARGIN
1116 0A1C                ;------
1117 0A1C 11 00 00       SPREAD: LXI     D,0
1118 0A1F 2A D6 03               LHLD    NEXTRA
1119 0A22 CD EC 05               CALL    COMPARE
1120 0A25 D0                     RNC
1121 0A26 13                     INX     D
1122 0A27 2A DE 03               LHLD    OUTWDS
1123 0A2A CD EC 05               CALL    COMPARE
1124 0A2D D0                     RNC
1125 0A2E 3A BC 03               LDA     DIR
1126 0A31 2F                     CMA
1127 0A32 32 BC 03               STA     DIR
1128 0A35 2B                     DCX     H
1129 0A36 22 D8 03               SHLD    NHOLES
1130 0A39 2A D6 03               LHLD    NEXTRA
1131 0A3C 22 D2 03               SHLD    NE
1132 0A3F EB                     XCHG
1133 0A40 2A DA 03               LHLD    OUTP
1134 0A43 2B                     DCX     H
1135 0A44 E5                     PUSH    H
1136 0A45 19                     DAD     D
1137 0A46 11 F4 00               LXI     D,TRIPLE-2
1138 0A49 CD 80 07               CALL    MIN
1139 0A4C 11 C9 01               LXI     D,OUTBUF
1140 0A4F 19                     DAD     D
1141 0A50 E3                     XTHL
1142 0A51 19                     DAD     D
1143 0A52 D1                     POP     D
1144 0A53 EB                     XCHG
1145 0A54 1B             SPLOOP: DCX     D
1146 0A55 2B                     DCX     H
1147 0A56 CD EC 05               CALL    COMPARE
1148 0A59 D0                     RNC
1149 0A5A 1A                     LDAX    D
1150 0A5B 77                     MOV     M,A
1151 0A5C FE 20                  CPI     ' '
1152 0A5E C2 54 0A               JNZ     SPLOOP
1153 0A61 D5                     PUSH    D
1154 0A62 E5                     PUSH    H
1155 0A63 3A BC 03               LDA     DIR
1156 0A66 F5                     PUSH    PSW
1157 0A67 2A D2 03               LHLD    NE
1158 0A6A FE 00                  CPI     0
1159 0A6C C2 ?? ??               JNZ     SP1
1160 0A6F 2B                     DCX     H
1161 0A70 EB             SP1:    XCHG
1162 0A71 2A D8 03               LHLD    NHOLES
1163 0A74 CD 0D 06               CALL    DIVIDE
1164 0A77 F1                     POP     PSW
1165 0A78 FE 00                  CPI     0
1166 0A7A C2 ?? ??               JNZ     SP2
1167 0A7D 23                     INX     H
1168 0A7E 22 D0 03       SP2:    SHLD    NB
1169 0A81 EB                     XCHG
1170 0A82 2A D2 03               LHLD    NE
1171 0A85 EB                     XCHG
1172 0A86 CD FD 05               CALL    DELESSHL
1173 0A89 22 D2 03               SHLD    NE
1174 0A8C 2A D8 03               LHLD    NHOLES
1175 0A8F 2B                     DCX     H
1176 0A90 22 D8 03               SHLD    NHOLES
1177 0A93 11 00 00               LXI     D,0
1178 0A96 2A D0 03               LHLD    NB
1179 0A99 CD EC 05       SP3:    CALL    COMPARE
```

```
0A9C D2 ?? ??           JNC     SP4
0A9F E3                 XTHL
0AA0 2B                 DCX     H
0AA1 36 20              MVI     M,' '
0AA3 E3                 XTHL
0AA4 2B                 DCX     H
0AA5 C3 99 0A           JMP     SP3
0AA8 E1         SP4:    POP     H
0AA9 D1                 POP     D
0AAA C3 54 0A           JMP     SPLOOP
0AAD
0AAD            ;------
0AAD            ; TEXT - PROCESS TEXT LINES
0AAD            ; ACCEPTS:  HL = @ LINE
0AAD            ;------
0AAD 7E         TEXT:   MOV     A,M
0AAE FE 20              CPI     ' '
0AB0 CA ?? ??           JZ      MOVLFT
0AB3 FE 0D              CPI     NEWLINE
0AB5 C2 ?? ??           JNZ     UNDERLINING
0AB8 CD 57 07   MOVLFT: CALL    LEADBL
0ABB            UNDERLINING:
0ABB E5                 PUSH    H
0ABC 11 00 00           LXI     D,0
0ABF 2A E8 03           LHLD    ULVAL
0AC2 CD EC 05           CALL    COMPARE
0AC5 D2 ?? ??           JNC     CENTERING
0AC8 21 F6 00           LXI     H,TRIPLE
0ACB 11 BF 02           LXI     D,WRDBUF
0ACE C1                 POP     B
0ACF C5                 PUSH    B
0AD0 CD ?? ??           CALL    UNDERL
0AD3 2A E8 03           LHLD    ULVAL
0AD6 2B                 DCX     H
0AD7 22 E8 03           SHLD    ULVAL
0ADA            CENTERING:
0ADA 11 00 00           LXI     D,0
0ADD 2A B8 03           LHLD    CEVAL
0AE0 CD EC 05           CALL    COMPARE
0AE3 D2 ?? ??           JNC     ALLBLANK
0AE6 E1                 POP     H
0AE7 CD 9C 04           CALL    CENTER
0AEA CD E6 07           CALL    PUT
0AED 2A B8 03           LHLD    CEVAL
0AF0 2B                 DCX     H
0AF1 22 B8 03           SHLD    CEVAL
0AF4 C9                 RET
0AF5
0AF5            ALLBLANK:
0AF5 E1                 POP     H
0AF6 7E                 MOV     A,M
0AF7 FE 0D              CPI     NEWLINE
0AF9 C2 ?? ??           JNZ     UNFILLED
0AFC CD E6 07           CALL    PUT
0AFF C9                 RET
0B00
0B00            UNFILLED:
0B00 3A BD 03           LDA     FILL
0B03 FE 00              CPI     NO
0B05 C2 ?? ??           JNZ     FILLED
0B08 CD E6 07           CALL    PUT
0B0B C9                 RET
0B0C
0B0C CD BA 06   FILLED: CALL    GETWRD
0B0F D8                 RC
0B10 CD AA 08           CALL    PUTWRD
0B13 EB                 XCHG
0B14 C3 0C 0B           JMP     FILLED
0B17
0B17            ;------
0B17            ; UNDERL - UNDERLINE A LINE
0B17            ; ACCEPTS:  BC = @ LINE
0B17            ;           DE = @ TEMPORARY BUFFER
0B17            ;           HL = SIZE OF LINE BUFFERS
0B17            ;------
0B17 C5         UNDERL: PUSH    B
0B18 D5                 PUSH    D
0B19 19                 DAD     D               ;GET @ END OF TEMP BUF + 1
0B1A 2B                 DCX     H               ;GET @ LAST POSSIBLE CHAR POSITION
0B1B 2B                 DCX     H               ;(NEWLINE & EOS TAKE LAST TWO)
0B1C 2B                 DCX     H
0B1D 2B                 DCX     H
0B1E 2B                 DCX     H
0B1F EB                 XCHG                    ;DE := @ LAST, HL := @ TEMP BUF
```

```
0B20 0A         UNLOOP: LDAX    B
0B21 FE 0D              CPI     NEWLINE
0B23 CA ?? ??           JZ      ULINED
0B26 CD EC 05           CALL    COMPARE
0B29 DA ?? ??           JC      ULINED
0B2C 0A                 LDAX    B
0B2D 03                 INX     B
0B2E 77                 MOV     M,A
0B2F 23                 INX     H
0B30 FE 20              CPI     ' '
0B32 CA 20 0B           JZ      UNLOOP
0B35 FE 09              CPI     TAB
0B37 CA 20 0B           JZ      UNLOOP
0B3A FE 08              CPI     BACKSPACE
0B3C CA 20 0B           JZ      UNLOOP
0B3F 36 08              MVI     M,BACKSPACE
0B41 23                 INX     H
0B42 36 5F              MVI     M,'_'
0B44 23                 INX     H
0B45 C3 20 0B           JMP     UNLOOP
0B48
0B48 36 0D      ULINED: MVI     M,NEWLINE
0B4A 23                 INX     H
0B4B 36 00              MVI     M,EOS
0B4D D1                 POP     D
0B4E E1                 POP     H
0B4F CD 59 09           CALL    SCOPY
0B52 C9                 RET
0B53
0B53            ;------
0B53            ; WIDTH - COMPUTE WIDTH OF CHARACTER STRING
0B53            ; ACCEPTS:  HL = @ STRING
0B53            ; RETURNS:  DE = WIDTH
0B53            ;------
0B53 11 00 00   WIDTH:  LXI     D,0
0B56 7E         WIDNXT: MOV     A,M
0B57 FE 00              CPI     EOS
0B59 C8                 RZ
0B5A 23                 INX     H
0B5B FE 08              CPI     BACKSPACE
0B5D CA ?? ??           JZ      NEGWID
0B60 FE 0D              CPI     NEWLINE
0B62 CA 56 0B           JZ      WIDNXT
0B65 13                 INX     D
0B66 C3 56 0B           JMP     WIDNXT
0B69
0B69 1B         NEGWID: DCX     D
0B6A C3 56 0B           JMP     WIDNXT
     ----- $GETC
     ----- $PUTC
     ----- $QUIT
     ----- $WAIT
     0007 A
     0AF5 ALLBLANK
     0461 ALPHA
     0000 B
     0008 BACKSPACE
     03B5 BACKUP
     03B6 BOTTOM
     04CF BREAK!
     0470 BRK
     0001 C
     049C CENTER
     04D3 CENTER!
     0ADA CENTERING
     03B8 CEVAL
     064E CHOP
     04BA CHRBAK
     04BF COMAND
     002E COMMAND
     03EB COMMANDS
     05EC COMPARE
     0678 COPYTL
     0695 CTOI
     03BA CURPAG
     0002 D
     09CB DEFAULTED
     05FD DELESSHL
     03BC DIR
     0604 DIVBY2
     060D DIVIDE
     085A DOTDIV
     0852 DOTPOS
     086E DOTPRI
     0875 DOTSKP
     0612 DVLOOP
     0003 E
     048C EMPTY
     0461 ENDCOM
     0000 EOS
     03BD FILL
     0B0C FILLED
     067D FINDARG
     0930 FITS
     001E FLUSH
     092D FLUSHIT
     002F FOOTER
     04EA FOOTER!
     0000 FORMAT
     097C FOUND
     062E GETC
     061D GETCHR
     098F GETEOS
     0632 GETLIN
     0636 GETNXT
     0660 GETTL
     067C GETVAL
     06BA GETWRD
     06B7 GOTVAL
     06C9 GWLOOP
     0004 H
     04F1 HANG!
     0803 HEADED
     0081 HEADER
     0504 HEADER!
     03E8 HUGE
     00D3 INBUF
     050B INDENT!
     06E9 INIT
     03BE INVAL
     0018 ITS_TEXT
     0522 JAGGED!
     052B JUSTIFY!
     0005 L
     03C0 LAST
     075D LBLOOP
     0757 LEADBL
     0772 LENEXT
     076F LENGTH
     0534 LENGTH!
     0659 LINEND
     03C2 LINENO
     0651 LINFIN
     0052 LINSIZ
     03C4 LLVAL
     03C6 LSVAL
     0006 M
     03C8 M1VAL
     03CA M2VAL
     03CC M3VAL
     03CE M4VAL
     0003 MAIN_LOOP
     056C MARGIN!
     0986 MATCH
     077B MAX
     08E3 MAYFIT
     0780 MIN
     09C0 MINMAX
```

```
0AB8 MOVLFT
03D0 NB
03D2 NE
0B69 NEGWID
000D NEWLINE
03D4 NEWPAG
03D6 NEXTRA
03D8 NHOLES
0000 NO
07F5 NOT0
0469 NOTLC
099B NXLOOP
0964 NXTCHR
099A NXTENT
01C9 OUTBUF
03DA OUTP
03DC OUTW
03DE OUTWDS
0583 PAGE!
0042 PAGELEN
0023 PAGENUM
003C PAGEWIDTH
0786 PFOOT
07A7 PHEAD
03E0 PLVAL
0006 PSW
07E6 PUT
084A PUTDEC
0889 PUTLIN
0894 PUTTL
08AA PUTWRD
09D4 RELMINUS
09D0 RELPLUS
03E2 RMVAL
0959 SCOPY
0963 SEARCH
09B0 SET
05FA SIGNEQ
09DA SKIP
05AE SKIP!
07D5 SKPHDR
0007 SP
0A70 SP1
0A7E SP2
0A99 SP3
0AA8 SP4
09E6 SPACE
05C5 SPACE!
0A54 SPLOOP
0A1C SPREAD
03E4 SPVAL
0009 TAB
087F TENTAB
0AAD TEXT
0806 TITEST
03E6 TIVAL
0814 TIZERO
08A4 TLCOPY
00F6 TRIPLE
0B48 ULINED
03E8 ULVAL
05D8 UNDER!
0B17 UNDERL
0ABB UNDERLINING
0B00 UNFILLED
0B20 UNLOOP
069A VALOOP
03EA WAITUP
0B56 WIDNXT
0B53 WIDTH
02BF WRDBUF
06DF WRDEND
FFFF YES

$GETC 18 442
$PUTC 19 756 817 839 857 1077
$QUIT 17 54 57
$WAIT 16 706
ALLBLANK 1219 1228
ALPHA 119 983 1004
BACKSPACE 21 1277 1279 1303
BACKUP 65 180 431 437 588
BOTTOM 66 284 625 746 764 779 1088 1108
BREAK 97 97 196
BRK 133 196 200 246 252 633 924 1085
CENTER 98 98 160 199 1221
CENTERING 1206 1215
CEVAL 67 207 209 599 1217 1223 1225
CHOP 461 467
CHRBAK 179 467
COMAND 44 186
COMMAND 22 41
COMMANDS 95 187
COMPARE 138 304 364 576 677 689 714 748 781 884 889 893 912 916 1091 1110 1119 1123
COPYTL 494 496 498 1147 1179 1205 1218 1267
CTOI 518 520 522
CURPAG 68 313 318 602 692 708 719
DEFAULTED 1042 1056
DELESSHL 164 280 283 384 621 624 767 879 898 903 1066 1098 1172
DIR 69 1125 1127 1155
DIVBY2 168 397
DIVIDE 413 1163
DOTDIV 801 808
DOTPOS 794 820
DOTPRI 812 815
DOTSKP 814 818
DVLOOP 416 424
EMPTY 139 147
ENDCOM 112 186
EOS 23 97 98 99 100 101 102 103 104 105 106 107 108 109 110 142 470 566 575 591 655
EXTRN 16 17 18 19 837 850 953 980 1008 1018 1287 1300
FILL 70 248 254 595 1237
FILLED 1239 1243 1247
FINDARG 509 513 515
FITS 890 894 925
FLUSH 40 51
FLUSHIT 913 917 923
FOOTER 59 99 99 212 213 590 593 691
FORMAT 35
FOUND 988 1011
GETC 434 442
GETCHR 431 454
GETEOS 1006 1009
GETLIN 39 451
GETNXT 454 465
GETTL 214 229 486 489 491
GETVAL 192 508
GETWRD 555 558 560 1243
GOTVAL 528 530 544
GWLOOP 564 573
HANG 100 100 217
HEADED 743 749 751
HEADER 60 101 101 227 228 589 592 718
HUGE 24 55 204 271 292 305 309 311 325 336 347
INBUF 38 61
INDENT 102 102 232
INIT 35 585
INVAL 71 239 241 597 760
ITS TEXT 43 47
JAGGED 103 103 245
JUSTIFY 104 104 251
LAST 72 875 886 896 899 930
LBLOOP 635 639
LEADBL 633 1200
LENEXT 654 659
LENGTH 105 105 257 653 871
LINEND 455 475
LINENO 51 73 303 601 728 738 766 775 777 1090 1104 1106
LINFIN 458 468 477
LINSIZ 25 32 59 60 459
LLVAL 74 880 900
LSVAL 75 339 341 607 769 773
M1VAL 76 259 613 711
M2VAL 77 261 610 721
M3VAL 78 264 279 611 620 685
M4VAL 79 267 282 614 623 688 694
MAIN_LOOP 37 45 49
MARGIN 106 106 287
MATCH 971 975 1001
MAX 170 667 1052
MAYFIT 885 891
MIN 667 677 771 1050 1102 1138
MINMAX 1048 1059 1063 1067
MOVLFT 1197 1200
NB 80 1168 1178
NE 81 1131 1157 1170 1173
NEGWID 1304 1310
NEWLINE 26 144 432 436 457 468 568 587 641 1041 1076 1198 1231 1265 1285 1305
NEWPAG 82 319 608 707 710
NEXTRA 83 905 918 1118 1130
NHOLES 84 1129 1162 1174 1176
NO 27 247 585 1238
NOEJECT 12
NOT0 741 745
NOTLC 120 124
NXLOOP 1016 1019
NXTCHR 969 1005
NXTENT 981 984 1015
OUTBUF 62 140 925 932 1139
OUTP 85 137 148 603 872 892 920 922 926 931 1133
OUTW 86 149 604 882 902 936 939
OUTWDS 87 150 605 915 940 942 1122
PAGE 107 107 300
PAGELEN 28 273 617
PAGENUM 29 852
PAGEWIDTH 30 294 615
PFOOT 685 782 1111
PHEAD 702 742 750 1096
PLVAL 88 275 277 618
PUT 146 737 1222 1233 1240
PUTDEC 789 854
PUTLIN 763 836 841
PUTTL 693 720 848 855 858
PUTWRD 864 1245
RELMINUS 1047 1065
RELPLUS 1044 1061
RMVAL 89 166 220 236 295 297 616 876
SCOPY 498 929 951 957 1290
SEARCH 189 968 1024 1027
SET 208 223 240 276 296 317 329 340 351 1041
SIGNEQ 373 376
SKIP 108 108 322 686 696 717 723 772 1073 1079 1103
SKPHDR 715 721
SP1 1159 1161
SP2 1166 1168
SP3 1179 1186
SP4 1180 1187
SPACE 56 109 109 306 331 334 1085
SPLOOP 1145 1152 1189
SPREAD 908 1117
SPVAL 90 328 330
TAB 31 490 514 559 570 1275
TENTAB 791 826
TEXT 48 1195
TITEST 752 758
TIVAL 91 162 171 222 224 242 289 598 644 751 761 878
TIZERO 754 760
TLCOPY 853 857
TRIPLE 32 61 62 63 888 1137 1207
ULINED 1266 1268 1285
ULVAL 92 350 352 600 1204 1212 1214
UNDER 110 110 344
UNDERL 1211 1255
UNDERLINING 1199 1201
UNFILLED 1232 1236
UNLOOP 1264 1274 1276 1278 1283
VALOOP 525 540 542
WAITUP 93 586 704
WIDNXT 1299 1306 1308 1311
WIDTH 161 866 1298
WRDBUF 63 1208
WRDEND 567 569 571 574
YES 33 253 594 705
```

end

CHANGING BASES

BY THEODORE C. HINES, ROSANN COLLINS, JERRY RUSSELL, LINDA SPENCER

Library Science/Educational Technology Division
School of Education
University of North Carolina at Greensboro
Greensboro, NC 27412

Recently, we needed a decimal-to-hex conversion routine in BASIC. We had seen many programs in the journals but, unfortunately, they would not work for us because our Processor Technology Extended Cassette BASIC does not have string arrays (though we fake them with programmer defined functions—is that of interest to anyone?).

Anyhow, here is a programmer-defined function in Processor Technology BASIC which does essentially the same thing, using a dibble-dabble algorithm, which we felt might be useful to others using SOL BASIC. Obviously, it returns a string, not a numeric value.

To this, we added a similar function which reverses the procedure, providing hex (or binary or octal) conversion to decimal.

The algorithms are well-known, of course, but we felt that having the code might be a time-saver—certainly published programs have been most helpful to us, including those in versions of BASIC not locally available.

These functions can be readily converted to subroutines—we just like the convenience of multi-line functions, which should be available in more versions of BASIC.

LISTING

```
 10 REM-                  NUMBER CONVERTER
 20 REM-
 30 REM-     This program demonstrates two utility functions
 40 REM- in Processor Technology Extended Cassette BASIC
 50 REM- which convert from any base from 2 to 16 to decimal,
 60 REM- or from decimal to any base 2 to 16.  The PRINT
 70 REM- "&K" instruction clears the video screen.  The
 80 REM- functions may be easily converted to subroutines if
 90 REM- your basic does not permit multiple line functions.
100 REM-
110 REM-
120 REM-    Dr. Theodore C. Hines
130 REM-    Director, Children's Media Data Bank
140 REM-    Library Science/Educational Technology Division
150 REM-    School of Education
160 REM-    University of North Carolina at Greensboro
170 REM-    Greensboro, North Carolina 27412
180 REM-    (919)  379-5710
190 REM-
200 REM-
210 DIM A$(30)
220 PRINT "&K"
230 PRINT "          NUMBER CONVERTER"
240 PRINT : PRINT : PRINT : PRINT
250 INPUT "Base of your number (2 TO 16)? ",B
260 INPUT "Your number? ",A$
270 LET Z=FNX2(B,A$)
280 PRINT "Your number in base 10 is: ";Z
290 LET Y$=FNA1$(B,Z)
300 PRINT "Your number reconverted to the orisinal base is :";Y$
310 PRINT
320 GOTO 250
330 DEF FNX2(B,A$)
340 REM- This function converts numbers to decimal
350 REM- from any other base from 2 to 16.
360 LET N=0
370 LET X=LEN(A$)
380 FOR L=1 TO LEN(A$)
390   LET Y$=A$(L,L)
400   SEARCH Y$,"ABCDEF",N1
410   IF N1<>0 THEN  LET Y=N1+9: GOTO 430
420   LET Y=VAL(Y$)
430   LET X=X-1
440   LET Z=INT(B+X+.5)
450   LET N=N+(Y*Z)
460 NEXT L
470 RETURN N
480 FNEND
490 REM- This function converts decimal numbers to other
500 REM- number systems, bases 2 to 16.
510 REM- (919) 379-5710
520 DEF FNA1$(M1,X1)
530 REM- NUMBER SYSTEM CONVERTER, BASE 2 TO 16
540 REM- M1 IS BASE, X1 IS NUMBER
550 LET N1$=""
560 LET Y1=INT(X1/M1)
570 LET Z1=X1-(Y1*M1)
580 FOR I1=1 TO Z1+1
590   READ Z1$
600 NEXT I1
610 LET N1$=Z1$+N1$
620 RESTORE
630 IF X1<M1 THEN  RETURN N1$
640 LET X1=Y1: GOTO 560
650 FNEND
660 DATA "0","1","2","3","4","5","6","7","8","9"
670 DATA "A","B","C","D","E","F"
680 END
```

A Z-80/ZAP DISASM

BY B. W. LEE
1385 Rifle Range Rd.
El Cerrito, CA 94530

As the title says, here is an assembly listing and user documentation for a Z-80 disassembler. It can be used as is under the TDL/NORTH STAR disk version 1.2NS operating system and, with some changes and/or patches, it can be adapted for use under other Z-80 based systems. The latter should be disk based, however, since an important feature is the capability to reconstruct from disassembly a TDL assembler compatible source file in disk (rather than in memory thereby avoiding potential conflicts with the object program).

The logic of the disassembler is closely modeled after an 8080 disassembler by W. Christensen which you published in February 1977. The functions of the front panel switches in controlling disassembly time options have been rewritten to be taken over by two control bytes in RAM.

Under the TDL,NS monitor, the disassembler can be evoked through the "I" user defined command and the symbol table editor through the "O" command. The format for using the "I" command is included in the user documentation. For the "O" command, a message with the number of symbol entries in the table is printed followed by the $ prompt. Entering an entry number (in HEX) will start the editing at that number (00 for the starting entry of the table). For each entry, the editor first prints the existing label followed by address value in HEX, and HALT. The user then can type in a new label if desired or skip to the next label entry with a CR. Editing may be terminated and return to the $ prompt by entering a control C, and a second control C will cause an exit and return to the TDL monitor.

Disassembly procedures given by Christensen are still applicable, except for the use of the control bytes and the differences in the reconstructed assembler code formats.

During a disassembly pass, entering a control C will halt until the entry of any other second control C. The latter will cause exit return to TDL.

I do not have facility to distribute machine readable copies in any form other than North Star diskette. For North Star/Z-80 users, I can supply a copy of source, documentation, and relocatable object in a diskette for $6.00 plus the cost of a diskette.

LISTING

```
;
;
;
;
;
;       ************************
;       *                      *
;       *    Z-80/ZAP DISASM   *
;       *                      *
;       *         BY           *
;       *   B.W. LEE CC1978    *
;       *  1385 Rifle Range Rd *
;       *  ElCerrito,cal,94530 *
;       *                      *
;       ************************
;
;
.RADIX  16
;
;
;This program is an Z-80 extension of an 8080
;disassembler by W.Christensen as published
;in DDJCCO,Februry,1977.
;It is intended for operation under the TDL/NS
;disk system monitor (ZAPPLE),but it can be
;easily adopted for other operating enviroment.
;
;The program will disassemble any Z-80 or 8080
;object program and reconstruct a TDL assembly
;format compatible source file in disk.The ob-
;ject program resides in memory starting at lo-
;cation BBBB and ending at location EEEE. Dis-
;assembly may be effected in multi-segment
;passes,with each segment starting at any loc-
;ation and ending at any location with in the
;range BBBB-EEEE. The latter may or may not be
;the execution locations of the object program.
;
;For each segment/pass,run-time options are
;controlled by the values of two control bytes
;-the CBYTE and the KBYTE.
;
;The options specified the CBYTE are as follow:
;
; BIT           ACTION IF BIT IS TRUE
;
;  0    Use the symbol table for the label
;       field and the address field (see bit 6)
;
;  1    Do not initialize the symbol table
;       nor the disk source file (see KBYTE
;       for exception). Use this bit to append
;       entries to the symbol table and new
;       lines to the disk file.
;
;  2    Build symbol table using the Lxxxx
;       format for extended reference instr.
;       (see bits 1&6 for exceptions)
;
;  3    Generate labels Lxxxx instead of
;       the default HEX values xxxxH.
;
;  4    Generate ASCII characters in quotes
;       depending on BIT 5. If BIT 5 is true,
;       all printable ASCII characters will
;       be placed in quotes following the
;       .BYTE pseudo opcode. If BIT 5 is
;       false,all two operand instr with
;       data will be followed as a comment
;       by the quoted ASCII character cor-
;       resonding the data.
;
;  5    Generate .BYTE pseudo op code
;       instead of default assembly code.
;       Use this bit for proper disassembly
;       of segments consisting of tables
;       and ASCII strings.
;
;  6    Normally the operands of LXI instr
;       are treated as 2 byte data rather
;       than as an addresses. Use this bit
;       to over ride this default option.
;       If this bit is true,the 2 byte op-
;       erand of LXI instructions will be
```

```
                        ;       used to build the symbol table and
                        ;       if already there, the symnol en-
                        ;       tries will be used in place of the
                        ;       default HEX format.
                        ;
                        ;  7    Write/build a reconstructed SOURCE
                        ;       FILE in disk and in a compatible
                        ;       format ready for re-assembly by the
                        ;       TDL MACRO RELOCATING ASSEMBLER.
                        ;
                        ;In addition,the KBYTE also control the run
                        ;time disassembly operations. The actions
                        ;specified by the KBYTE are as follows:
                        ;
                        ;BIT             ACTION IF TRUE
                        ;
                        ;  0    Restart the disk file regardless of
                        ;       bit 1 of the CBYTE. Use this bit to
                        ;       over ride the APPEND mode specified
                        ;       by the CBYTE without affect on the
                        ;       specified action on the symbol
                        ;       table.
                        ;
                        ;  1    Close the disk file with an END
                        ;       OF FILE MARK (↑Z) at the normal
                        ;       termination of this segment/pass.
                        ;
                        ; 2-6   Reserved for future use.
                        ;
                        ;  7    Load the specifed new value for
                        ;       CBYTE. Use this to set a new value
                        ;       for CBYTE. Alternately,a false value
                        ;       for this bit will cause the program
                        ;       to continue with the CBYTE previously
                        ;       set regardless of the current value
                        ;       entered.
                        ;
                        ;PARAMETERS for the program are deposited in
                        ;memory locations prior to entering the prog-
                        ;ram. For user of TDL's ZAPPLE/NS disk monitor
                        ;a short routine is included which patch the
                        ;disassembler to one of the un-used ZAPPLE
                        ;commands (i.e. the I command). In this case
                        ;the command for evolking the disassembler is:
                        ;
                        ;       >IBBBB,EEEE,KKCC(cr)
                        ;
                        ;       where   BBBB-is the address for begin-
                        ;                    ning disassembly.
                        ;               EEEE-is the address terminat-
                        ;                    ing disassembly.
                        ;               KK  -is the KBYTE.
                        ;               CC  -is the CBYTE.
                        ;
                        ;
                        ;       EQUATES:
BE87                    BBUF    =       ZMON+0E87 ;ADDR OF OBJ BEG ADDR
BE89                    EBUF    =       BBUF+2  ;ADDR OF OBJ END ADDR
BE8B                    KBYTE   =       EBUF+2
BE8C                    CBYTE   =       KBYTE+1
9C00                    SYMTB   =       NSDOS-0800 ;SYMBOL TBL BEG ADDR
BE8D                    SYMBC   =       CBYTE+1 ;SYMB TBL ENTRY COUNT
BE8E                    TYFLG   =       SYMBC+1
BE90                    CC1     =       TYFLG+2
BE93                    CCB     =       CC1+3
BE94                    TABC    =       CCB+1   ;TAB COUNT
B000                    ZMON    =       0B000   ;ZAPPLE/NS MONITOR
A400                    NSDOS   =       0A400   ;NORTH STAR DOS
B745                    ZCI     =       ZMON+0745
B009                    ZCO     =       ZMON+9
B00C                    ZPO     =       ZMON+0C
B4C2                    PEOL    =       ZMON+04C2
B53D                    EXPR3   =       ZMON+053D
B9C0                    REMAN   =       ZMON+09C0
B024                    CLOSE   =       ZMON+0024
B592                    LADR    =       ZMON+0592
B455                    TOM1    =       ZMON+0455
AD09                    NSCTC   =       NSDOS+0909 ;NORTH STAR CONTC
000D                    CR      =       0D
000A                    LF      =       0A
00FF                    TRUE    =       0FF
0000                    FALSE   =       00
0009                    TAB     =       'I'-40
                        ;
                        ;
                        ;       I/O JUMP TABLE
0000'  C3 001B'                 JMP     ENTRY
0003'  C3 B000          ZAPPLE: JMP     ZMON    ;MONITOR
0006'  C3 B745          CI:     JMP     ZCI     ;CONSOLE IN
0009'  C3 B009          CO:     JMP     ZCO     ;CONSOLE OUT
000C'  C3 B00C          PO:     JMP     ZPO     ;PUNCH/DISK OUT
000F'  C3 B024          CDKF:   JMP     CLOSE   ;CLOSE FILE
0012'  C3 B592          PRTAD:  JMP     LADR    ;LIST ADDR
0015'  C3 B9C0          RSDF:   JMP     REMAN   ;OPEN/REST FILE
0018'  C3 AD09          CONTC:  JMP     NSCTC   ;CONTROL C
                        ;
                        ;
                        ;       ENTRY ROUTINE FOR TDL/NS
                        ;
001B'  CD B53D          ENTRY:  CALL    EXPR3
001E'  22 BE87                  SHLD    BBUF
0021'  ED53 BE89                SDED    EBUF
0025'  78                       MOV     A,B
0026'  32 BE8B                  STA     KBYTE
0029'  E680                     ANI     80
002B'  280C                     JRZ     BEGIN
002D'  79                       MOV     A,C
002E'  32 BE8C                  STA     CBYTE   ;ENTRY ROUTINE END.
0031'  21 0564'                 LXI     H,SGNMG
0034'  0622                     MVI     B,SNMGL
0036'  CD B455                  CALL    TOM1
                        ;
                        ;
0039'  3A BE8C          BEGIN:  LDA     CBYTE   ;GET CONTROL BYTE
003C'  E602                     ANI     02      ;APPEND FILES?
003E'  2027                     JRNZ    APNFL   ;YES
0040'  3A BE8C          NEWFL:  LDA     CBYTE
0043'  E604                     ANI     04      ;BUILD SYMBOL TABLE?
0045'  2807                     JRZ     KSFIN   ;NO,NO SYMBOL FILE
0047'  AF                       XRA     A       ;CHEAP ZERO
0048'  32 9C00                  STA     SYMTB   ;INITIALIZE SYMTB
004B'  32 BE8D                  STA     SYMBC   ;INITIALIZE SYMBC
004E'  3A BE8C          KSFIN:  LDA     CBYTE
0051'  E680                     ANI     80      ;CONST. DISK FILE?
0053'  2806                     JRZ     START   ;NO,SO START
0055'  CD 0015'                 CALL    RSDF    ;RESET & OPEN DISK FILE
0058'  CD B4C2                  CALL    PEOL    ;FIRST CRLF TO DISK FILE
005B'  2A BE87          START:  LHLD    BBUF    ;GET STARTING ADDR
005E'  CD 0070'         DISAM:  CALL    LINE    ;DISASSEMBLE ONE LINE
0061'  CD 0526'                 CALL    ENDCK   ;CONTROL C OR END REACHED?
0064'  20F8                     JRNZ    DISAM   ;LOOP TILL STOPPED
0066'  C9                       RET             ;RETURN TO ZMON
0067'  3A BE8B          APNFL:  LDA     KBYTE   ;GET FILE CONTROL BYTE
006A'  E601                     ANI     01      ;DISK FILE REST OVER RIDE?
006C'  20E0                     JRNZ    KSFIN   ;YES,INITIALIZE DISK FILE
006E'  18EB                     JMPR    START   ;NO,SO START
0070'  CD 0012'         LINE:   CALL    PRTAD   ;PRINT OBJ CODE ADDRESS
0073'  3EFC                     MVI     A,0FC   ;ADJ TAB COUNT
0075'  32 BE94                  STA     TABC
0078'  3E09                     MVI     A,TAB   ;GET TAB
007A'  CD 03E7'                 CALL    TYPE    ;PRT AT CONSOLE
007D'  44                       MOV     B,H     ;SET UP FOR
007E'  4D                       MOV     C,L     ;...PRTLB ROUTINE
007F'  3A BE8C                  LDA     CBYTE
0082'  E601                     ANI     01      ;USE SYMBOL TABLE?
0084'  C4 0435'                 CNZ     PRTLB   ;YES,PRT LABEL
0087'  3005                     JRNC    ..A0
0089'  3E3A                     MVI     A,':'
008B'  CD 03D9'                 CALL    ZTYPE
008E'  3E09             ..A0:   MVI     A,TAB   ;GET TAB
0090'  CD 03D9'                 CALL    ZTYPE   ;PRT AT CONSOLE & DISK FILE
0093'  3A BE8C                  LDA     CBYTE
0096'  E620                     ANI     20      ;GEN .BYTE P-OP'S INSTEAD?
0098'  C2 0302'                 JNZ     PRDB    ;YES,PRT .BYTE'S
009B'  E5                       PUSH    H       ;SAVE OBJ. CODE PTR
009C'  66                       MOV     H,M     ;GET OBJ OPCODE
009D'  01 0606'                 LXI     B,OTABM ;SET FOR MAIN OPTABLE
                        ;
                        ;       OPCODE TABLE LOOKUP ROUTINE
                        ;
00A0'  0A               ULP:    LDAX    B       ;GET FIRST TABLE ENTRY
00A1'  B7                       ORA     A       ;END OF TABLE?
00A2'  CA 02E2'                 JZ      SFRET   ;YES,SEARCH FAIL RET.
00A5'  A4                       ANA     H       ;AND OFF REG. BITS
00A6'  6F                       MOV     L,A     ;SAVE REG'LESS OPCODE
00A7'  03                       INX     B       ;PT TO 2ND ENTRY
00A8'  0A                       LDAX    B       ;GET KEY
00A9'  BD                       CMP     L       ;SAME?
00AA'  CA 00B6'                 JZ      FOUND   ;YEA!
00AD'  79                       MOV     A,C     ;BOO! NO MATCH
00AE'  C606                     ADI     06      ;PT TO NEXT BLOCK
00B0'  4F                       MOV     C,A
00B1'  30ED                     JRNC    ULP     ;LOOP TO DO AGAIN
00B3'  04                       INR     B       ;..ADD CARRY OVERFLOW C TO B
00B4'  18EA                     JMPR    ULP     ;LOOP TO DO AGAIN
                        ;
00B6'  03               FOUND:  INX     B       ;PT TO TYPE ENTRY
00B7'  0A                       LDAX    B       ;GET TYPE
00B8'  32 BE8E                  STA     TYFLG   ;SAVE IT
00BB'  D5                       PUSH    D       ;SAVE DE
00BC'  5F                       MOV     E,A     ;MOVE TYPE TO E
00BD'  1600                     MVI     D,00    ;ZERO D
00BF'  21 0586'                 LXI     H,JMTBL ;SET UP FOR JUMP TABLE
00C2'  19                       DAD     D       ;ADD IN
00C3'  19                       DAD     D       ;..2* INSTR TYPE
00C4'  56                       MOV     D,M     ;GET JUMP ADDR-LOW BYTE
00C5'  23                       INX     H       ;PT TO HI BYTE
00C6'  66                       MOV     H,M     ;MOVE HI BYTE TO H
00C7'  6A                       MOV     L,D     ;MOVE LOW BYTE TO L
00C8'  D1                       POP     D       ;RESTORE DE
00C9'  E3                       XTHL            ;PUT JUMP ADDR TO TOP OF STK
00CA'  C9                       RET             ;& RESTORE PTR TO HL THEN GO
                        ;
```

```
                        ;
                        ;       TYPE ROUTINES
                        ;
00CB'  CD 037A'  TYPE0: CALL    PROP    ;PRINT OPCODE,RELOAD OPCODE
00CE'  23        IXHRT: INX     H       ;PT TO NEXT OPCODE
00CF'  C3 03D7'         JMP     RET     ;END OF LINE
                        ;
                        ;
00D2'  CD 037A'  TYPE1: CALL    PROP
00D5'  CD 03B7'  T1B:   CALL    REG     ;PRINT REG .....XXX
00D8'  18F4             JMPR    IXHRT   ;INC H,END OF LINE
                        ;
                        ;
00DA'  CD 037A'  TYPE2: CALL    PROP
00DD'  CD 03B4'         CALL    REGM    ;PRINT REG ..XXX...
00E0'  18EC             JMPR    IXHRT
                        ;
                        ;
00E2'  CD 037A'  TYPE3: CALL    PROP
00E5'  CD 038D'         CALL    XR      ;DO XR ..XX....
00E8'  18E4             JMPR    IXHRT
                        ;
                        ;
00EA'  CD 037A'  TYPE4: CALL    PROP
00ED'  CD 03B4'         CALL    REGM    ;PRINT REG ..XXX...
00F0'  CD 02FD'  T4B:   CALL    COMMA   ;.. A COMMA
00F3'  1803             JMPR    T52     ;..THEN HEX BYTE & EOL
                        ;
                        ;
00F5'  CD 037A'  TYPE5: CALL    PROP
00F8'  23        T52:   INX     H       ;PT TO OPERAND BYTE
00F9'  7E        T53:   MOV     A,M     ;GET IT
00FA'  CD 03CA'         CALL    X00     ;..OUTPUT WITH LEAD ZERO
00FD'  3E48      T54:   MVI     A,'H'   ;H FOR HEX
00FF'  CD 03D9'         CALL    ZTYPE   ;OUTPUT
0102'  7E               MOV     A,M     ;RELOAD BYTE
0103'  23               INX     H       ;PT TO NEXT OBJ CODE
0104'  C5               PUSH    B       ;SAVE B
0105'  47               MOV     B,A     ;SAVE A
0106'  3A BE8C          LDA     CBYTE   ;GET CONTROL BYTE
0109'  E610             ANI     10      ;CBYTE SAYS
010B'  78               MOV     A,B     ;..PRT ASCII AS COMMENT
010C'  C1               POP     B       ;..IF:
010D'  CA 03D7'         JZ      RET     ;...EOL, NO ASCII
0110'  FE20             CPI     ' '
0112'  DA 03D7'         JC      RET     ;...UNPRINTABLE
0115'  FE5B             CPI     'Z'+1   ;GTR THAN UPPER CASE
0117'  D2 03D7'         JNC     RET     ;UNPRINTABLE
011A'  3A BE8E          LDA     TYFLG   ;GET TYPE FLAG
011D'  FE04             CPI     04
011F'  DA 03D7'         JC      RET     ;NO ASCII FOR TYPES 0-3
0122'  FE06             CPI     06
0124'  D2 03D7'         JNC     RET     ;NOR FOR TYPE 6-1E
0127'  3E09             MVI     A,TAB   ;GET TAB
0129'  CD 03D9'         CALL    ZTYPE   ;OUTPUT TAB
012C'  3E3B             MVI     A,';'   ;GET COMMENT SYMBOL
012E'  CD 03D9'         CALL    ZTYPE   ;OUTPUT IT
0131'  2B               DCX     H       ;BACKUP FOR CHAR
0132'  7E               MOV     A,M     ;GET CHAR
0133'  23               INX     H       ;RESTORE H
0134'  F5       QCHAR: PUSH    PSW
0135'  CD 036B'         CALL    OQ      ;OUTPUT QUOTED CHAR
0138'  F1               POP     PSW     ;RESTORE AF
0139'  3E27             MVI     A,27
013B'  CD 03D9'         CALL    ZTYPE   ;CLOSING QUOTE
013E'  C3 03D7'         JMP     RET     ;END OF LINE
                        ;
                        ;
0141'  CD 037A'  TYPE6: CALL    PROP
0144'  CD 038D'         CALL    XR
0147'  CD 02FD'  T6B:   CALL    COMMA
014A'  3A BE8C          LDA     CBYTE
014D'  E640             ANI     40      ;USE SYMBOL TABLE?
014F'  2806             JRZ     TYP6H   ;NO,PRT DEFAULT HEX
0151'  CD 0418'  PRT2X: CALL    SYMBO   ;PRT FROM TABLE
0154'  DA 00CE'         JC      IXHRT   ;INX H & EOL
0157'  23        TYP6H: INX     H
0158'  46               MOV     B,M     ;SAVE FIRST BYTE
0159'  23               INX     H
015A'  7E               MOV     A,M     ;GET 2ND BYTE
015B'  CD 03CA'         CALL    X00     ;OUTPUT IT FIRST
015E'  78               MOV     A,B     ;GET FIRST BYTE
015F'  CD 035D'         CALL    XO      ;..THEN OUTPUT IT
0162'  3A BE8C          LDA     CBYTE
0165'  E608             ANI     08      ;LABEL OR HEX
0167'  2894             JRZ     T54     ;..HEX
0169'  C3 00CE'         JMP     IXHRT   ;INX H & EOL
                        ;
016C'  CD 037A'  TYPE7: CALL    PROP
016F'  CD 0418'         CALL    SYMBO   ;PRT FROM TABLE IF BIT0=TRUE
0172'  DA 03D7'         JC      RET
0175'  3A BE8C          LDA     CBYTE
0178'  E608             ANI     08      ;PRT LABEL?
017A'  28DB             JRZ     TYP6H   ;NO,SO PRT HEX
017C'  3E4C             MVI     A,'L'   ;LABEL PREFIX
017E'  CD 03D9'         CALL    ZTYPE
0181'  18D4             JMPR    TYP6H   ;..THEN ADDR PART OF LABEL
                        ;
0183'  03        TYPE8: INX     B       ;PT TO OPCODE ENTRY
0184'  0A               LDAX    B       ;GET 'J' OR 'C'
0185'  32 BE90           STA     CC1     ;STORE AS 1ST CHAR OF OPCODE
0188'  CD 032E'         CALL    GETCC   ;GET CONDITION CODE
018B'  18DF             JMPR    TYPE7   ;..THEN SAME AS TYPE7
                        ;
018D'  3E52      TYPE9: MVI     A,'R'
018F'  32 BE90           STA     CC1     ;STORE AS 1ST CCHAR
0192'  CD 032E'         CALL    GETCC   ;GET COND CODE
0195'  C3 00CB'         JMP     TYPE0   ;..THEN SAME AS TYPE 0
                        ;
0198'  CD 037A'  TYPEA: CALL    PROP
019B'  1F               RAR
019C'  1F               RAR
019D'  1F               RAR
019E'  E607             ANI     07      ;ISOLATE RST #
01A0'  C630             ADI     '0'     ;MAKE PRINTABLE
01A2'  CD 03D9'         CALL    ZTYPE
01A5'  C3 00CE'         JMP     IXHRT   ;..EOL
                        ;
01A8'  CD 037A'  TYPEB: CALL    PROP
01AB'  CD 03B4'         CALL    REGM    ;PRT REG ..XXX...
01AE'  CD 02FD'         CALL    COMMA   ;..THEN COMMA
01B1'  7E               MOV     A,M     ;RELOAD OPCODE
01B2'  C3 00D5'         JMP     T1B     ;..THEN PRT REG .....XXX
                        ;
00F5'           TYPEC   =       TYPE5
                        ;
01B5'  03        TYPED: INX     B       ;PT TO OPCODE
01B6'  0A               LDAX    B       ;GET 'J'
01B7'  32 BE8F           STA     CC1-1   ;STORE AS 1ST CHAR
01BA'  03               INX     B       ;PT TO 2ND CHAR
01BB'  0A               LDAX    B       ;GET 'R'
01BC'  32 BE90           STA     CC1     ;STORE AS 2ND CHAR
01BF'  7E               MOV     A,M     ;RELOAD OPCODE
01C0'  E6DF             ANI     0DF     ;MASK OFF BIT 5 TO ADJUST
01C2'  CD 032F'         CALL    GETC    ;GET CONDITION CODE
01C5'  0B               DCX     B       ;BACKUP B PTR (TO TYFLG)
01C6'  C3 00F5'         JMP     TYPE5   ;..THEN SAME AS TYPE 5
                        ;
01C9'  23        TYPEE: INX     H       ;PT TO 2ND OBJ CODE
01CA'  E5               PUSH    H       ;SAVE IT
01CB'  66               MOV     H,M     ;MOVE 2ND OPCODE TO H
01CC'  01 07D5'         LXI     B,OTABC ;SETUP FOR CB-OPTABLE
01CF'  3E21      ULP1:  MVI     A,21    ; MARKER FOR SIMPLE 2ND PASS
01D1'  32 BE8E   ULP2:  STA     TYFLG   ;PRESET TYFLG AS PASS FLG
01D4'  C3 00A0'         JMP     ULP     ;GO DO TABLE LOOKUP AGAIN
                        ;
01D7'  23        TYPEF: INX     H       ;PT TO 2ND OBJ CODE
01D8'  E5               PUSH    H       ;SAVE IT
01D9'  66               MOV     H,M     ;MOVE 2ND OBJ CODE TO H
01DA'  01 07F2'         LXI     B,OTABE ;SETUP FOR ED-OPTABLE
01DD'  18F0             JMPR    ULP1    ;MARK & DO SEARCH AGAIN
                        ;
01DF'  03        TYP10: INX     B       ;PT TO IREG ID
01E0'  C5               PUSH    B       ;SAVE IT FOR LATER
01E1'  23               INX     H       ;PT TO 2ND OBJ CODE
01E2'  E5               PUSH    H       ;SAVE IT FOR LATER ALSO
01E3'  66               MOV     H,M     ;GET 2ND OBJ CODE
01E4'  3ECB             MVI     A,0CB   ;GET TEST BYTE CB
01E6'  BC               CMP     H       ;SAME?
01E7'  CA 01F1'         JZ      TY10C   ;YES, INDEXED CB GROUP
01EA'  01 088D'         LXI     B,OTBIR ;SETUP FOR IREG OPTABLE
01ED'  3E41             MVI     A,41    ;MARKER FOR IREG 2ND PASS
01EF'  18E0             JMPR    ULP2    ;MARK & DO SEARCH AGAIN
                        ;
01F1'  E1        TY10C: POP     H       ;REMOVE OBJ PTR FROM STACK
01F2'  23               INX     H       ;ADVANCE PTR TO
01F3'  23               INX     H       ;..4TH BYTE=3RD OBJ OPCODE
01F4'  E5               PUSH    H       ;  ..SAVE IT
01F5'  66               MOV     H,M     ;    ..GET 3RD OBJ OPCODE
01F6'  01 08E9'         LXI     B,OTBIC ;SETUP FOR IREG CB-OPTABLE
01F9'  3E43             MVI     A,43    ;MARKER FOR IREG 3 OPC PASS
01FB'  18D4             JMPR    ULP2    ;MARK & DO SEARCH AGAIN
                        ;
01FD'  CD 049C'  TYP11: CALL    GRSCC   ;GET ROTATE & SHIFT CODE
0200'  C3 00D2'         JMP     TYPE1   ;..THEN SAME AS TYPE 1
                        ;
0203'  CD 037A'  TYP12: CALL    PROP
0206'  CD 04BF'         CALL    POTLM   ;PRT OCTAL DATA ..DDD...
0209'  CD 02FD'         CALL    COMMA
020C'  7E               MOV     A,M     ;RELOAD OPCODE
020D'  C3 00D5'         JMP     T1B     ;PRT REG .....XXX
                        ;
0210'  03        TYP13: INX     B       ;PT TO OPCODE ENTRY
0211'  0A               LDAX    B       ;..GET IT
0212'  32 BE90           STA     CC1     ;STORE AS 1ST CHAR
0215'  03               INX     B       ;SKIP NEXT
0216'  03               INX     B       ;..TWO CHAR
0217'  03               INX     B       ;PT TO LAST OPCODE ENTRY
0218'  0A               LDAX    B       ;..GET IT
0219'  32 BE93           STA     CCB     ;STORE AS LAST CHAR
021C'  CD 04C9'         CALL    GETXR   ;GET XR CODE
021F'  C3 016C'         JMP     TYPE7   ;..THEN SAME AS TYPE 7
                        ;
0222'  03        TYP14: INX     B       ;PT TO OPCODE ENTRY
```

```
0223'  0A              LDAX   B       ;..GET IT
0224'  32 BE90         STA    CC1     ;STORE AS 1ST CHAR
0227'  03              INX    B       ;PT TO NEXT ENTRY
0228'  0A              LDAX   B       ;..GET IT
0229'  32 BE91         STA    CC1+1   ;STORE AS 2ND CHAR
022C'  CD 04F3'        CALL   GETBX   ;GET BX CODE
022F'  C3 00CB'        JMP    TYPE0   ;..THEN SAME AS TYPE 0
                ;
0232'  3E4F     TYP15: MVI    A,'O';GET A 'O' PREFIX
0234'  CD 03D9'        CALL   ZTYPE   ;OUTPUT IT
0237'  18E9            JMPR   TYP14   ;..THEN SAME AS TYPE 14
                ;
0239'  03       TYP16: INX    B       ;PT TO OPCODE ENTRY
023A'  0A              LDAX   B
023B'  32 BE90         STA    CC1     ;STORE AS 1ST CHAR
023E'  03              INX    B
023F'  0A              LDAX   B
0240'  32 BE91         STA    CC1+1
0243'  C1              POP    B       ;RESTORE IREG ID
0244'  3E49            MVI    A,'I'   ;GET A 'I' PREFIX
0246'  32 BE92         STA    CC1+2
0249'  03              INX    B       ;SKIP PARENTHESIS
024A'  0A              LDAX   B       ;GET IREG ID X OR Y
024B'  32 BE93         STA    CCB
024E'  01 BE8F         LXI    B,CC1-1 ;SET B PTR FOR OUTPUT
0251'  C3 00CB'        JMP    TYPE0   ;..THEN SAME AS TYPE 0
                ;
0254'  CD 037A' TYP17: CALL   PROP
0257'  C1              POP    B       ;RESTORE IREG ID PTR
0258'  CD 050D'        CALL   PRIR    ;PRT IREG
025B'  C3 00CE'        JMP    IXHRT   ;INX H & EOL
                ;
025E'  CD 037A' TYP18: CALL   PROP
0261'  23              INX    H       ;PT TO OFFSET BYTE
0262'  7E              MOV    A,M     ;GET OFFSET
0263'  CD 03CA'        CALL   XOO     ;PRT IT WITH LEAD ZERO
0266'  C1              POP    B       ;RESTORE IREG ID PTR
0267'  CD 0513'        CALL   PRIRP   ;PRT IREG WITH PAREN
026A'  3E04            MVI    A,04    ;ADJUST TYFLG
026C'  32 BE8E         STA    TYFLG   ;..TO TYPE 4
026F'  C3 00F0'        JMP    T4B     ;..THEN LAST OF TYPE 4
                ;
0272'  CD 037A' TYP19: CALL   PROP
0275'  C1              POP    B       ;RESTORE IREG ID PTR
0276'  CD 050D'        CALL   PRIR    ;PRT IREG
0279'  C3 0147'        JMP    T6B     ;..DO LAST OF TYPE 6
                ;
027C'  03       TYP1A: INX    B       ;PT TO OPCODE ENTRY
027D'  0A              LDAX   B
027E'  32 BE90         STA    CC1
0281'  03              INX    B       ;SKIP NEXT
0282'  03              INX    B       ;..TWO
0283'  03              INX    B       ;PT TO LAST
0284'  0A              LDAX   B       ;..GET IT
0285'  32 BE93         STA    CCB     ;STORE AS 4TH CHAR
0288'  C1              POP    B       ;RESTORE IREG ID PTR
0289'  3E49            MVI    A,'I'
028B'  32 BE91         STA    CC1+1
028E'  03              INX    B       ;SKIP PAREN
028F'  0A              LDAX   B       ;GET IREG CODE,X OR Y
0290'  32 BE92         STA    CC1+2
0293'  01 BE8F         LXI    B,CC1-1 ;SET B PTR FOR OUTPUT
0296'  C3 016C'        JMP    TYPE7   ;..THEN DO TYPE 7
                ;
0299'  CD 037A' TYP1B: CALL   PROP
029C'  23              INX    H       ;PT TO OFFSET BYTE
029D'  7E              MOV    A,M     ;GET OFFSET
029E'  CD 03CA'        CALL   XOO     ;PRT WITH LEAD ZERO
02A1'  C1              POP    B       ;RESTORE IREG ID PTR
02A2'  CD 0513'        CALL   PRIRP   ;PRT IREG WITH PAREN
02A5'  2B              DCX    H       ;BACKUP FOR REG CODE
02A6'  CD 02FD'        CALL   COMMA
02A9'  7E              MOV    A,M     ;GET IREG CODE
02AA'  CD 03B7'        CALL   REG     ;PRT REG .....XXX
02AD'  23       IXHHR: INX    H       ;RESTORE H PTR
02AE'  C3 00CE'        JMP    IXHRT   ;INX H & EOL
                ;
02B1'  CD 037A' TYP1C: CALL   PROP
02B4'  CD 03B4'        CALL   REGM    ;PRT REG ..XXX...
02B7'  CD 02FD'        CALL   COMMA
02BA'  23              INX    H       ;PT TO OFFSET BYTE
02BB'  7E              MOV    A,M     ;GET OFFSET
02BC'  CD 03CA'        CALL   XOO     ;PRT WITH LEAD ZERO
02BF'  C1              POP    B       ;RESTORE IREG ID PTR
02C0'  CD 0513'        CALL   PRIRP   ;PRT IREG WITH PAREN
02C3'  C3 00CE'        JMP    IXHRT   ;EOL
                ;
02C6'  CD 037A' TYP1D: CALL   PROP
02C9'  CD 04BF'        CALL   POTLM   ;PRT OCTAL ..XXX...
02CC'  CD 02FD'        CALL   COMMA
02CF'  2B       TY1DB: DCX    H       ;BACKUP FOR OFFSET
02D0'  7E              MOV    A,M     ;GET OFFSET
02D1'  CD 03CA'        CALL   XOO
02D4'  C1              POP    B
02D5'  CD 0513'        CALL   PRIRP
02D8'  18D3            JMPR   IXHHR   ;ADJ PTR INX H THEN EOL
                ;
02DA'  CD 049C' TYP1E: CALL   GRSCC   ;GET R & S CODE
02DD'  CD 037A'        CALL   PROP
02E0'  18ED            JMPR   TY1DB   ;..THEN DO LAST OF TYPE 1D
                ;
02E2'  3A BE8E  SFRET: LDA    TYFLG   ;RECOVER PASS MARKER
02E5'  FE20            CPI    20      ;ONE BYTE OPCODE PASS
02E7'  E1              POP    H       ;RESTORE OBJ CODE PTR
02E8'  DA 0302'        JC     PRDB    ;YES,PRT .BYTE P-OP
02EB'  FE40            CPI    40      ;SIMPLE MULTIPASS OPCODE
02ED'  3801            JRC    BKUPH   ;YES, BACKUP TO 1ST BYTE
02EF'  C1              POP    B       ;NO, REMOVE IREG ID PTR
02F0'  E60F     BKUPH: ANI    0F      ;STRIP OFF MARKER
02F2'  47              MOV    B,A     ;MOVE DISPLACEMENT TO B
02F3'  AF              XRA    A       ;RESET TYFLG
02F4'  32 BE8E         STA    TYFLG   ;..TO NORMAL
02F7'  2B       BUHLP: DCX    H       ;BACKUP
02F8'  10FD            DJNZ   BUHLP   ;..TILL B=0
02FA'  C3 0302'        JMP    PRDB    ;...THEN PRT .BYTE'S
                ;
                ;       ****************************
                ;       * DISASSEMBLER SUBROUTINES  *
                ;       ****************************
                ;
                ;       PRINT COMMA
02FD'  3E2C     COMMA: MVI    A,','
02FF'  C3 03D9'        JMP    ZTYPE
                ;
                ;       PRINT .BYTE'S
0302'  E5       PRDB:  PUSH   H
0303'  21 055F'        LXI    H,BYTSG ;PT TO '.BYTE' STRG
0306'  0605            MVI    B,BYSGL
0308'  7E       PRDB1: MOV    A,M
0309'  CD 03D9'        CALL   ZTYPE
030C'  23              INX    H
030D'  10F9            DJNZ   PRDB1
030F'  3E09            MVI    A,TAB
0311'  CD 03D9'        CALL   ZTYPE
0314'  E1              POP    H
0315'  3A BE8C         LDA    CBYTE
0318'  E630            ANI    30      ;CTRL BITS 4&5
031A'  FE30            CPI    30      ;BOTH TRUE?
031C'  C2 00F9'        JNZ    T53     ;NO,PRT HEX
031F'  7E              MOV    A,M
0320'  FE20            CPI    ' '
0322'  DA 00F9'        JC     T53     ;UNPRINTABLE
0325'  FE5B            CPI    'Z'+1
0327'  D2 00F9'        JNC    T53     ;UNPRINTABLE
032A'  23              INX    H       ;PT TO NEXT OBJ CODE
032B'  C3 0134'        JMP    QCHAR   ;PRT QUOTED CHAR & EOL
                ;
                ;       SETUP CONDITION CODE FOR OUTPUT
032E'  7E       GETCC: MOV    A,M     ;RELOAD COND CODE
032F'  1F       GETC:  RAR
0330'  1F              RAR
0331'  E60E            ANI    0E      ;ISOLATE COND CODE
0333'  E5              PUSH   H       ;SAVE OBJ CODE PTR
0334'  21 05C6'        LXI    H,CCTAB ;PT TO CC-TABLE
0337'  CD 03C5'        CALL   ADD     ;ADD A TO H
033A'  7E              MOV    A,M     ;GET 1ST CC CHAR
033B'  32 BE91         STA    CC1+1
033E'  23              INX    H
033F'  7E              MOV    A,M
0340'  32 BE92         STA    CC1+2
0343'  E1              POP    H
0344'  01 BE8F         LXI    B,CC1-1 ;SET B PTR FOR OUTPUT
0347'  3E20            MVI    A,' '   ;GET A SPACE
0349'  32 BE93         STA    CCB
034C'  C9              RET
                ;
                ;       HEX TO ASCII CONVERSION
034D'  1F       HEXL:  RAR     ;GET LEFT DIGIT
034E'  1F              RAR
034F'  1F              RAR
0350'  1F              RAR
0351'  E60F     HEXR:  ANI    0F      ;GET RIGHT DIGIT
0353'  FE0A            CPI    0A
0355'  DA 035A'        JC     HEXRN
0358'  C607            ADI    07
035A'  C630     HEXRN: ADI    '0'     ;MAKE PRINTABLE
035C'  C9              RET
                ;
                ;       HEX OUTPUT ROUTINE
035D'  F5       XO:    PUSH   PSW
035E'  CD 034D'        CALL   HEXL    ;GET LEFT NIBBLE
0361'  CD 03D9'        CALL   ZTYPE   ;OUTPUT IT
0364'  F1              POP    PSW
0365'  CD 0351'        CALL   HEXR    ;GET RIGHT NIBBLE
0368'  C3 03D9'        JMP    ZTYPE   ;OUTPUT THEN RET
                ;
                ;       OUTPUT SPACE,QUOTE,CHAR IN A,RET
036B'  F5       OQ:    PUSH   PSW
036C'  3E20            MVI    A,' '
036E'  CD 03D9'        CALL   ZTYPE
0371'  3E27            MVI    A,27
0373'  CD 03D9'        CALL   ZTYPE
0376'  F1              POP    PSW
0377'  C3 03D9'        JMP    ZTYPE
```

```
                        ;
                        ;       PRINT 4 BYTE OPCODE FROM TABLE
037A'   D5              PROP:   PUSH    D
037B'   1604                    MVI     D,4     ;OPCODE LENGTH
037D'   03              PROPL:  INX     B       ;INC OPCODE PTR
037E'   0A                      LDAX    B       ;GET OPCODE CHAR
037F'   CD 03D9'                CALL    ZTYPE   ;OUTPUT IT
0382'   15                      DCR     D       ;DONE?
0383'   20F8                    JRNZ    PROPL   ;..NO
0385'   3E09                    MVI     A,TAB   ;GET A TAB
0387'   CD 03D9'                CALL    ZTYPE   ;OUTPUT IT
038A'   D1                      POP     D
038B'   7E                      MOV     A,M     ;RELOAD OPCODE
038C'   C9                      RET
                        ;
                        ;       PRINT XR IN BITS ..XXX...
038D'   1F              XR:     RAR
038E'   1F                      RAR             ;MOVE ..XXX...
038F'   1F                      RAR             ;..TO .....XXX
0390'   E606                    ANI     06      ;ISOLATE XR VALUE
0392'   FE06                    CPI     06      ;IS IT A 6?
0394'   2021                    JRNZ    REG     ;..NO
0396'   7E                      MOV     A,M     ;RELOAD OPCODE
0397'   B7                      ORA     A       ;PSW OR SP?
0398'   FA 03A5'                JM      PRPSW   ;PRT PSW
039B'   3E53            PRTSP:  MVI     A,'S'   ;PRINT
039D'   CD 03D9'                CALL    ZTYPE
03A0'   3E50                    MVI     A,'P'
03A2'   C3 03D9'                JMP     ZTYPE   ;..'SP' THEN RET
03A5'   3E50            PRPSW:  MVI     A,'P'   ;PRINT
03A7'   CD 03D9'                CALL    ZTYPE
03AA'   3E53                    MVI     A,'S'
03AC'   CD 03D9'                CALL    ZTYPE
03AF'   3E57                    MVI     A,'W'   ;..'PSW'
03B1'   C3 03D9'                JMP     ZTYPE   ;....THEN RET
                        ;
                        ;       PRINT REG IN BITS ..XXX...
03B4'   1F              REGM:   RAR             ;RIGHT
03B5'   1F                      RAR
03B6'   1F                      RAR             ;...ADJUST
                        ;
                        ;       PRINT REG IN BITS .....XXX
03B7'   E5              REG:    PUSH    H
03B8'   21 05D6'                LXI     H,RTAB  ;PT TO REG TABLE
03BB'   E607                    ANI     07      ;ISOLATE REG VALUE
03BD'   CD 03C5'                CALL    ADD     ;ADD A TO HL
03C0'   7E                      MOV     A,M     ;GET ASCII REG
03C1'   E1                      POP     H
03C2'   C3 03D9'                JMP     ZTYPE   ;OUTPUT REG NAME & RET
                        ;
                        ;       ROUTINE TO ADD (A) TO HL
03C5'   85              ADD:    ADD     L       ;ADD IN (A)
03C6'   6F                      MOV     L,A     ;MOVE IT BACK
03C7'   D0                      RNC             ;RET IF NO CRY
03C8'   24                      INR     H       ;ADD CRY TO H
03C9'   C9                      RET
                        ;
                        ;       HEX OUTPUT WITH LEAD ZERO IF REQUIRED
03CA'   FEA0            XOO:    CPI     0A0
03CC'   388F                    JRC     XO
03CE'   F5                      PUSH    PSW
03CF'   3E30                    MVI     A,'0'
03D1'   CD 03D9'                CALL    ZTYPE
03D4'   F1                      POP     PSW
03D5'   1886                    JMPR    XO
                        ;
                        ;       END OF LINE, FALL INTO ZTYPE
03D7'   3E0D            RET:    MVI     A,CR
                        ;
                        ;       OUTPUT ROUTINE, TO CONSOLE & DISK FILE
03D9'   C5              ZTYPE:  PUSH    B
03DA'   47                      MOV     B,A
03DB'   3A BE8C                 LDA     CBYTE
03DE'   17                      RAL             ;BUILD DISK SOURCE FILE?
03DF'   78                      MOV     A,B
03E0'   3004                    JRNC    TYPEP   ;NO, SO TYPE ONLY
03E2'   4F              DFOUT:  MOV     C,A     ;MOVE CHAR INTO C FOR ZAPPLE
03E3'   CD 000C'                CALL    PO      ;OUTPUT TO FILE BUFFER
03E6'   C1              TYPEP:  POP     B       ;RESTORE B
03E7'   C5              TYPE:   PUSH    B       ;SAVE B
03E8'   FE09                    CPI     TAB     ;TAB CHAR?
03EA'   2815                    JRZ     TABX    ;YES
03EC'   4F                      MOV     C,A     ;SET UP AGAIN FOR ZAPPLE
03ED'   CD 0009'                CALL    CO      ;OUTPUT TO CONSOLE
03F0'   47                      MOV     B,A
03F1'   3A BE94                 LDA     TABC    ;GET TAB COUNT
03F4'   3D                      DCR     A
03F5'   32 BE94                 STA     TABC    ;SAVE
03F8'   78                      MOV     A,B
03F9'   C1                      POP     B
                        ;       IF EOL: ADD LF BEFORE RETURN
03FA'   FE0D                    CPI     CR      ;WAS CHAR A CR?
03FC'   C0                      RNZ             ;NO, THEN RETURN
03FD'   3E0A            EOL:    MVI     A,LF    ;GET A LF CHAR
03FF'   18D8                    JMPR    ZTYPE   ;OUTPUT LF CHAR BEFORE EXIT
                        ;
                        ;       TAB EXPANSION FOR CONSOLE OUT
0401'   F5              TABX:   PUSH    PSW
0402'   3A BE94                 LDA     TABC    ;GET TAB COUNT
0405'   47                      MOV     B,A
0406'   0E20            TABXL:  MVI     C,20    ;SET FOR SPACE
0408'   CD 0009'                CALL    CO      ;OUT TO CONSOLE
040B'   78                      MOV     A,B     ;GET LOOP COUNT
040C'   E607                    ANI     07      ;MAKE IT MODULO 8
040E'   47                      MOV     B,A     ;REPLACE LOOP COUNT
040F'   10F5                    DJNZ    TABXL   ;DO TILL B=0
0411'   AF                      XRA     A
0412'   32 BE94                 STA     TABC    ;RESET TAB COUNT
0415'   F1                      POP     PSW
0416'   C1                      POP     B
0417'   C9                      RET
                        ;
                        ;       SYMBO ROUTINE, ROUTINE TO BUILD AND ACCESS
                        ;       THE SYMBOL TABLE
0418'   3A BE8C         SYMBO:  LDA     CBYTE
041B'   E605                    ANI     05      ;EITHER BUILD OR USE REQ'D?
041D'   C8                      RZ              ;NEITHER, SO RET
041E'   E601                    ANI     01      ;USE REQ'D?
0420'   2005                    JRNZ    FRMTB   ;YES, USE TABLE
                        ;
                        ;       BUILD SYMBOL TABLE PASS
0422'   CD 0448'                CALL    SYBLD   ;PUT ENTRY IN TABLE
0425'   B7                      ORA     A       ;ENSURE NO CRY
0426'   C9                      RET             ;...THEN RET
                        ;
                        ;       OUTPUT LABEL BASED ON OPERAND POINTED
                        ;       TO BY HL
0427'   23              FRMTB:  INX     H       ;PT TO OPERAND
0428'   4E                      MOV     C,M     ;GET LOW BYTE
0429'   23                      INX     H
042A'   46                      MOV     B,M     ;GET HIGH BYTE
042B'   23                      INX     H       ;PT TO NEXT INSTRUCTION
042C'   CD 0435'                CALL    PRTLB   ;PRT LABEL IF IN TABLE
042F'   D8                      RC              ;RET IF PRINTED
0430'   2B                      DCX     H       ;BACKUP FOR PROG
0431'   2B                      DCX     H       ;...TO PRINT LABEL
0432'   2B                      DCX     H       ;BECAUSE IT WASN'T IN TABLE
0433'   B7                      ORA     A       ;TURN OFF CRY
0434'   C9                      RET
0435'   CD 047C'        PRTLB:  CALL    LOOKU   ;FIND LABEL
0438'   3F                      CMC             ;CRY OFF IF NOT FOUND
0439'   D0                      RNC             ;RET IF NOT FOUND
                        ;       LABEL FOUND IN TABLE
043A'   0605                    MVI     B,5     ;LABEL LENGTH
043C'   1A              SYPLP:  LDAX    D       ;GET LABEL CHAR
043D'   13                      INX     D       ;PT TO NEXT
043E'   B7                      ORA     A       ;A BLANK?
043F'   2805                    JRZ     SYPEX   ;SHORT LABEL EXIT
0441'   CD 03D9'                CALL    ZTYPE   ;OUTPUT LABEL CHAR
0444'   10F6                    DJNZ    SYPLP   ;DO TILL END OF 5 LABEL CHAR
0446'   37              SYPEX:  STC             ;SET CRY SO PROG WON'T
0447'   C9                      RET             ;..PRT DEFAULT LABEL
                        ;
                        ;       BUILD SYMBOL TABLE
0448'   23              SYBLD:  INX     H       ;PT TO OPERAND
0449'   4E                      MOV     C,M     ;LO BYTE
044A'   23                      INX     H
044B'   46                      MOV     B,M     ;HI BYTE
044C'   2B                      DCX     H       ;RESTORE PTR TO
044D'   2B                      DCX     H       ;...ORG VALUE
044E'   CD 047C'                CALL    LOOKU   ;IN TABLE?
0451'   D0                      RNC             ;YES, THEN RET
                        ;       ADD NEW ENTRY TO SYMBOL TABLE
                        ;       BC CONTAINS ENTRY,DE POINTS TO OPEN'G
                        ;       IN TABLE
0452'   3E4C                    MVI     A,'L'   ;DEFAULT TO LABEL LXXXX
0454'   12                      STAX    D       ;STORE IN TABLE
0455'   13                      INX     D
0456'   78                      MOV     A,B     ;GET HI BYTE
0457'   CD 046F'                CALL    SYSTX   ;STORE PRINTABLE HEX
045A'   79                      MOV     A,C     ;GET LO BYTE
045B'   CD 046F'                CALL    SYSTX
045E'   78                      MOV     A,B     ;GET HI BYTE
045F'   12                      STAX    D       ;STORE HI BYTE
0460'   13                      INX     D
0461'   79                      MOV     A,C     ;LO BYTE
0462'   12                      STAX    D
0463'   AF                      XRA     A       ;GET A ZERO, STORE IT AT
0464'   13                      INX     D       ;..START OF NEXT LABEL
0465'   12                      STAX    D       ;...TO SHOW E.O.TABLE
0466'   3A BE8D                 LDA     SYMBC   ;GET SYMBOL COUNT
0469'   3C                      INR     A       ;ADD 1
046A'   32 BE8D                 STA     SYMBC   ;RESTORE IT
046D'   37                      STC             ;SET CRY, NO DEFAULT LABEL
046E'   C9                      RET
                        ;
                        ;       ROUTINE TO STORE PRINTABLE HEX IN TABLE
046F'   F5              SYSTX:  PUSH    PSW
0470'   CD 034D'                CALL    HEXL    ;CONVERT LEFT NIBBLE
0473'   12                      STAX    D       ;..TO PRINTABLE THEN STORE
0474'   13                      INX     D
0475'   F1                      POP     PSW
0476'   CD 0351'                CALL    HEXR    ;CONVERT RIGHT NIBBLE
0479'   12                      STAX    D       ;..TO PRINTABLE THEN STORE
```

```
047A'  13                INX     D       ;PT TO NEXT
047B'  C9                RET
                ;
                ;       SYMBOL TABLE LOOKUP
                ;       BC HEX SYMBOL TO FIND
                ;       DE POINT TO SYMBOL ON RETURN
                ;       HL RESTORED BEFORE RETURN
                ;       IF NOT FOUND: CRY SET & DE POINTS TO
                ;       OPEN LOCATION OF TABLE
047C'  11 9C00   LOOKU:  LXI     D,SYMTB ;PT TO SYMBOL TABLE
047F'  1A        LULP:   LDAX    D       ;GET PRINTABLE LABEL
0480'  B7                ORA     A       ;BLANK?
0481'  37                STC             ;SET CRY
0482'  C8                RZ              ;RET IF BLANK
                ;       NONBLANK ENTRY
0483'  13                INX     D       ;SKIP
0484'  13                INX     D       ;..ASCII
0485'  13                INX     D       ;...LABEL
0486'  13                INX     D       ;....VALUES
0487'  13                INX     D
0488'  1A                LDAX    D
0489'  B8                CMP     B       ;MATCH?
048A'  13                INX     D
048B'  200C              JRNZ    LUNO    ;NO MATCH
                ;       HI BYTE MATCHED, CHECK LO
048D'  1A                LDAX    D
048E'  B9                CMP     C
048F'  2008              JRNZ    LUNO    ;NO MATCH
                ;       ENTRY FOUND
0491'  1B                DCX     D       ;BACKUP
0492'  1B                DCX     D       ;..TO
0493'  1B                DCX     D       ;...ASCII
0494'  1B                DCX     D       ;....PART
0495'  1B                DCX     D       ;.....OF
0496'  1B                DCX     D       ;......LABEL
0497'  B7                ORA     A       ;ENSURE NO CRY
0498'  C9                RET             ;NO CRY IMPLY FOUND
                ;       ENTRY DON'T MATCH
0499'  13        LUNO:   INX     D       ;PT TO START OF NEXT
049A'  18E3              JMPR    LULP    ;LOOP BACK
                ;
                ;       GET ROTATE & SHIFT GROUP OPCODES
049C'  7E        GRSCC:  MOV     A,M     ;GET R&S CODE
049D'  1F                RAR
049E'  E61C              ANI     1C      ;ISOLATE
04A0'  E5                PUSH    H
04A1'  21 05DE'          LXI     H,RSTAB ;PT TO RS TABLE
04A4'  CD 03C5'          CALL    ADD
04A7'  7E                MOV     A,M
04A8'  32 BE90           STA     CC1
04AB'  23                INX     H
04AC'  7E                MOV     A,M
04AD'  32 BE91           STA     CC1+1
04B0'  23                INX     H
04B1'  7E                MOV     A,M
04B2'  32 BE92           STA     CC1+2
04B5'  23                INX     H
04B6'  7E                MOV     A,M
04B7'  32 BE93           STA     CCB
04BA'  E1                POP     H       ;RESTORE H PTR
04BB'  01 BE8F           LXI     B,CC1-1 ;SET B PTR FOR OUTPUT
04BE'  C9                RET
                ;
                ;       PRINT OCTAL DATA IN BITS ..XXX...
04BF'  1F        POTLM:  RAR             ;MOVE ..XXX...
04C0'  1F                RAR
04C1'  1F                RAR             ;..TO .....XXX
04C2'  E607      POTL:   ANI     07      ;ISOLATE
04C4'  C630              ADI     30      ;MAKE PRINTABLE
04C6'  C3 03D9'          JMP     ZTYPE   ;OUTPUT IT
                ;
                ;       GET XR IN REG PAIR
04C9'  7E        GETXR:  MOV     A,M     ;GET XR CODE
04CA'  1F                RAR             ;MOVE ..XR....
04CB'  1F                RAR
04CC'  1F                RAR             ;..TO ......XR
04CD'  E606              ANI     06      ;ISOLATE
04CF'  FE06              CPI     06      ;IS IT A 06?
04D1'  200E              JRNZ    GETRP   ;NO, GET RP FROM TABLE
04D3'  3E53              MVI     A,'S'   ;YES, STORE
04D5'  32 BE91           STA     CC1+1
04D8'  3E50              MVI     A,'P'
04DA'  32 BE92           STA     CC1+2   ;..'SP'
04DD'  01 BE8F           LXI     B,CC1-1 ;SET B PTR FOR OUTPUT
04E0'  C9                RET
04E1'  E5        GETRP:  PUSH    H
04E2'  21 05D6'          LXI     H,RTAB  ;PT TO REG TABLE
04E5'  CD 03C5'          CALL    ADD
04E8'  7E                MOV     A,M
04E9'  32 BE91           STA     CC1+1
04EC'  23                INX     H
04ED'  7E                MOV     A,M
04EE'  32 BE92           STA     CC1+2
04F1'  1815              JMPR    GETEX   ;POP H THEN RET
                ;
04F3'  7E        GETBX:  MOV     A,M     ;RELOAD OPCODE
04F4'  1F                RAR             ;MOVE ...XX...
04F5'  1F                RAR             ;.. TO ......XX
04F6'  E606              ANI     06      ;ISOLATE
04F8'  E5                PUSH    H
04F9'  21 05FE'          LXI     H,BXTAB ;PT TO BX TBALE
04FC'  CD 03C5'          CALL    ADD
04FF'  7E                MOV     A,M
0500'  32 BE92           STA     CC1+2
0503'  23                INX     H
0504'  7E                MOV     A,M
0505'  32 BE93           STA     CCB
0508'  E1        GETEX:  POP     H
0509'  01 BE8F           LXI     B,CC1-1 ;SET FOR OUTPUT
050C'  C9                RET
                ;
                ;       PRINT IREG
050D'  03        PRIR:   INX     B
050E'  0A                LDAX    B
050F'  CD 03D9'          CALL    ZTYPE
0512'  C9                RET
                ;
                ;       PRINT IREG WITH PAREN
0513'  3E48      PRIRP:  MVI     A,'H'   ;GET A 'H'
0515'  CD 03D9'          CALL    ZTYPE   ;OUTPUT IT TO COMPLETE
0518'  D5                PUSH    D       ;..OFFSET DATA
0519'  1603              MVI     D,03    ;3 MORE CHAR'S
051B'  0B                DCX     B       ;BACKUP
051C'  03        PIRPL:  INX     B       ;PT TO NEXT CHAR
051D'  0A                LDAX    B
051E'  CD 03D9'          CALL    ZTYPE
0521'  15                DCR     D       ;DONE?
0522'  20F8              JRNZ    PIRPL   ;NO, LOOP BACK
0524'  D1                POP     D
0525'  C9                RET
                ;
                ;       CHECK FOR END OF RANGE OR CONTROL C
0526'  3A BE89   ENDCK:  LDA     EBUF    ;EBUF LO BYTE
0529'  95                SUB     L       ;SUB PTR LO BYTE
052A'  3A BE8A           LDA     EBUF+1  ;EBUF HI BYTE
052D'  9C                SBB     H       ;SUB HI BYTE & CRY
052E'  301E              JRNC    CKCTC   ;NOT END, CHECK FOR CTRL C
0530'  3A BE8B   END:    LDA     KBYTE   ;GET DISK FILE CTRL BYTE
0533'  E602              ANI     02      ;CLOSE DISK FILE?
0535'  2002              JRNZ    ENDSF   ;YES
0537'  AF                XRA     A       ;SET ZERO FOR EXIT
0538'  C9                RET
0539'  E5        ENDSF:  PUSH    H
053A'  AF                XRA     A
053B'  32 BE94           STA     TABC
053E'  21 0558'          LXI     H,ESFSG
0541'  0607              MVI     B,ESFSL
0543'  7E        EDSFL:  MOV     A,M
0544'  CD 03D9'          CALL    ZTYPE
0547'  23                INX     H
0548'  10F9              DJNZ    EDSFL
054A'  E1                POP     H
054B'  C3 000F'          JMP     CDKF
054E'  CD 0018'  CKCTC:  CALL    CONTC   ;CHECK FOR CTRL C
0551'  C0                RNZ             ;NO,THEN RET
0552'  CD 0006'          CALL    CI      ;GET SOMETHING FROM CONSOLE
0555'  FE03              CPI     'C'-40  ;CTRL C AGAIN?
0557'  C9                RET             ;RET WITH ZERO FLG SET
                ;
                ;
0558'  09        ESFSG:  .BYTE   TAB
0559'  2E454E44          .ASCII  '.END'
055D'  0D1A              .BYTE   CR,'Z'-40
0007             ESFSL   =.-ESFSG
055F'  2E42595445 BYTSG:  .ASCII  '.BYTE'
0005             BYSGL   =       .-BYTSG
0564'  5A38302D4449 SGNMG:  .ASCII  'Z80-DISASM V'
0570'  312E302C203C       .ASCII  '1.0, <C>B.W.LEE,1978'
0584'  0D0A              .BYTE   CR,LF
0022             SNMGL   =       .-SGNMG
                ;
                ;
                ;       **********
                ;       * TABLES *
                ;       **********
                ;
                ;       MAIN JUMP TABLE
                ;
                ;                               LEN OPRANDS EXAMPLE
                ;
0586'  00CB'     JMTBL:  .WORD   TYPE0   ;1  NONE     DAA,E1
0588'  00D2'             .WORD   TYPE1   ;1  REG      ADD,ANA
058A'  00DA'             .WORD   TYPE2   ;1  REG      INR,DCR
058C'  00E2'             .WORD   TYPE3   ;1   XR      STAX,LDAX
058E'  00EA'             .WORD   TYPE4   ;2  REG,DATA MVI(ONLY)
0590'  00F5'             .WORD   TYPE5   ;2  DATA     OUT,CPI
0592'  0141'             .WORD   TYPE6   ;3  XR,DATA  LXI(ONLY)
0594'  016C'             .WORD   TYPE7   ;3  ADDR     LHLD,JMP
0596'  0183'             .WORD   TYPE8   ;3  ADDR     JZ,CNC
0598'  018D'             .WORD   TYPE9   ;1  NONE     RZ,RC
059A'  0198'             .WORD   TYPEA   ;1  RST #    RST(ONLY)
059C'  01A8'             .WORD   TYPEB   ;1  REG,REG  MOV(ONLY)
059E'  00F5'             .WORD   TYPEC   ;2  OFS      DJNZ,JMPR
```

```
05A0'  01B5'              .WORD   TYPED   ;2  OFS      JRCC,JRZ
05A2'  01C9'              .WORD   TYPEE   ;2  BIT,RS   BIT, SRL
05A4'  01D7'              .WORD   TYPEF   ;2-4 --      ED PREFIX GP
05A6'  01DF'              .WORD   TYP10   ;2-4 --      DD/FD PREFIX GP
05A8'  01FD'              .WORD   TYP11   ;2  REG      SRLR,RLCR
05AA'  0203'              .WORD   TYP12   ;2  BIT      BIT,RES
05AC'  0210'              .WORD   TYP13   ;4  RP       SDED,LBCD
05AE'  0222'              .WORD   TYP14   ;2  NONE      INIR,CCIR
05B0'  0232'              .WORD   TYP15   ;2  REG      OUTP(ONLY)
05B2'  0239'              .WORD   TYP16   ;2  IREG     SPIX,SPIY
05B4'  0254'              .WORD   TYP17   ;2  IREG     POP X,POP Y
05B6'  025E'              .WORD   TYP18   ;3  OFS,DATA MVI D(X),N
05B8'  0272'              .WORD   TYP19   ;4  IREG,ADDR LXI Y,LXI X
05BA'  027C'              .WORD   TYP1A   ;4  IREG,ADDR SIXD,LIYD
05BC'  0299'              .WORD   TYP1B   ;3  IREG,REG  MOV D(X),A
05BE'  02B1'              .WORD   TYP1C   ;3  IREG,REG  MOV B,D(Y)
05C0'  02C6'              .WORD   TYP1D   ;4  IREG,BIT  BIT B,D(X)
05C2'  02DA'              .WORD   TYP1E   ;4  IREG,OFS  RLCR D(Y)
05C4'  0302'              .WORD   PRDB    ;UNDEFINE
                    ;
                    ;       CONDITION CODE TABLE
05C6'  4E5A         CCTAB:  .ASCII  'NZ'
05C8'  5A20                 .ASCII  'Z '
05CA'  4E43                 .ASCII  'NC'
05CC'  4320                 .ASCII  'C '
05CE'  504F                 .ASCII  'PO'
05D0'  5045                 .ASCII  'PE'
05D2'  5020                 .ASCII  'P '
05D4'  4D20                 .ASCII  'M '
                    ;
                    ;       REGISTER TABLE
05D6'  4243         RTAB:   .ASCII  'BC'
05D8'  4445                 .ASCII  'DE'
05DA'  484C                 .ASCII  'HL'
05DC'  4D41                 .ASCII  'MA'
                    ;
                    ;       ROTATE & SHIFT GORUP TABLE
05DE'  524C4352     RSTAB:  .ASCII  'RLCR'
05E2'  52524352             .ASCII  'RRCR'
05E6'  52414C52             .ASCII  'RALR'
05EA'  52415252             .ASCII  'RARR'
05EE'  534C4152             .ASCII  'SLAR'
05F2'  53524152             .ASCII  'SRAR'
05F6'  554E4446             .ASCII  'UNDF'
05FA'  53524C52             .ASCII  'SRLR'
                    ;
                    ;       BLOCK XFER CODE TABLE
05FE'  4920         BXTAB:  .ASCII  'I '
0600'  4420                 .ASCII  'D '
0602'  4952                 .ASCII  'IR'
0604'  4452                 .ASCII  'DR'
                    ;
                    ;
                    ;
                    ;       ****************
                    ;       * OPCODE TABLES *
                    ;       ****************
                    ;
                    ;       MAIN (1ST PASS) OPCODE TABLE
                    ;
0606'  FF           OTABM:  .BYTE   0FF
0607'  0000                 .WORD   0000
0609'  4E4F5020             .ASCII  'NOP '
060D'  FF                   .BYTE   0FF
060E'  0007                 .WORD   0007
0610'  524C4320             .ASCII  'RLC '
0614'  FF                   .BYTE   0FF
0615'  000F                 .WORD   000F
0617'  52524320             .ASCII  'RRC '
061B'  FF                   .BYTE   0FF
061C'  0017                 .WORD   0017
061E'  52414C20             .ASCII  'RAL '
0622'  FF                   .BYTE   0FF
0623'  00EB                 .WORD   00EB
0625'  58434847             .ASCII  'XCHG'
0629'  FF                   .BYTE   0FF
062A'  001F                 .WORD   001F
062C'  52415220             .ASCII  'RAR '
0630'  FF                   .BYTE   0FF
0631'  0722                 .WORD   0722
0633'  53484C44             .ASCII  'SHLD'
0637'  FF                   .BYTE   0FF
0638'  0027                 .WORD   0027
063A'  44414120             .ASCII  'DAA '
063E'  FF                   .BYTE   0FF
063F'  072A                 .WORD   072A
0641'  4C484C44             .ASCII  'LHLD'
0645'  FF                   .BYTE   0FF
0646'  002F                 .WORD   002F
0648'  434D4120             .ASCII  'CMA '
064C'  FF                   .BYTE   0FF
064D'  0732                 .WORD   0732
064F'  53544120             .ASCII  'STA '
0653'  FF                   .BYTE   0FF
0654'  0037                 .WORD   0037
0656'  53544320             .ASCII  'STC '
065A'  FF                   .BYTE   0FF
065B'  073A                 .WORD   073A
065D'  4C444120             .ASCII  'LDA '
0661'  FF                   .BYTE   0FF
0662'  003F                 .WORD   003F
0664'  434D4320             .ASCII  'CMC '
0668'  FF                   .BYTE   0FF
0669'  0076                 .WORD   0076
066B'  484C5420             .ASCII  'HLT '
066F'  C0                   .BYTE   0C0
0670'  0B40                 .WORD   0B40
0672'  4D4F5620             .ASCII  'MOV '
0676'  FF                   .BYTE   0FF
0677'  07C3                 .WORD   07C3
0679'  4A4D5020             .ASCII  'JMP '
067D'  FF                   .BYTE   0FF
067E'  00C9                 .WORD   00C9
0680'  52455420             .ASCII  'RET '
0684'  FF                   .BYTE   0FF
0685'  07CD                 .WORD   07CD
0687'  43414C4C             .ASCII  'CALL'
068B'  FF                   .BYTE   0FF
068C'  05D3                 .WORD   05D3
068E'  4F555420             .ASCII  'OUT '
0692'  FF                   .BYTE   0FF
0693'  05DB                 .WORD   05DB
0695'  494E2020             .ASCII  'IN  '
0699'  FF                   .BYTE   0FF
069A'  00E3                 .WORD   00E3
069C'  5854484C             .ASCII  'XTHL'
06A0'  FF                   .BYTE   0FF
06A1'  00E9                 .WORD   00E9
06A3'  5043484C             .ASCII  'PCHL'
06A7'  FF                   .BYTE   0FF
06A8'  00F3                 .WORD   00F3
06AA'  44492020             .ASCII  'DI  '
06AE'  FF                   .BYTE   0FF
06AF'  00FB                 .WORD   00FB
06B1'  45492020             .ASCII  'EI  '
06B5'  FF                   .BYTE   0FF
06B6'  00F9                 .WORD   00F9
06B8'  5350484C             .ASCII  'SPHL'
06BC'  FF                   .BYTE   0FF
06BD'  05C6                 .WORD   05C6
06BF'  41444920             .ASCII  'ADI '
06C3'  FF                   .BYTE   0FF
06C4'  05CE                 .WORD   05CE
06C6'  41434920             .ASCII  'ACI '
06CA'  FF                   .BYTE   0FF
06CB'  05D6                 .WORD   05D6
06CD'  53554920             .ASCII  'SUI '
06D1'  FF                   .BYTE   0FF
06D2'  05DE                 .WORD   05DE
06D4'  53424920             .ASCII  'SBI '
06D8'  FF                   .BYTE   0FF
06D9'  05E6                 .WORD   05E6
06DB'  414E4920             .ASCII  'ANI '
06DF'  FF                   .BYTE   0FF
06E0'  05EE                 .WORD   05EE
06E2'  58524920             .ASCII  'XRI '
06E6'  FF                   .BYTE   0FF
06E7'  05F6                 .WORD   05F6
06E9'  4F524920             .ASCII  'ORI '
06ED'  FF                   .BYTE   0FF
06EE'  05FE                 .WORD   05FE
06F0'  43504920             .ASCII  'CPI '
06F4'  FF                   .BYTE   0FF
06F5'  0008                 .WORD   0008
06F7'  45584146             .ASCII  'EXAF'
06FB'  FF                   .BYTE   0FF
06FC'  0C10                 .WORD   0C10
06FE'  444A4E5A             .ASCII  'DJNZ'
0702'  FF                   .BYTE   0FF
0703'  0C18                 .WORD   0C18
0705'  4A4D5052             .ASCII  'JMPR'
0709'  FF                   .BYTE   0FF
070A'  00D9                 .WORD   00D9
070C'  45585820             .ASCII  'EXX '
0710'  FF                   .BYTE   0FF
0711'  0ECB                 .WORD   0ECB
0713'  2E2E2E2E             .ASCII  '....'
0717'  FF                   .BYTE   0FF
0718'  0FED                 .WORD   0FED
071A'  2E2E2E2E             .ASCII  '....'
071E'  FF                   .BYTE   0FF
071F'  10DD                 .WORD   10DD
0721'  2858292E             .ASCII  '(X).'
0725'  FF                   .BYTE   0FF
0726'  10FD                 .WORD   10FD
0728'  2859292E             .ASCII  '(Y).'
072C'  E7                   .BYTE   0E7
072D'  0D20                 .WORD   0D20
072F'  4A522E2E             .ASCII  'JR..'
0733'  F8                   .BYTE   0F8
0734'  0180                 .WORD   0180
0736'  41444420             .ASCII  'ADD '
073A'  F8                   .BYTE   0F8
073B'  0188                 .WORD   0188
```

```
073D'  41444320          .ASCII  'ADC '
0741'  F8                .BYTE   0F8
0742'  0190              .WORD   0190
0744'  53554220          .ASCII  'SUB '
0748'  F8                .BYTE   0F8
0749'  0198              .WORD   0198
074B'  53424220          .ASCII  'SBB '
074F'  F8                .BYTE   0F8
0750'  01A0              .WORD   01A0
0752'  414E4120          .ASCII  'ANA '
0756'  F8                .BYTE   0F8
0757'  01A8              .WORD   01A8
0759'  58524120          .ASCII  'XRA '
075D'  F8                .BYTE   0F8
075E'  01B0              .WORD   01B0
0760'  4F524120          .ASCII  'ORA '
0764'  F8                .BYTE   0F8
0765'  01B8              .WORD   01B8
0767'  434D5020          .ASCII  'CMP '
076B'  C7                .BYTE   0C7
076C'  0204              .WORD   0204
076E'  494E5220          .ASCII  'INR '
0772'  C7                .BYTE   0C7
0773'  0205              .WORD   0205
0775'  44435220          .ASCII  'DCR '
0779'  EF                .BYTE   0EF
077A'  0302              .WORD   0302
077C'  53544158          .ASCII  'STAX'
0780'  EF                .BYTE   0EF
0781'  030A              .WORD   030A
0783'  4C444158          .ASCII  'LDAX'
0787'  CF                .BYTE   0CF
0788'  03C1              .WORD   03C1
078A'  504F5020          .ASCII  'POP '
078E'  CF                .BYTE   0CF
078F'  03C5              .WORD   03C5
0791'  50555348          .ASCII  'PUSH'
0795'  C7                .BYTE   0C7
0796'  0406              .WORD   0406
0798'  4D564920          .ASCII  'MVI '
079C'  CF                .BYTE   0CF
079D'  0303              .WORD   0303
079F'  494E5820          .ASCII  'INX '
07A3'  CF                .BYTE   0CF
07A4'  0309              .WORD   0309
07A6'  44414420          .ASCII  'DAD '
07AA'  CF                .BYTE   0CF
07AB'  030B              .WORD   030B
07AD'  44435820          .ASCII  'DCX '
07B1'  C7                .BYTE   0C7
07B2'  08C2              .WORD   08C2
07B4'  4A2E2E2E          .ASCII  'J...'
07B8'  C7                .BYTE   0C7
07B9'  08C4              .WORD   08C4
07BB'  432E2E2E          .ASCII  'C...'
07BF'  CF                .BYTE   0CF
07C0'  0601              .WORD   0601
07C2'  4C584920          .ASCII  'LXI '
07C6'  C7                .BYTE   0C7
07C7'  09C0              .WORD   09C0
07C9'  522E2E2E          .ASCII  'R...'
07CD'  C7                .BYTE   0C7
07CE'  0AC7              .WORD   0AC7
07D0'  52535420          .ASCII  'RST '
07D4'  00                .BYTE   00      ;END OF OTABM
                         ;
                         ;
                         ;       2ND PASS CB GROUP OPCODE TABLE
                         ;
07D5'  C0                OTABC:  .BYTE   0C0
07D6'  1100              .WORD   1100
07D8'  2E2E2E2E          .ASCII  '....'
07DC'  C0                .BYTE   0C0
07DD'  1240              .WORD   1240
07DF'  42495420          .ASCII  'BIT '
07E3'  C0                .BYTE   0C0
07E4'  1280              .WORD   1280
07E6'  52455320          .ASCII  'RES '
07EA'  C0                .BYTE   0C0
07EB'  12C0              .WORD   12C0
07ED'  53455420          .ASCII  'SET '
07F1'  00                .BYTE   00      ;END OF OTABC
                         ;
                         ;       2ND PASS ED GROUP OPCODE TABLE
                         ;
07F2'  FF                OTABE:  .BYTE   0FF
07F3'  0044              .WORD   0044
07F5'  4E454720          .ASCII  'NEG '
07F9'  FF                .BYTE   0FF
07FA'  0045              .WORD   0045
07FC'  5245544E          .ASCII  'RETN'
0800'  FF                .BYTE   0FF
0801'  004D              .WORD   004D
0803'  52455449          .ASCII  'RETI'
0807'  FF                .BYTE   0FF
0808'  0046              .WORD   0046
080A'  494D3020          .ASCII  'IM0 '
080E'  FF                .BYTE   0FF
080F'  0056              .WORD   0056
0811'  494D3120          .ASCII  'IM1 '
0815'  FF                .BYTE   0FF
0816'  0052              .WORD   0052
0818'  494D3220          .ASCII  'IM2 '
081C'  FF                .BYTE   0FF
081D'  0067              .WORD   0067
081F'  52524420          .ASCII  'RRD '
0823'  FF                .BYTE   0FF
0824'  006F              .WORD   006F
0826'  524C4420          .ASCII  'RLD '
082A'  FF                .BYTE   0FF
082B'  0047              .WORD   0047
082D'  53544149          .ASCII  'STAI'
0831'  FF                .BYTE   0FF
0832'  004F              .WORD   004F
0834'  53544152          .ASCII  'STAR'
0838'  FF                .BYTE   0FF
0839'  0057              .WORD   0057
083B'  4C444149          .ASCII  'LDAI'
083F'  FF                .BYTE   0FF
0840'  005F              .WORD   005F
0842'  4C444152          .ASCII  'LDAR'
0846'  CF                .BYTE   0CF
0847'  1343              .WORD   1343
0849'  532E2E44          .ASCII  'S..D'
084D'  CF                .BYTE   0CF
084E'  134B              .WORD   134B
0850'  4C2E2E44          .ASCII  'L..D'
0854'  E7                .BYTE   0E7
0855'  14A0              .WORD   14A0
0857'  4C442E2E          .ASCII  'LD..'
085B'  E7                .BYTE   0E7
085C'  14A1              .WORD   14A1
085E'  43432E2E          .ASCII  'CC..'
0862'  E7                .BYTE   0E7
0863'  14A2              .WORD   14A2
0865'  494E2E2E          .ASCII  'IN..'
0869'  E7                .BYTE   0E7
086A'  15A3              .WORD   15A3
086C'  55542E2E          .ASCII  'UT..'
0870'  C7                .BYTE   0C7
0871'  0240              .WORD   0240
0873'  494E5020          .ASCII  'INP '
0877'  C7                .BYTE   0C7
0878'  0241              .WORD   0241
087A'  4F555450          .ASCII  'OUTP'
087E'  CF                .BYTE   0CF
087F'  034A              .WORD   034A
0881'  44414443          .ASCII  'DADC'
0885'  CF                .BYTE   0CF
0886'  0342              .WORD   0342
0888'  44534243          .ASCII  'DSBC'
088C'  00                .BYTE   00      ;END OF OTABE
                         ;
                         ;       2ND PASS IREG OPCODE TABLE
                         ;
088D'  FF                OTBIR:  .BYTE   0FF
088E'  16E9              .WORD   16E9
0890'  50432E2E          .ASCII  'PC..'
0894'  FF                .BYTE   0FF
0895'  16F9              .WORD   16F9
0897'  53502E2E          .ASCII  'SP..'
089B'  FF                .BYTE   0FF
089C'  16E3              .WORD   16E3
089E'  58542E2E          .ASCII  'XT..'
08A2'  FF                .BYTE   0FF
08A3'  17E1              .WORD   17E1
08A5'  504F5020          .ASCII  'POP '
08A9'  FF                .BYTE   0FF
08AA'  17E5              .WORD   17E5
08AC'  50555348          .ASCII  'PUSH'
08B0'  FF                .BYTE   0FF
08B1'  1723              .WORD   1723
08B3'  494E5820          .ASCII  'INX '
08B7'  FF                .BYTE   0FF
08B8'  172B              .WORD   172B
08BA'  44435820          .ASCII  'DCX '
08BE'  FF                .BYTE   0FF
08BF'  1836              .WORD   1836
08C1'  4D564920          .ASCII  'MVI '
08C5'  FF                .BYTE   0FF
08C6'  1921              .WORD   1921
08C8'  4C584920          .ASCII  'LXI '
08CC'  FF                .BYTE   0FF
08CD'  1A22              .WORD   1A22
08CF'  532E2E44          .ASCII  'S..D'
08D3'  FF                .BYTE   0FF
08D4'  1A2A              .WORD   1A2A
08D6'  4C2E2E44          .ASCII  'L..D'
08DA'  F8                .BYTE   0F8
08DB'  1B70              .WORD   1B70
08DD'  4D4F5620          .ASCII  'MOV '
08E1'  C7                .BYTE   0C7
08E2'  1C46              .WORD   1C46
08E4'  4D4F5620          .ASCII  'MOV '
```

```
08E8'   00                      .BYTE   00      ;END OF OTBIR
                        ;
                        ;       3 BYTE OPCODE TABLE
                        ;
08E9'   C7              OTBIC:  .BYTE   0C7
08EA'   1D46                    .WORD   1D46
08EC'   42495420                .ASCII  'BIT '
08F0'   C7                      .BYTE   0C7
08F1'   1D86                    .WORD   1D86
08F3'   52455320                .ASCII  'RES '
08F7'   C7                      .BYTE   0C7
08F8'   1DC6                    .WORD   1DC6
08FA'   53455420                .ASCII  'SET '
08FE'   C6                      .BYTE   0C6
08FF'   1E06                    .WORD   1E06
0901'   2E2E2E2E                .ASCII  '....'
0905'   00                      .BYTE   00      ;END OF OTBIC
                        ;
                        ;
                                .END

                        ;
                        ;
                        ;
                        ;
                        ;
                        ;       ************************
                        ;       *                      *
                        ;       *    Z-80/ZAP DISASM   *
                        ;       * SYMBOL TABLE EDITOR  *
                        ;       *                      *
                        ;       ************************
                        ;
                        ;
                        ;
                        ;
                        .RADIX 16
                        ;
                        ;
                        .PABS
                        ;       EQUATES:
                        ;
B000                    ZMON    =       0B000
A400                    NSDOS   =       0A400
B745                    ZCI     =       ZMON+0745
B009                    ZCO     =       ZMON+9
B592                    LADR    =       ZMON+0592
B519                    CRLF    =       ZMON+0519
B597                    LBYTE   =       ZMON+0597
9C00                    SYMTB   =       NSDOS-0800
BE8D                    SYMBC   =       ZMON+0E8D
0008                    BS      =       'H'-40
000D                    CR      =       0D
000A                    LF      =       0A
B455                    TOM1    =       ZMON+0455
B548                    EXPR1   =       ZMON+0548
                        ;
BE87                    .LOC ZMON+0E87
                        ;
                        ;
BE87                    .BLKB 6
BE8D    00              .BYTE 0
                        ;
BE95                    .LOC ZMON+0E95
                        ;
BE95    21 BF4D         ENTRY:  LXI     H,SEDMG
BE98    062D                    MVI     B,SEMGL
BE9A    CD B455                 CALL    TOM1
BE9D    3A BE8D                 LDA     SYMBC
BEA0    F5                      PUSH    PSW
BEA1    CD B597                 CALL    LBYTE
BEA4    F1                      POP     PSW
BEA5    B7                      ORA     A
BEA6    2001                    JRNZ    BEGIN
BEA8    C9                      RET
                        ;
                        ;
BEA9    11 BEA9         BEGIN:  LXI     D,BEGIN
BEAC    D5                      PUSH    D       ;SET UP RETURN TO BEGIN
BEAD    CD B519                 CALL    CRLF
BEB0    0E24                    MVI     C,'$'
BEB2    CD B009                 CALL    ZCO
BEB5    CD B548                 CALL    EXPR1
BEB8    E1                      POP     H
BEB9    7C                      MOV     A,H
BEBA    B7                      ORA     A
BEBB    2801                    JRZ     START
BEBD    C9                      RET             ;TRY AGAIN
                        ;
BEBE    CD B519         START:  CALL    CRLF
BEC1    54                      MOV     D,H
BEC2    5D                      MOV     E,L
                        ;  **MULTIPLY ENTRY # BY 7BYTE/ENTRY**
BEC3    29                      DAD     H       ;TIMES 2
BEC4    19                      DAD     D       ;TIMES 3
BEC5    29                      DAD     H       ;TIMES 6
BEC6    19                      DAD     D       ;TIMES 7
BEC7    11 9C00                 LXI     D,SYMTB ;PT TO TABLE
BECA    19                      DAD     D       ;PT HL TO PROPER ENTRY
BECB    7E              LABEL:  MOV     A,M     ;GET 1ST CHAR
BECC    B7                      ORA     A       ;BLANK?
BECD    C8                      RZ              ;END OF TBL,SO RET
                        ;  **PRINT EXISTING LABEL**
BECE    0605                    MVI     B,5     ;MAX LABEL LENGTH
BED0    7E              PRINT:  MOV     A,M     ;GET LABEL CHAR
BED1    B7                      ORA     A       ;END OF SHORT LABEL?
BED2    2806                    JRZ     ENDP    ;..YES, END OF PRINT
BED4    CD BF49                 CALL    TYPE    ;TYPE CHAR
BED7    23                      INX     H       ;PT TO NEXT
BED8    10F6                    DJNZ    PRINT   ;TILL 5 CHAR'S
                        ;  **END OF LABEL**
BEDA    78              ENDP:   MOV     A,B     ;GET LABEL RESIDUAL COUNT
BEDB    85                      ADD     L       ;ADD TO HL
BEDC    6F                      MOV     L,A     ;MOVE BACK
BEDD    3001                    JRNC    NC      ;SKIP IF NO CRY TO H
BEDF    24                      INR     H       ;ADD IN CRY
BEE0    CD BF45         NC:     CALL    SPACE   ;PRINT SPACE
                        ;  **PRINT HEX LABEL VALUE**
BEE3    56                      MOV     D,M     ;1ST CHAR
BEE4    23                      INX     H       ;PT TO 2ND CHAR
BEE5    5E                      MOV     E,M     ;2ND CHAR
BEE6    EB                      XCHG            ;MOVE TO HL
BEE7    CD B592                 CALL    LADR    ;PRT AS 4 HEX
BEEA    CD BF45                 CALL    SPACE   ;PRT SPACE
BEED    EB                      XCHG            ;RESTORE HL & DE
BEEE    11 FFFA                 LXI     D,-6    ;BACKUP PTR TO
BEF1    19                      DAD     D       ;..START OF ENTRY
BEF2    E5                      PUSH    H       ;SAVE PTR FOR LATER
                        ;
                        ;       READ NEW LABEL TO TBL
                        ;
BEF3    1605                    MVI     D,5     ;MAX LABEL LENGTH
BEF5    CD B745         INPLP:  CALL    ZCI
BEF8    FE03                    CPI     'C'-40  ;CTL C?
BEFA    2847                    JRZ     ABORT
BEFC    FE08                    CPI     BS      ;BACKSPACE?
BEFE    2018                    JRNZ    NOTBS   ;..NO
BF00    7A                      MOV     A,D     ;MOVE RESIDUAL TO A
BF01    FE05                    CPI     5       ;START OF ENTRY?
BF03    28F0                    JRZ     INPLP   ;YES,SKIP BS
BF05    3E08                    MVI     A,'H'-40 ;GET A BS CHAR
BF07    F5              XBS:    PUSH    PSW     ;SAVE BS CHAR
BF08    CD BF49                 CALL    TYPE    ;ECHO BS TO CONSOLE
BF0B    3E20                    MVI     A,20
BF0D    CD BF49                 CALL    TYPE    ;DELETE EXISTING CHAR
BF10    F1                      POP     PSW     ;RESTORE BS CHAR
BF11    CD BF49                 CALL    TYPE    ;RESTORE TYPE POSITION
BF14    2B                      DCX     H       ;BACKUP TBL 1 CHAR
BF15    14                      INR     D       ;ADJ COUNT
BF16    18DD                    JMPR    INPLP   ;DO AGAIN
BF18    FE0D            NOTBS:  CPI     CR      ;CR?
BF1A    280F                    JRZ     ENDLB   ;END SHORT LABEL
BF1C    CD BF49                 CALL    TYPE    ;ECHO INPUT CHAR
BF1F    77                      MOV     M,A     ;STORE CHAR IN TBL
BF20    23                      INX     H       ;PT TO NEXT POSITION
BF21    15                      DCR     D       ;5 CHAR'S?
BF22    20D1                    JRNZ    INPLP   ;DO AGAIN TILL 5 CHAR'S
BF24    CD B745         ENDLM:  CALL    ZCI     ;GET CHAR
BF27    FE08                    CPI     BS      ;BACKSPACE?
BF29    28DC                    JRZ     XBS     ;DO BACKSPACE
                        ;  **END OF LABEL**
BF2B    CD B519         ENDLB:  CALL    CRLF    ;PRINT CR & LF
BF2E    7A                      MOV     A,D     ;GET RESIDUAL COUNT
BF2F    B7                      ORA     A       ;ZERO?
BF30    280A                    JRZ     NOLAB   ;YES,DON'T STORE ZERO
BF32    FE05                    CPI     5       ;NO LABEL CHANGE?
BF34    2806                    JRZ     NOLAB   ;LEAVE IT ALONE
BF36    AF                      XRA     A       ;GET A ZERO
BF37    77              PADLB:  MOV     M,A     ;STORE ZERO IN TBL
BF38    23                      INX     H       ;PT TO NEXT POSITION
BF39    15                      DCR     D       ;STILL MORE?
BF3A    20FB                    JRNZ    PADLB   ;DO TILL D=0
BF3C    E1              NOLAB:  POP     H       ;RESTORE PTR TO START
BF3D    11 0007                 LXI     D,7     ;5 FOR LABEL,2 FOR VALUE
BF40    19                      DAD     D       ;PT TO NEXT LABEL ENTRY
BF41    1888                    JMPR    LABEL   ;BACK FOR NEXT LABEL
                        ;
BF43    E1              ABORT:  POP     H       ;CLR STK FOR RET TO BEGIN
BF44    C9                      RET             ;RETURN
                        ;  **PRINT SPACE**
BF45    3E20            SPACE:  MVI     A,' '
BF47    1800                    JMPR    TYPE
                        ;  *TYPE CHAR**
BF49    4F              TYPE:   MOV     C,A     ;MOVE CHAR FOR ZAPPLE
BF4A    C3 B009                 JMP     ZCO     ;TYPE CHAR
                        ;
                        ;
                        ;
BF4D    53594D424F4C    SEDMG:  .ASCII 'SYMBOL TBL EDITOR, V1.0'
BF64    0D0A                    .BYTE   CR,LF
BF66    23204F462053            .ASCII  '# OF SYMBOL ENTRY = '
002D                    SEMGL   =       .-SEDMG
                                .END
```

PATCHES TO SWTPC BASIC:
CHANGE OUTPUT FORMAT OF NUMBERS

BY ROBERT L. PIGFORD
300 Wilson Road
Newark, Delaware 19711

As written by Robert Uiterwyck, SWTPC 8-K BASIC outputs numbers in a special format. All numbers having magnitudes smaller than 0.01 are printed in the scientific notation or E-format. The same is true for numbers equal to or greater than 10^9. Numbers in the intermediate range are printed in the F-format with a decimal point in the proper place. This is sometimes inconvenient and annoying if one is doing engineering or scientific computations. Arrays of numbers on the terminal may be arranged in a disorderly fashion if the magnitudes cover a wide range. Significant figures may be lost near the limits, especially if DIGITS has been set at a small value.

One way to reduce the problem is to change the bounds for the choice of the format. This can be done by changing the number at $156C in Flex BASIC to alter the upper limit. To change from 10^9 to 10^m use the BASIC command POKE (5484,m). To restore the original upper bound use POKE(5484,9). To change the lower bound the hex number in $1570 must be altered. Its original value is $FF or 255 decimal, corresponding to the output format 1.0E-02. To change the lower bound to 1.0E-0n, use the decimal numbers 1,0, 255,254 or the hex numbers 01,00,FF,FE for n = 0,1,2, or 3, respectively.

A better solution is to force all numbers to appear in the E-format. This can be accomplished by introducing one branch instruction in memory positions $1569 and $156A. These bytes should become $20 and $41, respectively. This will cause a jump across the part of the output program which chooses between the two formats. All output will be in scientific notation. With this change some memory space becomes available for additional patches, as follows.

Two additional patches can be made if desired to cause all E-format output to have the same length when displayed on the terminal. The first introduces a "+" sign just after the E in numbers having positive exponents. The second puts a space before positive numbers to correspond to the minus sign printed before negative numbers. The changes can be listed in assembly language as follows:

LIST OF MODIFICATIONS IN FLEX BASIC

A. Jump Instructions:

```
1569  20  41          BRA     $15AC   jump over choice of magnitude range
                                        for E and F formats
15C9  7E  156B        JMP     MINUS   jump for output of "-" in exponent
15CD  7E  1571        JMP     PLUS    jump for output of "+" in exponent
15E   7E  157F        JMP     SPACE   jump for output of "-" or space
                                        before number
```

B. Patch Program in Vacated Memory:

```
156B  BD  150F  MINUS   JSR     $150F   output "-" in exponent
156E  50                NEG B
156F  20  05            BRA     HERE    branch over "+" output
1571  86  2B    PLUS    LDA A   #'+
1573  BD  150F          JSR     $150F   output "+" in exponent
1576  4F        HERE    CLR A           initialize first exp. digit
1577  4C        LOOP    INC A           increment it
1578  C0  0A            SUB B   #10     decimal decr. second digit
157A  2A  FB            BPL     LOOP    loop if dig. greater than ten
157C  7E  15D3          JMP     $15D3   return from patch
                *
15?F  6D  00    SPACE   TST     00,X    get sign from number buffer
1581  2A  05            BPL     SP1     branch if it's positive
1583  86  2D            LDA A   #'-
1585  7E  1564          JMP     $1564   return from patch for output
1588  86  20    SP1     LDA A   #$20    put space char in A-reg.
158A  BD  150F          JSR     $150F   print it
158D  7E  15AC          JMP     $15AC   continue output in main program
```

These patches cause the number of columns used for both positive and negative numbers to be equal to seven, plus the number of digits specified in the DIGITS command. The number-output format in the revised BASIC will closely resemble that in FORTRAN when the E-format is called for.

PROGRAMMING PASTIMES AND PLEASURES

© 1978 CHARLES WETHERELL

COMPUTER GAME PLAYING AND THE GAME OF NEUTRON

Prefatory Note

Our Editor recently polled the readers and discovered that you are a serious minded, sobersided lot. In particular, you would like to get your daily dose of humor from another source. So this column will take on a more earnest tone. Still, topics will be chosen to emphasize the enjoyment and pleasure that can be gained from programming and computers. I hope you like the new style. And now to work.

Game Playing Programs

No doubt many *Dr. Dobb's* readers have been following the progress in computer chess. Probably some of you have thought of writing your own chess program and maybe some of you have tried. But a chess program is a large undertaking and can easily become a discouraging failure. Further, there are good reasons why even ardent and accomplished programmers should try some game other than chess.

First, one may well write a game program to learn something about the technical methods involved – alpha-beta pruning, evaluation strategies, tree searching, opening books, and the like. Unfortunately, the complexity of chess forces much of the effort into the chess-specific parts of the program, detracting from the learning. This complexity also means time must be spent making the program efficient, time perhaps better spent developing a more sophisticated program. Second, a great deal of the enjoyment of game programs lies in experimentation; devising better ways to include game specific knowledge is a major challenge. Once again, chess is so complicated that even minor advances take a long time to develop and incorporate into mature programs.

And that leads to our third reason for trying some other game. Chess has a glorious history. But this history is also a barrier to invention. Presumably, all the easy discoveries have been made. If you are going to increase the skill of a chess program beyond what is already known, you are in for a truckload of hard work. Newton said that he saw further because he stood on the shoulders of those who came before; he forgot to mention how hard it is to climb onto someone's shoulders.

The Game of Neutron

And so I would like to introduce you to the new game of *Neutron*. *Neutron* was invented by Robert A. Kraus and published in issue 71 of the excellent English magazine *Games & Puzzles*. The structure of *Neutron* is familiar, but it has several unusual twists. First I will describe the rules and then I will suggest some programming ideas.

Neutron is a two-player game played on a 5 X 5 square board; White owns the five white pawns at the bottom of the board and Black owns the five black pawns at the top of the board. The neutron begins in the center and doesn't belong to anybody. Figure 1 shows the board at the beginning of the game. Notice the coordinates used for recording moves.

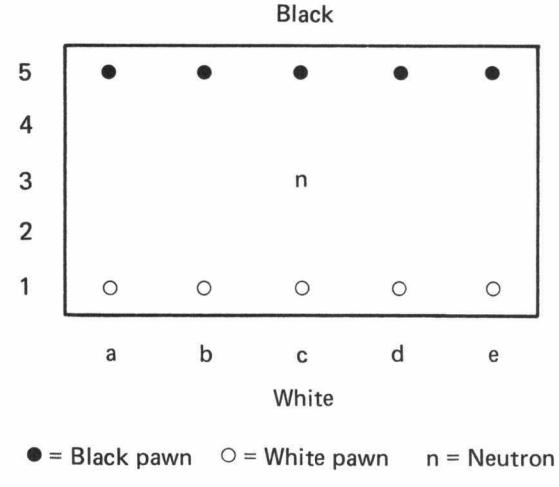


Figure 1. Beginning of Neutron.

Winning the game is simple; just maneuver the neutron onto your own back row. It doesn't matter if you move it there yourself or if you force your opponent to do the deed for you. As soon as the neutron shows up on your back row, you've won the game. You can also win by stalemating your opponent. That is, if your opponent gets trapped in a situation where he cannot complete his turn, you've also won the game. Each turn has two parts and it is easy to lose because both parts must be completed before the turn is legal.

White always plays first and the turns alternate—White, Black, White, Black, ... As I said above, a turn has two parts. First the player moves the neutron; then he moves one of his own pawns. Now it is a little easier to see how a stalemate might occur. If Black can trap the neutron—surround it completely with pawns—then White will not be able to complete even the first part of his turn and so will lose the game. There is one exception to the two part turn rule. On White's very first turn only, White does not move the neutron. This helps even out the advantage White gets from moving first.

Every piece, even the neutron, moves in exactly the same way—along a straight line horizontally, vertically, or diagonally. A piece stops moving just before it runs into another piece or the side wall; there is no jumping or capturing. But there is one unusual feature. When a piece moves, it must go as far as possible in the direction chosen. There is no stopping short. For example, White's pawn which begins on b1 can go to a2, b4, or e4, but it cannot stop on b3 or c2. Neutron pieces are a lot like politicians; they aren't smart enough to stop when they're ahead.

Just to make sure you understand the game, why don't you try to answer these questions about Figure 2? First, where can White's pawn on c1 move to? Second, what squares can Black's pawn on b2 move to? Third, can White win if it is his move? Finally, can Black win *if it is his move*? There are answers at the end of the column, but don't peek right away.

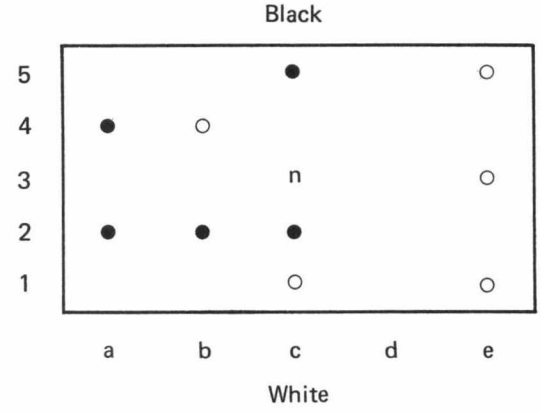


Figure 2. A sample game.

Ideas and Experiments

If you decide to write a *Neutron* program, you will surely begin by writing an alpha-beta tree searcher with a simple evaluation function. You can find out how to write such a program (if you don't already know) from my book *Etudes for Programmers* [1], from Nilsson's fine artificial intelligence book [2], or from almost any text in artificial intelligence. So far as I know, tree searching of complete trees is used by all current competitive chess programs.

After your program can play, you will probably want to experiment with its evaluation function, the routine which decides whether a board position is good or bad. Since *Neutron* is a new game, you are on your own in deciding what factors are important to a good position and you will almost certainly make some mistakes. However, your program can also help develop the evaluation function by learning as it plays. It seems to me that the techniques described by Samuel [3] on machine learning in checkers will apply directly to *Neutron*. Because *Neutron* is such a simple game, a good learning program would contribute greatly to our knowledge of machine learning.

Your program would be extremely impressive if it always won in endgames (at least, in those it had seen before). Murray and Elcock [4] describe an endgame learning program for the game of *Go-Moku*. At the end of each game, the program abstracts the board position into a specialized description language, along with the moves that led to the position. In the future, all potential moves are checked against the abstract descriptions; when a match is found, the computer knows that a forcing position exists. The research problems here involve development of an appropriate description language for *Neutron* positions and correct programming of the matching algorithms.

Although *Neutron* is a much simpler game than chess, it has sufficient tactical and strategic complexity to be an interesting object of study. And I think you will find that a game playing program for any game is not a small project. I would be interested in hearing from anyone who writes a good player for *Neutron* or any other small but non-trivial game.

Answers

1. a1, d1, d2.
2. a1, b1, a3, b3.
3. White moves the neutron to a3 and his pawn on e3 to b3.
   Now White wins because Black is stalemated.
4. Black moves the neutron to a1 and the pawn on a4 to b3.
   Now White must move the neutron to a5 giving Black the win.

Patching The SWTPC Co-resident Editor-Assembler

BY GEOFFREY A. GASS
5240 S. W. Dosch Road
Portland, Oregon 97201

The SWTPC Co-resident 6800 Editor-Assembler program at under $15 (cassette or paper-tape) is about as cheap an assembler as you're ever likely to find without resorting to outright theft.

But, of course, there's a catch. For that kind of money, the documentation is minimal: 9 pages, with almost nothing revealing the inner workings of the program. Further, SWTPC will not for any price deliver the original source code (they may not even *have* it!), nor will they work up custom patches for any system configuration other than that for which the code was designed.

The SWTPC assembler is essentially the Motorola MSM 1.2 Assembler (even containing the same logic error in branch computations—see box), but with a line-oriented editor replacing Motorola's character-oriented, "programmable" editor, and with the I/O routines set up to work with SWTPC products.

If your system is not specifically a SWTPC 6800 with MIKBUG (TM Motorola, Inc.), using the Control Port (I/O slot 1) to drive a cassette recorder via the SWTPC AC-30 interface, and having a SWTPC PR-40 printer interfaced via I/O slot 7A (addresses $801C-D), the assembler as furnished is going to need some patches.

The only clue that SWTPC gives you is that addresses $17A3 and $1A7F (documentation error; actually, $1A87 in the furnished version), contain instructions referring to the printer I/O addresses.

Disassembly of the whole program runs about 80 pages (with many halts because of texts and data intermixed with the instructions, and a certain amount of "garbage" left over from prior patchwork). However, your new patches will probably be confined mostly to the area between $17A2 (initialize printer-port PIA) and $1AA8 (output routine). Some of these areas are disassembled for you here—see Listing 2.

Yes, patching is gonna be some work. If you hire a programmer to do it for you at $25 (or even $2.50!) per hour, the total bill will be more than you'd have to pay for a first-class commercial, well-documented assembler. On the other hand, if you do it yourself, you'll have the satisfactions of having accomplished something useful, and of owning a quite serviceable assembler tailored exactly to your system, without the really arduous labor of constructing your own assembler from scratch.

HEREDITARY BUG IN 6800 ASSEMBLERS

The original MSM 1.2 6800 Resident Editor/Assembler contains a logic error which fails to look at the MSB of the source-destination difference when computing a branch whenever the LSB difference is 0. As a result, it accepts all branches which compute to even (hexadecimal) hundreds, and codes them all as "00" offset.

The error has been inherited by derivative assemblers, such as the SWTPC co-resident editor-assembler (at least through version 1.1) and the MSI disc-oriented CORES program.

While no programmer familiar with the specified limitations of the assembler would deliberately code "BRA*+258," it's common and quite easy to make what amounts to this same kind of mistake when reassembling large programs and working with symbolic operands. The Motorola assembler and its progeny don't catch the error, which can have serious consequences at run-time.

The bug is easy to fix, though. NOP out two bytes of code and change one other, and it's done (see Listing 1). The addresses shown are for the SWTPC and MSI versions. In the MSM 1.2 (paper-tape version), the start of the routine is at $0F01; for other versions, look 21 bytes past the location addressed (JMP nnnn) by location $05D8 in the second dispatch-table. The register I've called LCOUNT is at $0027-8 in the Motorola MSM 1.2.

—0—

When this assembler detects a branch error (it will detect 'em all after this patch), it assigns $FE as the machine-code offset, making the instruction branch to itself—a tight loop that will hang up at run-time, but will do no damage with conditional and unconditional branches.

However, the assembler will treat a BSR the same way. And with a tight loop BSR, the same return address will be added to the stack again and again, until the overflowed stack reaches the loop and over-writes it, after which the program will run down through the addresses, trying to execute them as instructions. A more thoroughly devastating bomb could hardly be devised.

The penalty for running a program in spite of detected errors is made particularly severe in this case. There won't be many programmers who do it more than once!

When you're through, here's what you'll have:

1. **Text Editor.** The line-oriented text-editor lets you compose and edit assembly-language programs—or any other text material—in memory, and then display it, print it, output it to tape, or (if you followed the rules) turn it over to the assembler for conversion into a machine-language ("object code") program and formatted assembly listing. The editor features automatic line-numbering, resequencing and a "search" capability of finding each occurrence of any character-string in your text.

2. **Assembler.** The two-pass assembler reads the test file and generates a machine-language program from your labels, mnemonics and directives, calculating your branches and doing any necessary decimal/hex/octal conversions—or, if it can't, telling you why. The assembler can output a full "assembly listing," neatly formatted, to your terminal or printer, and punch object-code tapes in the Motorola ASCII/hex S-format for loading the program into memory and running it.

The editor/assembler programs occupy all of memory from $00F0 to $1B33 (and also use all, or nearly all, addresses from $0000 through $00EF for registers, buffers, stack and flags). Your organized text file extends from $1B8C to within a few bytes of the top of your contiguous memory (the editor kicks out a "CORE FULL" message if you run off the end of memory). However, for the assembler to work, an additional 24 free bytes are needed, plus 8 bytes for every label or symbol you've used.

The recommended minimum memory size for operation of the editor-assembler is 12K ($0000-$2FFF). This allows about 5000 bytes for your text file—enough for 3- to 5-page programs, depending on your use of comments and the number of symbols and labels used. For straight text, 5000 bytes would allow about a page and a half of single-spaced 72-character lines.

The two-pass assembler spends the first pass collecting all labels and symbolic operands, and getting or computing values for them. If any are undefined, they are listed on your terminal or printer. It also checks to see if you've given the program a name (and gives you an error message if you haven't). It also watches for an 'END' directive, and if it doesn't find one, proceeds to "assemble" your entire memory address space with an error-message output for every few bytes. The first pass also gives you error-messages for all syntax errors (e.g., misspelled words in the mnemonic column) —but doesn't always tell you where the errors are! There is just one usable option for Pass 1 (1P).

After the first pass, you have a chance to go back to the Editor and clean up any detected errors.

The second pass goes through the text file again, line-by-line, generating object code and formatting a full "assembly listing." The operator can select one of the three modes here:

2L: Type or display assembly-listing only—no tape.
2T: Punch an object-tape only—no listing.
2P: Listing plus Tape: Type the listing until a block of object code is complete, halt the listing and ouput a block of object code to tape, then switch back to displaying or printing the listing again until the file is exhausted.

NOTE: Where the printer and punch are in the same physical device sharing a common data bus—as in the ASR-33 or a Dura/Itel Selectric machine—the 2P pass is useless unless the devices can be selectively enabled and disabled by control characters.

The SWTPC documentation does not mention it, but the program also accepts a "1S" pass command. This mode is supposed to perform an assembly using a pre-existing symbol-table (very useful if you are assembling a giant program in segments with lots of cross-references). Unfortunately, the "1S" pass is not supported by the rest of the code, so its acceptance by the input filter is meaningless* and can be eliminated by a minor code-change.

Problem Areas—General.

Even if your hardware system conforms exactly to SWTPC's expectations, there are a few problems which may surprise you:

The first is in loading the original tape. For some reason, the data on the tape starts at $0020 and runs on well past $1B32, which is the last actual instruction. My tape-loader program uses the area #0020-$002F for format, mode, checksum and load-limit registers, and so got clobbered by the incoming data. Because this loader also accepts selective-loading limits, I was later able to start the loading at $0030 and ultimately discover that nothing on the SWTPC tape below $00FE is permanent data, and never need be loaded or saved.

The second is in the program's extensive use of Page 0 when running, including the area $0000-$001F. Motorola refuses to accept software using this area for any purpose, since it is reserved for their operating system registers. Because of this reservation, other operating-system designers have used this same area for critical flags, addresses and registers. Any of these operating systems will be *clobbered* by the SWTPC assembler. This, by the way, is not due to any of SWTPC's patching. The original Motorola MSM 1.2 from which it was derived tramples all over the $0000-$001F locations, and will blow away any operating system relying on data in this area *—or anywhere else on Page 0.*

Third, there are many routines in the assembler which use the Stack Pointer in conjunction with the Index Register to transfer data from one location to another. For some, *but not all* these routines, the interrupt mask is set before the routine, thus preventing an interrupt from dumping the CPU registers into your data file. In most of the routines which set the interrupt mask at the start of the routine, the flag is *cleared again at the end.* So what's bad about that? In a system equipped with interrupts but in which they are normally masked off, the vector location may not even be set up. When the assembler gratuitously clears the mask, any IRQ or snivet from a peripheral can send the program off to a random location—usually a pointer to bombsville.

In some of the routines using the SP, there's *no* initial setting of the interrupt mask. I have not verified to my satisfaction that the mask is always set elsewhere before these routines are entered, and I regard them as potential sources of

*It's possible that a prior version of the assembler put the symbol table at the very top of memory, working backward toward the text file. The SWTP version starts the symbol-table immediately following the text-file, and uses the end-of-text pointer as the start-of-symbol-table pointer. It would take major code-changes now to get that symbol-table back where it belongs and make it increase downward instead of upward, to allow a usable "1S" assembly pass.

trouble. The various CLI instructions are located at $0765, $0A1F, $12AD and $13D3. Except in interrupt-driven systems, these CLI's should be NOP'd out.

Fourth, there's the logic error in the assembler itself (see box), which fails to flag a significant program error. This needs fixing in all cases.

Fifth, there's a routine at $08F4 near the start of the Assembler proper which locates the top of contiguous memory by stuffing $55 into every location above the top of the text file. The first location that doesn't read $55 is saved at "MEMEND" ($0003), and this address -8 is saved at "LASTSY" ($0053). If you have utilities to protect at the top of memory you'll want to NOP out $08F4-$08FE and put in a "LDX #nnnn" at $08FF (where nnnn is the first address to be protected).

NOTE: The *Editor* can still overwrite your utilities—it doesn't use MEMEND: it just writes until some address fails to respond, and then kicks out that "CORE FULL" message. But at least it doesn't *automatically* chew up all of memory as this assembler routine does.

Problem Areas—System Dependent.

Each departure of your system from the package envisioned by SWTPC will generate its own little set of problems.

If your system doesn't use MIKBUG, SWTBUG, RT/68 (TM's Motorola, SWTPC and Microware Systems Corp., respectively), or other MIKBUG-compatible monitor, you have quite a few problems.

First in this category is the Stack Pointer. In MIKBUG-based systems, the power-on restart initializes the SP to $A042 for monitor operations. On receipt of a "G" (go to user program) command, the CPU executes an RTI instruction, which ratchets the SP up to $A049 as the CPU registers are loaded from locations $A043-A049, with the contents of $A048-9 going into the Program Counter.

It's common practice among 6800/MIKBUG programmers to execute a BSR or JSR instruction just ahead of the soft-restart (a point where the program can be reentered after a system reset without the program resetting pointers, clearing memory or otherwise spoiling work in process), re-entry point in a program, then to set the SP to a position two bytes lower than its initial value. The return address from the JSR or BSR is thus "protected" on the stack until it's revealed again by a system reset. After a system reset, a "G" command will restart the program at this "safe" location.

In the case of the SWTPC assembler, this JSR and SP Reset (to $A047) occurs in such a way as to "save" a reentry at a location *following* printer-port initialization and just *ahead of* some pointer re-setting. Therefore, if you do a system-reset-and-go, (a) your printer port will be dead (a PIA is disabled by a system reset and must be reinitialized) and (b) your text-file line-number registers will be messed up (not a fatal error).

There are 15 (one is the "routine" address $17A2 for the NEW command, hidden in the Command table at $15E2) program references to the initialization area $17A2-$17C1, so any patchwork here must be done with care. The DESEQ operation, for instance, jumps right into the middle of this code at $17B3 to finish its work. I have not yet bothered to fix this awkwardness in my assembler, but simply reinitialize the PIA manually from the keyboard and hand-enter the "soft-restart" address of $0103 at $A048-9 after a system reset—this address will thereafter be protected on the stack for subsequent resets. Where operators unfamiliar with machine-language patching and poking will be using the program, the whole initialization area should be rewritten, to allow a clean reentry after reset.

In any case, for systems not having read-write memory at $A000-$A07F, the stack-setting commands at $17C1 (in the Editor) and at $1870 (PATCH command) will have to be changed. The assembler proper (as distinguished from the Editor routines) relocates the Stack to $00EA, reverting to $A047 on return to Editor.

Somewhat more complex are the MIKBUG INEEE/OUTEEE problems. The OUTEEE ($E1D1) routine is called for output either to the terminal or to the tape, depending on the state of control and data lines at I/O Port 1 ($8004-7). Similarly, INEEE can be either keyboard input or tape input, depending on the state of the software-controlled lines.

The control lines are set by entering the routine I have called CTLSET at $1A12 with a bit set in ACC B as follows:

B = $04 OUTEEE will address tape
B = $08 OUTEEE will address terminal
B = $20 INEEE will read from tape
B = $10 INEEE will read keyboard input

The hardware patches to get all these good things to happen are vouchsafed only to purchasers of the SWTPC AC-30 cassette interface.

The appropriate control character $12 (DC2, punch on), $14 (DC4, punch off), $11 (DC1, reader on) or $13 (DC3, reader off) is output via OUTEEE just before calling CTLSET. ASR-33 and some other keyboard/tape terminals can be wired to respond to these characters to enable and disable desired subsystems.

The OUTEEE pointer at $1A79 is used *only* for the output of these control characters, and so may be abandoned if not needed for this purpose (however, the pointer is conveniently located for use by OUTQ, as will be described later).

Table 1 shows all of the JMP's and JSR's to MIKBUG I/O routines in the SWTPC Editor/Assembler.

Just to make life interesting, two of the MIKBUG I/O references are to *pointers* in MIKBUG, which may not be covered by all "MIKBUG-compatible" monitors. At $014D, the assembler calls $E078 "INCH", a pointer to INEEE. At $01FA, the assembler calls $E075 "OUTCH", a pointer to OUTEEE.

Even in MIKBUG-based systems, if tape I/O is not performed via the terminal/control port, the various tape-in and tape-out INEEE and OUTEEE operations will have to be segregated.

Fortunately, this can be done without too much difficulty, since the calling routines are already separate. The PDATA1 (string-output) call at $0172, for instance, is *only* used during the 2P or 2T (object-tape output) pass of the assembler for tape output and for no other purpose: no fear of missing prompts or data if this pointer is redirected to a dedicated tape-output routine.

Table 1—References to MIKBUG I/O Routines

```
Ø13Ø    JMP   $EØDØ   START:  MIKBUG power-on restart
Ø14D    JSR   $EØ78   INCH:   Pointer to INEEE
Ø16E    JSR   $E1D1   OUTEEE: Output content of ACC A to terminal
```

```
Ø172   JMP   $EØ7E   PDATA1: Output string per XR to terminal
Ø1FA   JSR   $EØ75   OUTCH:  Pointer to OUTEEE
Ø919   JSR   $EØ7E   PDATA1
162Ø   JSR   $E1AC   INEEE:  Input to ACC A from terminal
1876   JMP   $EØE3   CONTRL: Re-entry to MIKBUG monitor
19BE   JSR   $E1AC   INEEE
19C5   JSR   $E1AC   INEEE
1A79   JMP   $E1D1   OUTEEE
1AA4   JMP   $E1D1   OUTEEE
1B2D   JSR   $EØC8   OUT4HS: Output 2 bytes per XR as 4 hex
                     ASCII characters followed by a space
```

These are the ultimate pointers used for various I/O operations—usually via a deep nest of subroutine calls:

Command input to Editor	JSR INEEE at $162Ø
Text input to Editor	JSR INEEE at $162Ø
Command (Pass I.D.) input to Assembler	JMP INCH at $Ø14D
Interrupt to STRNGO operations	LDA A $8ØØ4 at $167F
Output to terminal (READY message, line numbers, CRLF's, LIST output, 2L pass listing output, error messages, etc.)	JMP OUTEEE at $1AA4
"Pass" prompt in Assembler	JSR PDATA1 at $Ø919
"Size" report	JSR OUT4HS at $1B2D
Tape input (LOAD routine)	JSR INEEE at $19BE, & JSR INEEE at $19C5
Tape output (SAVE only)	JMP OUTEEE at $1AA4
Tape data output (2P or 2T pass)	JMP PDATA1 at $Ø172
Tape leader & trailer (2P or 2T pass)	JSR OUTEEE at $Ø16E
Printer output (all modes)	PRINTC routine $1A83-$1A9C
Printer PIA initialization	In-line code, $17A2-$17AB

A list of all valid references to I/O ports is shown in Table 2. A code-search also reveals "TST $8006, INC $8006, DEC $8006" in the $0295 to $029D area, but these are just uncollected garbage from a previous tenant, and may be ignored.

An I/O-associated DELAY routine is located at $0605-$0616. The routine is only required when software-controlled mag-tape (cassette) start-stop is used. The routine is only called by the PCHON routine ($1A3A), by a JSR DELAY at $1A44. If the DELAY routine is not used, the space $0605-$061C ($0616-1C is more garbage) is available for other uses.

The "interrupt" in STRNGO is valid only for use with MIKBUG's "software UART." The routine looks for a low (start bit) in the MSb position of the PIA *data* register. For normal I/O systems (PIA or ACIA), the routine should look for a *high* (IRQ) at the MSb position of the *control* register of the interface adapter.

Table 2—References to I/O Ports

```
167F   LDA A  $8ØØ4
17A2   LDX    #$81C
1A15   LDX    #$8ØØ4
1A5F   STA A  $8ØØ7
1A6F   STA A  $8ØØ7
1A86   LDX    #$81C
```

Modifications for Independent I/O Devices.

Where the terminal, printer and tape I/O all must be independently addressed, the CTLSET routine may be abandoned. It is called from just four locations—the PCHON, PCHOFF, RDRON and RDROFF routines, all in same area. There are no "external" calls to this routine. RDRON and RDROFF are called only by the LOAD routine, and if un-needed, they can be abandoned, or modified as required.

PCHON and PCHOFF are called from several locations. It's easy to just make the first instruction at each location an RTS, but if bytes are precious, Table 3 shows the locations of all the PCHON and PCHOFF calls, which may be disabled to allow dispensing with PCHON and PCHOFF altogether. However, it may be desirable in some systems to retain these routines, as suggested below.

NOTE: In the optional 2P assembler pass, the tape is turned on and off repeatedly as the program switches back and forth between printing and listing and outputting object-code. If mag tape is your object-code medium, but you do not have software-controlled start-stop, I'd recommend disabling the "2P" pass, and using a flag system as outlined below to prompt the operator to start and stop the tape, using only the "2T" pass (continuous object-code output).

Table 3—References to PCHON ($1A3A) and PCHOFF ($1A48)

```
Ø1A2   JSR   $1A3A
Ø1FF   JSR   $1A3A
Ø2Ø6   JSR   $1A48
1548   JMP   $1A48
1983   JSR   $1A3A
1A7F   JSR   $1A3A
```

The central routine which handles most output operations is OUTQ at $1A9D. As set up, it looks at location $01C8, "PFLAG". If the value there is $FF, it outputs the character in ACC A to the terminal via the OUTEEE pointer at $1AA4. If the value is 00, it branches to PRINTC at $1A83 for output via the printer.

OUTQ was carefully written to preserve ACC B, but if you look at PRINTC, you'll notice that ACC B is clobbered there, so OUTQ needn't be so fussy. Leaving out those PSH B and PUL B instructions leaves enough space to make OUTQ a three-way (or even four-way!) output routine. If an intermediate value code ($01-$7F) is used at PFLAG for "punch," OUTQ can be steered as follows:

```
1A9D   OUTQ    TST $Ø1C8     Sets or clears Z and N flags
1AAØ           BEQ PRINTC    ØØ = printer
1AA2           BMI OUTEP     $FF = Terminal. OUTEEE pointer at $1A79
1AA4           BRA PUNCH     New routine for tape output.
```

This three-way split requires that the PCHON routine be retained, but modified to set PFLAG at $01C8 to the new "punch" code value. A return to "command" mode always resets PFLAG to $FF. If the new PUNCH routine is located in the former CTLSET area, the last instruction of OUTQ would have to be a JMP—it's too far for a BRA. However, the nine bytes following OUTQ ($1AAB-$1AB3) are really unnecessary. They set $01C8 to $FF in response to the TE(rminal) command. Since a return to COMAND at $17BC always does this anyway, the address in the command table at $1608 for implementing the TE command can simply be changed to $17BC, thus opening up the entire area from $1AA4 through $1AB3 for a PUNCH routine (16 bytes).

With OUTQ set up as a three-way output, two more changes need to be made to steer data to OUTQ for punch operations. SAVE, including leader and trailer punching, already calls OUTQ.

For the 2T and 2P assembly passes, however, these two changes are required:

Leader and trailer tape generation for 2P and 2T passes are done by a routine at $0139. It calls $E1D1 (OUTEEE) at $0161. The code at $061E should be changed to JSR OUTQ (BD 1A 9D).

The object-code data output is handled by a pointer at $0172 to $E07E (PDATA1). This pointer should be redirected to the STRNGO routine at $1688.

A special problem exists for manually-controlled cassettes or paper-tape punches which must be manually enabled prior to punch operation. If the PCHON routine includes an operator-prompt and halt for acknowledgement, there will be several of these (and also PCHOFF calls) even in the 2T pass of the assembler, an annoyance in the case of paper tape, and a waste of tape and loading time for the cassette.

Under these circumstances, the PCHOFF location should be disabled by an RTS in the first position, and the PCHON routine should look to see if the PFLAG is already set to the "punch" code—and if it is, exit. Only if the flag is *not* already set should the prompt text be displayed and a JSR to the keyboard for operator go-ahead be performed.

CAUTION: On exiting the PCHON routine, ACC A must always contain a non-zero value. The value in ACC A after PCHON is called is put into location $0025 as a pass flag. Zero is the Pass 1 flag; any non-zero value is the flag for Pass 2.

In the case where the 2P pass is to be supported (switching back and forth between assembly listing and tape output), the PCHON and PCHOFF routines will have to be quite different. PCHON must *save* the previous PFLAG value ($00 or $FF); PCHOFF must restore the previous value. Supporting the 2P pass using a cassette without software-controlled start-stop is really not practical (a halt and restart for every new ORG directive and for every 27 bytes of object code). For a paper-tape punch requiring just one turn-on or arming, however, the job can be done in one of two ways:

(1) The driver routine for the punch can incorporate a "time-out": a character is transmitted to the punch, and if the punch fails to respond with a handshake response within—say —100 ms, the character is saved, a prompt is sent to the terminal, and the program halts for operator action. When the operator hits any key on the keyboard, the saved character is recovered and another attempt is made to output it. With this scheme, the PCHON routine may omit the "turn-on" prompt and need only look at PFLAG. If the punch flag is set, exit. If the punch flag is not set, save the old flag at OLDFLG and substitute the punch code in PFLAG. PCHOFF would perform the opposite function: if the punch code is not in PFLAG, exit. If the punch code is set, recover the OLDFLG value and put it into PFLAG and exit.

A punch driver routine with a 100 ms time-out uses up about 50 bytes of code, however (plus the operator prompt-text), so finding room for it in the I/O area might be difficult.

(2) Without the "time-out" feature, another flag needs to be set up, which is cleared during initialization. PCHON looks first at this flag. If it is not set, PCHON requests a turn-on with a prompt-message to the terminal, waits for operator response, then sets the flag. If this new "PON" flag is set, it skips the prompt message. Then PCHON looks at PFLAG. If PFLAG is $FF or 00, that value is saved at OLDFLG, and the punch code is put into PFLAG. If the value at $01C8 (PFLAG) is already the "punch" code, PCHON exits without changing it.

With this scheme, the operation of PCHOFF is the same as under (1): if $01C8 is $FF or 00, PCHOFF exits. If the punch code is set, PCHOFF gets OLDFLG and puts it into PFLAG.

One new element must be added—getting "PON" cleared. This can be done in the ASSEMBLE routine: the main routine is set up to jump back to $17BC (COMAND routine in Editor) on receipt of a CTRL/X (CAN) input from the keyboard. This can be changed to a jump to a short routine that clears PON and then jumps back to COMAND.

The SAVE routine (tape copy of text file) can call the modified PCHON, perform its output via OUTQ, call PCHOFF, and then exit via the routine that clears PON. SAVE has its own in-line coding for leader punch (10 bytes of code) and for trailer punch (another 10 bytes). If these are consolidated into a single subroutine, even the addition of two BSR's and an RTS will still leave 5 free bytes of code, allowing the PON-clearing routine to be added at the end of SAVE. This entry-point can be the jump destination for the CAN (CTRL/X) escape from ASSEMBLE for case (2) above where the driver-routine does not have a "time-out" feature.

If SAVE is aimed at a paper-tape punch, the "LDA A #$FF" instruction should be changed to "LDA A #0" or

"CLR A" instruction. The $FF (all holes) leader is not good for paper tape—tears easily, and you can't write on it. Blank leader is better.

Problems for Specific Devices.

Sheet-fed printers. The assembly-listing printer output assumes that you have roll or fanfold paper in your printer. Every 54 lines of text it kicks out a string of line-feeds, a few hyphens to mark the end of the page, and some more line-feeds, then types the header for the next page.

This continuous printer output provides no way to change paper in a sheet-fed printer.

However, a look at the PAGEND routine at $06C5 reveals a strange anomaly: the current page number is maintained in a 16-bit register, counting out page numbers up to 65,535! (Actually, the first two decimal digits are suppressed in the output routine, so you only see page numbers up to 999.)

If you are willing to live with only 255 page numbers (at 54 lines per page and at least 8 bytes per line, that would require a minimum of 110k bytes of memory to hold your text file, unless you use an awful lot of PAGE directives), the 16-bit arithmetic in PAGEND can be supplanted with a LDA B PAGENO, INC B, STA B PAGENO, CLR A sequence, and then bounce off to the binary-to-decimal ASCII conversion routine at $08B7 (via a pointer at $05C6).

This change opens up enough bytes for a call to your keyboard input routine (JSR INEEE, or whatever) which would hold up the printer output until you hit some key on the keyboard, and allow you to change the paper.

Now, down at $06E0, the XR is loaded with $056C, which is the start of the text string that produces a series of line-feeds, the hyphens to mark the page-end and the header for the new page. You don't need all those line-feeds or hyphens any more, so change the value loaded into the XR to $0574, a later point in the text string, just past the code for the hyphens. Your printer will now do one CR and three LF's before printing the new page number.

Limited-Size CRT's. The NEXTLN routine which leads to the PAGEND routine makes no distinction between a printer which has a nominal 66 lines to a page and a terminal which may not have this kind of display capacity.

If you have a 16-, 24-, or 32-line display capacity on your terminal and would like to be able to read a "2L" pass without gobbling up a lot of paper, there are two changes you can make in the PAGEND routine which will facilitate a glass-terminal display.

At $06C5, ACC A picks up $CA, which is the complement of the line-count which will make up a page (54 lines of data: the other 12 lines are supplied by the 11 LF's in the header string and one other CRLF I haven't spotted yet).

Instead of having ACC A pick up the $CA code, make it ACC B (LDA B #$CA). Now, JSR to a new PAGHLD routine which includes:

```
PAGHLD    JSR INEEE   Wait for keyboard input (ACC A will be clobbered)
          LDA A PFLAG Check printer flag ($01C8)
          BEQ TRET    00? Printer on. 54 lines.
          LDA B #$F4  $FF: Terminal: 16 lines minus header.
TRET      RTS         Return with complement of LINECT - 4 in ACC B
```

Now, when you return, put ACC B into LINE ($0019) and do the page-number incrementation bit.

If you take out the LF's from the text string at $0575-6-7 and substitute the control characters for home-up and erase (assuming that your printer doesn't balk at receiving these characters and that your terminal is wired to use them), then changing the XR loading which takes place at $06E0 to get the value $0574 (as suggested above for a sheet-fed printer), will give you a "page" routine compatible with both a sheet-fed printer and a page-mode terminal (the home-up/erase can be omitted for a scrolling type).

For compatibility with a glass terminal and continuous-paper printer, the terminal's page-mode operation with automatic home-up and erase will have to be sacrificed, as the printer must have *all* of those LF's in that string for the page-length to come out right.

In this case, the added subroutine could look like this:

```
PAGEHLD   LDA A PFLAG   Check $01C8
          BEQ TRET      Printer on? No halt.
          JSR INEEE     Terminal. Hold for keyboard
                                          release.
          LDA B #$F0    16 lines for 16-line terminal
TRET      RTS
```

And put ACC B into $0019 as above. Your terminal will start a new page by doing the 11-line header-trailer routine, and then your listing. For more than a few lines of listing you'll need to operate the terminal in *scroll* mode, so the header material will run off the top and your listing will be in proper order. (By the way, for any of these glass-terminal schemes to work with a 64-character line, you will have to limit your comments to 18 characters or less (40 characters in all-comment lines), or the formatted listing will overrun the line-ends and thus generate extra lines not counted by the program's line-counter. If you have an 80-column terminal (lucky you!), you'll see everything that's actually saved in the file).

The PAGEND routine is skipped over during tape output, so modifications between $06C5 and $06E5 will not affect anything but printer and terminal operations.

CR-LF Problems. (Selectric-based printers & some terminal types)

(a) If your terminal or printer automatically does a line-feed for every carriage-return (nearly all Selectric-based printers must do this—it takes a major mechanical modification to prevent it), you'll want to suppress the extra LF, but without suppressing *all* LF's. Here's a possible routine:

```
PRINTC    LDA B OLDCHAR   Previous character output
          STA A OLDCHAR   ACC A has the new character
          CMP A #$0A      New character a LF?
          BNE PRINT2      If not, output it
          CMP B #$0A      New one is LF.  Is old one also LF?
          BNE NOPR        Only accept multiple LF's
PRINT2    JSR PRINTO      Driver routine
NOPR      LDA A OLDCHAR   (If clobbered by PRINTO)
          RTS
```

This routine skips the first LF after a CR, but no others.

(b) A terminal which performs a line-feed automatically for every carriage-return will quickly uncover a small annoyance built into the output-buffer loading routine. Any line containing a comment which does not fill or overflow the buffer will get two carriage-returns instead of one. Thus two line-feeds when using this equipment.

The problem is in the algorithm which loads the CRLF into the buffer when it detects the end of the comment. The input buffer where the comment was stored had been pre-filled with CR's ($0D). The program detects the end of the comment by watching for a CR. It looks only *after* a character is loaded into the output buffer, then jumps to a routine which adds a CRLF into the buffer.

The fix for this one is to change the logic to look for a CR *first*, then if the character is *not* a CR, put it into the buffer and loop. If the character is a CR, the routine jumps to the add-a-CRLF routine.

By skipping the load operation, there's one byte in the output buffer skipped over (1st byte after the end of the comment). Fortunately, the output buffer had been pre-loaded with *spaces* ($20)—so the skipped byte is just an extra space before the CRLF—no problem.

Listing 3 shows the exact patch required to get rid of the extra CR.

Terminals Responding to NAK Character. The control character NAK ($15) is incorporated in several CRLF routines in the Editor and Assembler. If this control character produces adverse reactions in your printer or terminal, it will have to be changed. In some "smart" terminals NAK can trigger a repeat of the last line-buffer dump; in mine, it just turns off the cursor.

A BEL code would be appropriate for the "WHAT??" and "CORE FULL" messages; for the CRLF generated when you terminate a line and for the READY message, NUL's would be better. Table 3 shows the locations of the various NAK codes.

Table 3--Location of NAK Characters in Text Strings

```
Ø61F    In "ENTER PASS" text string

15BE    In "READY" text string

15C7    In "WHAT?" text string

15DØ    In "CORE FULL" text string

169D    In general CRLF text string (starts at $169B)
```

Terminals or Printers Responding to the 8th Bit. The assembler flags every defined symbol in the symbol table by changing the MSb of the first character to a "1". This bit is not scrubbed off when the symbol table is printed under an OPT S directive. If your terminal or printer driver routines do not delete this bit (OUTEEE does not), you may get an unexpected reaction from your printer or terminal. My terminal, for instance, is set to display reverse video for characters so flagged; the code-conversion for my (Selectric) printer outputs nothing for codes $80-$FF. If you have similar problems, the OUTQ routine should include an AND A #$7F instruction to reset the undesired bit.

LISTING 1 -- CORRECTED BRACHK ROUTINE

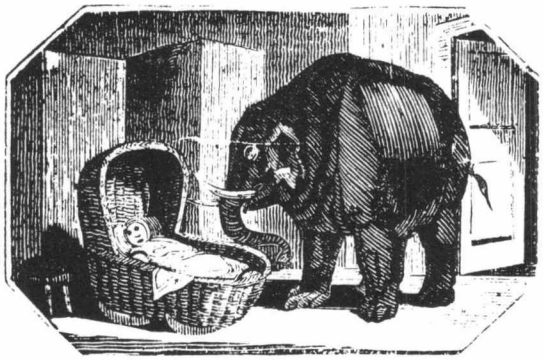
```
ØEB5  DE ØA    BRACHK  LDX    TEMX          Destination computed at $1Ø5C
ØEB7  DF ØE            STX    BRANCH        Destination in $ØE-$ØF
ØEB9  9C 26            CPX    LCOUNT        Object pgm location counter
ØEBB  26 Ø6            BNE    *+$Ø8         Not the same? Skip next.
ØEBD  86 FE    TLOOP   LDA A  #$FE          Code for tight loop (Halt)
ØEBF  97 ØF            STA A  BRANCH + 1    Location for machine offset
ØEC1  2Ø 3Ø            BRA    EXBRCH        Get branch code & exit.
ØEC3  DE 26            LDX    LCOUNT        Get current location
ØEC5  Ø8               INX
ØEC6  Ø8               INX                  +2 = next instruction
ØEC7  DF ØA            STX    TEMX          Ref. for machine-code offset
ØEC9  96 ØA            LDA A  TEMX          MSB of location counter
ØECB  D6 ØB            LDA B  TEMX + 1      LSB
ØECD  DØ ØF            SUB B  BRANCH + 1    Counter - destination

ØECF  Ø1               NOP                  This was error (BEQ EXBR1)
ØED0  Ø1               NOP                  --Allowed $nnØØ offset error

ØED1  92 ØE            SBC A  BRANCH        Counter - Dest (MSB)
ØED3  24 14            BCC    NBRCH         No carry? Negative branch.
ØED5  96 ØE            LDA A  BRANCH        Carry set. Forward branch
ØED7  D6 ØF            LDA B  BRANCH + 1    Get the LSB
ØED9  DØ ØB            SUB B  TEMX + 1      Subtract the other way
ØEDB  92 ØA            SBC A  TEMX          Dest - Location (MSB)
ØEDD  4D               TST A                Should be Ø
ØEDE  26 16            BNE    BRERR         If not, flag the error
ØEE0  C1 8Ø            CMP B  #$8Ø          Check magnitude of offset
ØEE2  2A 12            BPL    BRERR         $8Ø or more, no good.
ØEE4  D7 ØF    EXBR1   STA B  BRANCH + 1    Location for computed offset
ØEE6  C6 E7    EXBRCH  LDA B  #$E7          Code for branch coding
ØEE8  39               RTS                  Exit
ØEE9  4D       NBRCH   TST A                Should be Ø
ØEEA  26 ØA            BNE    BRERR         If not, it's an error
ØEEC  C1 81            CMP B  #$81          Max neg branch is -$8Ø

*The following instruction was "BPL".  Must now be "BCC" to
*handle the "ØØ" branch case formerly caught by "BEQ EXBR1".

ØEEE  24 Ø6            BCC    BRERR         B $81 or higher, bad news.
ØEFØ  5Ø               NEG B                2's complement for neg branch
ØEF1  D7 ØF            STA B  BRANCH + 1    Could be BRA EXBR1 here
ØEF3  C6 E7            LDA B  #$E7          Get code for assembler
ØEF5  39               RTS                  And return.

ØEF6  86 DØ    BRERR   LDA A  #$DØ          Code to generate ERROR 2Ø8 msg.
ØEF8  BD Ø5ED          JSR    ERROR         Run message if Pass 2
ØEFB  2Ø CØ            BRA    TLOOP         Code tight loop & exit.
```

LISTING 2 -- VARIOUS ROUTINES

```
BUFIN --  KEYBOARD INPUT

STRNGC -- STRING OUTPUT WITH CRLF
STRNGO -- STRING OUTPUT
CRLF   -- CARRIAGE RETURN & LINE-FEED

EDITOR -- INITIALIZATION
COMAND -- SOFT RESTART

SAVE   -- TAPE OUTPUT OF TEXT FILE
LOAD   -- TAPE INPUT TO TEXT FILE

CTLSET -- CONTROL-LINE MANIPULATION
PCHON  -- TURN ON TAPE (PUNCH/WRITE)
PCHOFF -- TURN OFF TAPE
RDRON  -- TURN ON TAPE (READ)
RDROFF -- TURN OFF TAPE
PRINTC -- PRINTER DRIVER
OUTQ   -- TERMINAL/PRINTER OUTPUT STEERING


*SWTPC EDITOR/ASSEMBLER ØØFØ-1B32
*December 4, 1978

161A  BD 1A9D  BUFINX  JSR    OUTO          Output ACC A to terminal
161D  CE 1B33  BUFINØ  LDX    #$1B33        Re-entry. Get start of buffer.
162Ø  BD E1AC  BUFIN   JSR    INEEE         Keyboard input, in MIKBUG
1623  81 ØØ            CMP A  #$ØØ          Look for NUL (CTRL/Ø, CTRL/SP)
1625  26 Ø3            BNE    *+$Ø5         No null, no escape.
1627  7E 17BC          JMP    COMAND        Escape to COMAND routine.
162A  81 18            CMP A  #$18          CAN (CTRL/X)?
162C  27 31            BEQ    CANC          CAN will clear buffer
162E  81 ØD            CMP A  #$ØD          RETURN?
163Ø  26 ØA            BNE    *+$ØC         Not RETURN? Keep checking
1632  8C 1B33          CPX    #$1B33        At start of buffer?
1635  26 2F            BNE    TERM          If not, terminate line
1637  BD 168F          JSR    CRLF          1st character. Do a CRLF.
163A  2Ø E1            BRA    BUFINØ        Reload XR and start over.
163C  81 ØA            CMP A  #$ØA          Check for LF
163E  27 EØ            BEQ    BUFIN         If LF, ignore it.
164Ø  81 Ø8            CMP A  #$Ø8          Check for backspace (CTRL/H)
1642  27 Ø6            BEQ    BACKS         If BS, echo another one (!)
1644  81 ØF            CMP A  #$ØF          CTRL/O is alternate backspace
1646  26 ØD            BNE    BUFLD         Not CTRL/O, put into buffer.
1648  86 5F            LDA A  #$5F          Back-arrow to indicate BS
164A  BD 1A9D  BACKS   JSR    OUTO          Output character in A
```

```
164D  8C 1B33          CPX    #$1B33       At start of buffer?
165Ø  27 CE            BEQ    BUFIN        If so, ignore BS
1652  Ø9               DEX                 Step back 1 byte
1653  2Ø CB            BRA    BUFIN        Back for more input.
1655  8C 1B7A  BUFLD   CPX    #$1B7A       At end of buffer?
1658  27 C6            BEQ    BUFIN        If so, ignore input
165A  A7 ØØ            STA A  $ØØ,X        All's well. Data into buffer
165C  Ø8               INX                 Next position
165D  2Ø C1            BRA    BUFIN        Back for more
*
165F  CE 166F  CANC    LDX    #DELTX       "DELETED" text
1662  8D 14            BSR    STRNGC       Run text plus CRLF
1664  2Ø B7            BRA    BUFINØ       Reload XR & start over.
*
1666  86 Ø4    TERM    LDA A  #$Ø4         Get EOT
1668  A7 ØØ            STA A  $ØØ,X        Into next buffer position.
166A  DF ØC            STX    $ØC          Save buffer position on Page Ø
166C  8D 21            BSR    CRLF         Do a CRLF
166E  39               RTS                 And exit.

*
*STRNGC: Output character string per XR, then CRLF
1678  8D ØE    STRNGC  BSR    STRNGO       Run character string up to EOT
167A  2Ø 13            BRA    CRLF         Do a CRLF and then RTS.
*
167C  BD 1A9D  STRNG1  JSR    OUTQ         ACC A out via flagged device
167F  B6 8ØØ4          LDA A  $8ØØ4        Check for keyboard interrupt
1682  2B Ø3            BMI    *+$Ø5        No interrupt? Skip next.
1684  7E 17BC          JMP    COMAND       Any KB input, halt output.
1687  Ø8               INX                 XR to next character
1688  A6 ØØ    STRNGO  LDA A  $ØØ,X        This is normal entry point.
168A  81 Ø4            CMP A  #$Ø4         Check for EOT
168C  26 EE            BNE    STRNG1       If not EOT, output character
168E  39               RTS                 Was EOT. Exit.
*
168F  FF 16A3  CRLF    STX    SAVEX        Save XR
1692  CE 169B          LDX    #CRLFTX      Get CR LF NAK string
1695  8D F1            BSR    STRNGO       Run it
1697  FE 16A3          LDX    SAVEX        Recover XR
169A  39               RTS                 Scram
17A2  CE 8Ø1C  EDITOR  LDX    #$8Ø1C       Address of printer port PIA
17A5  C6 FF            LDA B  #$FF         Handful of bits
17A7  E7 ØØ            STA B  $ØØ,X        Set 8 output lines, A side
17A9  C6 3F            LDA B  #$3F         Handshake & lock-em-up
17AB  E7 Ø1            STA B  $Ø1,X        Control register of PIA
17AD  DE FE            LDX    FILAD        Start of text file
17AF  DF FA            STX    FILEND       Next available file loc.
17B1  DF FC            STX    FILOC        Working location
17B3  BD 16C8          JSR    ZSET         Set $F1-$F5 to $3Ø (Ø)
17B6  DE F1            LDX    $F1          Get $3Ø3Ø
17B8  DF F5            STX    $F5          Into $F5-F6
17BA  DF F7            STX    $F7          Line number now ØØØØØØ (!)
*
17BC  C6 FF    COMAND  LDA B  #$FF         Code for OUTO output to terminal
17BE  F7 Ø1C8          STA B  PFLAG        Steering for OUTO
17C1  8E AØ47  COMND1  LDS    #$AØ47       Set SP back 2 from normal.
17C4  86 Ø4            LDA A  #$Ø4         EOT
17C6  B7 169E          STA A  $169E        Shorten CRLF string
17C9  7F ØØF9          CLR    AUTOF        $F9 flags AUTO numbering mode
17CC  CE 15BB          LDX    #READYT      "READY" text
17CF  BD 1678          JSR    STRNGC       Run text and CRLF
17D2  CE 1B33  COMND2  LDX    #$1B33       1st position in buffer
17D5  96 F9            LDA A  AUTOF        Check AUTO flag
17D7  27 1Ø            BEQ    COMND3       AUTOF Ø, skip next
17D9  BD 171D          JSR    ASCINC       Routine steps ASCII line # by 1Ø
17DC  BD 178C          JSR    MOVLNO       Transfer line # to buffer
17DF  86 Ø4            LDA A  #$Ø4         Get an EOT
17E1  A7 ØØ            STA A  $ØØ,X        Into buffer after line #
17E3  CE 1B33          LDX    #$1B33       Set XR to start of buffer
17E6  BD 1688          JSR    STRNGO       Output line # to terminal
17E9  BD 162Ø  COMND3  JSR    BUFIN        Get input
17EC  CE 1B33          LDX    #$1B33       Start of buffer into XR
17EF  BD 16BE          JSR    SPSKIP       Skip over any spaces
17F2  A6 ØØ            LDA A  $ØØ,X        Get non-space character
17F4  BD 16B2          JSR    NUMCHK       Check for a number
17F7  25 Ø3            BCS    *+$Ø5        C set? Not a number.
17F9  7E 18CA          JMP    TEXTIN       Was a number -- text input.
17FC  A6 ØØ            LDA A  $ØØ,X        Not a number. What was it?
17FE  81 Ø4            CMP A  #$Ø4         An EOT?
18ØØ  27 DØ            BEQ    COMND0       If EOT, start over.
1802  Ø8               INX                 Probably a letter. Step to next.
1803  E6 ØØ            LDA B  $ØØ,X        Get next character.
1805  Ø8               INX
1806  DF ØA            STX    $ØA          Save position in buffer.
1808  CE 15EØ          LDX    #COMTBL      Start of command table
180B  A1 ØØ    NEXCOM  CMP A  $ØØ,X        Check against command word
180D  26 Ø8            BNE    STEPC        No match? Step to next command.
180F  E1 Ø1            CMP B  $Ø1,X        Check 2nd character
1811  26 Ø4            BNE    STEPC        No match? Step to next.
1813  EE Ø2            LDX    $Ø2,X        Match. Get routine address.
1815  6E ØØ            JMP    $ØØ,X        Perform routine.
1817  Ø8       STEPC   INX                 4 bytes for each command
1818  Ø8               INX
1819  Ø8               INX
181A  Ø8               INX
181B  8C 1618          CPX    #$1618       At end of command table?
181E  26 EB            BNE    NEXCOM       If not, check next command.
182Ø  CE 15C5          LDX    #WHATX       "WHAT??" text
1823  BD 1678          JSR    STRNGC       Run text plus CRLF
1826  2Ø AA            BRA    COMND2       Get fresh input.
*
1983  BD 1A3A  SAVE    JSR    PCHON        Turn on punch, set control lines
1986  C6 4Ø            LDA B  #$4Ø         6.4" of leader if paper-tape
1988  86 FF            LDA A  #$FF         All bits (NG for paper-tape)
198A  BD 1A9D          JSR    OUTO         Run leader character
198D  5A               DEC B               Character-count
198E  26 F8            BNE    *-$Ø6        Loop until done
1990  86 Ø2            LDA A  #$Ø2         STX: Here come de data!
1992  BD 1A9D          JSR    OUTQ         Output the STX
1995  4F               CLR A               NUL
1996  B7 169E          STA A  $169E        Extends CRLF string with NULs
1999  DE FE            LDX    FILAD        Start of file
199B  DF 48            STX    FROM         Start point for FROMTO
199D  DE FA            LDX    FILEND       End of file + 1
199F  DF 4A            STX    TO           Termination for FROMTO
19A1  BD 16A5          JSR    FROMTO       Output everything
19A4  C6 2Ø            LDA B  #$2Ø         Shorter trailer-tape
19A6  86 Ø3            LDA A  #$Ø3         ETX -- end of tape.
19A8  BD 1A9D          JSR    OUTQ         Output the ETX
19AB  86 FF            LDA A  #$FF         Leader. Use ØØ for paper tape.
19AD  BD 1A9D          JSR    OUTQ         Output character
19BØ  5A               DEC B               Count
19B1  26 F8            BNE    *-$Ø6        Loop until done
19B3  BD 1A48          JSR    PCHOFF       Turn off tape, restore ctl lines
19B6  7E 17BC          JMP    COMAND       Back to command mode.
*
*
19B9  DE FE    LOAD    LDX    FILAD        Start of file
19BB  BD 1A59          JSR    RDRON        Turn on reader, set ctl. lines
19BE  BD E1AC          JSR    INEEE        Input from tape
19C1  81 Ø2            CMP A  #$Ø2         Watch for STX
19C3  26 F9            BNE    *-$Ø5        Loop 'til you get it.
19C5  BD E1AC  LOAD1   JSR    INEEE        Get more input
19C8  81 Ø3            CMP A  #$Ø3         Watch for ETX
19CA  27 1B            BEQ    EXLOAD       ETX signals end of tape
19CC  81 ØD            CMP A  #$ØD         Watch for RETURN code in CRLF
19CE  26 Ø2            BNE    *+$Ø4        Not RETURN? Skip next
19DØ  86 Ø4            LDA A  #$Ø4         RETURN.  Substitute EOT in file.
19D2  81 ØA            CMP A  #$ØA         LF code?
19D4  27 EF            BEQ    LOAD1        Ignore it.
19D6  81 ØØ            CMP A  #$ØØ         NUL?
19D8  27 EB            BEQ    LOAD1        Ignore it.
19DA  81 15            CMP A  #$15         NAK? (Part of CRLF string)
19DC  27 E7            BEQ    LOAD1        Ignore NAK
19DE  81 12            CMP A  #$12         DC2 (=Punch On)?
19EØ  27 E3            BEQ    LOAD1        If DC2, ignore it.
19E2  A7 ØØ            STA A  $ØØ,X        Anything else, into the file
19E4  Ø8               INX                 Next file location
19E5  2Ø DE            BRA    LOAD1        Get more input
19E7  BD 1A69  EXLOAD  JSR    RDROFF       Turn off reader, reset ctl lines
19EA  DF FA            STX    FILEND       New "next" location
19EC  CE 3939          LDX    #$3939       Get "99" (ASCII)
19EF  DF F5            STX    $F5          Into Line # register
19F1  DF F7            STX    $F7          Line # now 9999
19F3  7E 17BC          JMP    COMAND       All done.  Back to command mode
*
*
1A12  FF Ø1CC  CTLSET  STX    SAVEXØ       Enter with on/off code in ACC B
1A15  CE 8ØØ4          LDX    #$8ØØ4       Control port PIA address
1A18  86 Ø2            LDA A  #$Ø2         Playing games with bits
1A1A  CA Ø1            ORA B  #$Ø1         Add "1" bit to incoming code
1A1C  A7 Ø1            STA A  $Ø1,X        Into control register
1A1E  E7 ØØ            STA B  $ØØ,X        Setting data lines
1A2Ø  86 Ø6            LDA A  #$Ø6         Add lockup code ("4" bit)
1A22  A7 Ø1            STA A  $Ø1,X        Control register
1A24  E7 ØØ            STA B  $ØØ,X        Outputs code
1A26  86 Ø2            LDA A  #$Ø2         Here we go again
1A28  C6 Ø1            LDA B  #$Ø1         Just the "1" bit this time
1A2A  E7 ØØ            STA B  $ØØ,X        Output it via data line
1A2C  A7 Ø1            STA A  $Ø1,X        Opens up control reg. again
1A2E  E7 ØØ            STA B  $ØØ,X        New outputs pattern
1A3Ø  86 Ø6            LDA A  #$Ø6         Lock-em-up code
1A32  A7 Ø1            STA A  $Ø1,X        Control register
1A34  E7 ØØ            STA B  $ØØ,X        Data register
1A36  FE Ø1CC          LDX    SAVEXØ       Recover XR stored at start
1A39  39               RTS                 Return.
*
1A3A  86 12    PCHON   LDA A  #$12         DC2 -- punch on code
1A3C  8D 3B            BSR    OUTEP        Pointer to OUTEEE
1A3E  37               PSH B               Save ol' B
1A3F  C6 Ø4            LDA B  #$Ø4         Special code for CTLSET
1A41  8D CF            BSR    CTLSET       Set up control lines
1A43  33               PUL B               Recover ACC B
1A44  BD Ø6Ø5          JSR    DELAY        Wait for machine to get going
1A47  39               RTS                 Return with ACC A = 2
*
1A48  86 14    PCHOFF  LDA A  #$14         DC4 = punch offcode
1A4A  8D 2D            BSR    OUTEP        OUTEEE pointer
1A4C  37               PSH B
1A4D  C6 Ø8            LDA B  #$Ø8         Pattern for CTLSET
1A4F  8D C1            BSR    CTLSET       Woggle the control lines
1A51  33               PUL B
1A52  39               RTS                 That's all.
*
1A53  BD Ø257  ASSEMB  JSR    ASMSET       Modify operands for ASSEMBLE
1A56  7E Ø3ØØ          JMP    ASMPTR       Do the assembly thing
*
1A59  86 11    RDRON   LDA A  #$11         DC1 = reader-on code
1A5B  8D 1C            BSR    OUTEP        Output the code
1A5D  86 3C            LDA A  #$3C         Control register code
1A5F  B7 8ØØ7          STA A  $8ØØ7        "B" side control register
1A62  37               PSH B
1A63  C6 2Ø            LDA B  #$2Ø         Pattern for CTLSET
1A65  8D AB            BSR    CTLSET       Set up for tape input
1A67  33               PUL B
```

```
1A68   39              RTS             That's all.
*
1A69   86 13   RDROFF  LDA A  #$13     DC3 = reader-off code
1A6B   8D ØC           BSR     OUTEP   Output it
1A6D   86 34           LDA A  #$34     Code for normal PIA setting
1A6F   B7 8ØØ7         STA A  $8ØØ7    Set "B" side control register
1A72   37              PSH B
1A73   C6 1Ø           LDA B  #$1Ø     Pattern for CTLSET
1A75   8D 9B           BSR     CTLSET  Reset control lines
1A77   33              PUL B
1A78   39              RTS             The end.
*
1A79   7E E1D1 OUTEP   JMP     OUTEEE  Pointer to MIKBUG output routine
*
1A7C   7C ØØ1F TPASS   INC     $ØØ1F   Set up for 2T pass
1A7F   BD 1A3A         JSR     PCHON   Turn on punch (recorder)
1A82   39              RTS             That's enough.
*
1A83   FF Ø1CA PRINTC  STX     SAVEX1  Save XR
1A86   CE 8Ø1C         LDX     #$8Ø1C  Printer port PIA address
1A89   A7 ØØ           STA A  $ØØ,X    Char for output was in A
1A8B   C6 37           LDA B  #$37     Set CA-2, look for -edge
1A8D   E7 Ø1           STA B  $Ø1,X    Control register
1A8F   C6 3F           LDA B  #$3F     Release CA-2
1A91   E7 Ø1           STA B  $Ø1,X
1A93   6D Ø1           TST     $Ø1,X   Wait for DAC from printer
1A95   2A FC           BPL     *-$Ø2   Loop until you get it.
1A97   E6 ØØ           LDA B  $ØØ,X    Got it. Clear IRQ.
1A99   FE Ø1CA         LDX     SAVEX1  Recover XR
1A9C   39              RTS             Exit.
*
1A9D   37       OUTQ   PSH B           Save B (not necessary)
1A9E   F6 Ø1C8         LDA B  PFLAG    Steering flag
1AA1   27 Ø4           BEQ     *+$Ø6   ØØ means printer output
1AA3   33              PUL B           Recover ACC B
1AA4   7E E1D1         JMP     OUTEEE  $FF means terminal output.
1AA7   33              PUL B           Recover ACC B
1AA8   7E 1A83         JMP     PRINTC  Printer output routine.
*
1AAB   7F Ø1C8 TERMO   CLR     PFLAG   Clear printer flag.
1AAE   73 Ø1C8         COM     PFLAG   Set to $FF = terminal output
1AB1   7E 17BC         JMP     COMAND  COMAND does this same thing (!)
*
*
*
*Patch to OBUFLD to eliminate double C/R on line with comments
*
Ø7E4   BD Ø5DE         JSR     $Ø5DE   Get character from line buffer
Ø7E7   81 ØD           CMP A  #$ØD     Check for C/R first
Ø7E9   27 Ø8           BEQ     CRLD    If C/R, terminate line
Ø7EB   A7 ØØ           STA A  $ØØ,X    Not C/R. Load into buffer.
Ø7ED   Ø8              INX             Next position.
Ø7EE   8C ØØCA         CPX     #$ØØCA  At end of output buffer?
Ø7F1   26 F1           BNE     *-$ØD   If not, get more input
Ø7F3   86 ØD    CRLD   LDA A  #$ØD     End of line. Get C/R.
Ø7F5   A7 Ø1           STA A  $Ø1,X    Into next position.
*
```



Tiny-C Interpreter On CDOS

BY RAY DUNCAN
Laboratory Microsystems
710 E. Gladstone
La Verne, CA 91750

Dear *Dr. Dobbs*:

As another purchaser of the Tiny-c Owners Manual, I would like to confirm all the favorable comments made by Ted Shapin in *DDJ #32*. For clarity and completeness this documentation is unparalleled in the personal computer field. We hand translated the Intel mnemonics into Zilog mnemonics, reassembled the whole thing and got the interpreter running without any trouble. We have written a set of interface routines to let the Tiny-c Interpreter run under Cromemco CDOS (virtually identical to CP-M) with full disc file capabilities. The source listing is enclosed. Of course, readers must obtain the code for the Interpreter from Tiny-c Associates.

Sincerely,
Ray Duncan

(The address for tiny c associates is P.O. Box 269, Holmdel, N.J. 07733)

```
0002 ;
0003 ;
0004 ; CSMC  NEOMATAL ICU
0005 ; VERSION 2.0 18 JAN 1979
0006 ; VERSION 1.0 7 DEC 1978
0007 ;
0008 TSTAT   EQU   0
0009 TDATA   EQU   1
0010 TBE     EQU   80H
0011 RDA     EQU   40H
0012 NULL    EQU   0
0013 EOF     EQU   1AH
0014 ;
0015 ;
0016 ;
0017 FCB1    DS    33
0018 DPTR1   DB    0
0019 DSTAT1  DB    0
0020 DBUFF1  DS    128
0021 ;
0022 FCB2    DS    33
0023 DPTR2   DB    0
0024 DSTAT2  DB    0
0025 DBUFF2  DS    128
0026 ;
0027 FCB3    DS    33
0028 DPTR3   DB    0
0029 DSTAT3  DB    0
0030 DBUFF3  DS    128
0031 ;
0032 FCB4    DS    33
0033 DPTR4   DB    0
0034 DSTAT4  DB    0
0035 DBUFF4  DS    128
0036 ;
0037 ;
0038 UINCH   LD    A,(UCHRDA)      ;RETURNS A=ASCII CHAR
0039         OR    A
0040         JR    NZ,UINCH2
0041         CALL  UCHRDY
0042         JR    UINCH
0043 UINCH2  PUSH  AF
0044         XOR   A
0045         LD    (UCHRDA),A
0046         POP   AF
0047         RET
0048 ;
0049 ;
0050 ;
0051 UCHRDY                        ;TEST IF NEW CHAR READY
0052                               ;NO CHAR READY->A=0, Z=TRUE
0053                               ;NON-NULL READY=>A=CHAR, Z=F
0054                               ;NULL READY=>A=1, Z=FALSE
0055         IN    A,TSTAT
0056         AND   RDA
0057         RET   Z
0058         IN    A,TDATA
0059         CP    NULL
0060         JR    NZ,UCHRD2
0061         LD    A,1
0062 UCHRD2  AND   7FH
0063         LD    (UCHRDA),A
0064         OR    A
0065         RET
0066 UCHRDA  DB    0               ;SAVE INPUT CHAR
0067 ;
0068 ;
0069 UOUTCH                        ;CALL A=ASCII CHAR
0070                               ;A,DE,HL PRESERVED
0071         PUSH  AF
0072         IN    A,TSTAT
0073         AND   TBE
0074         JR    Z,UOUTCH+1
0075         POP   AF
```

```
13AA  D301      0076          OUT     TDATA,A
13AC  FE0D      0077          CP      0DH     ;IF CR SEND ALSO LF
13AE  C0        0078          RET     NZ
13AF  3E0A      0079          LD      A,0AH
13B1  CDA213    0080          CALL    UOUTCH
13B4  3E0D      0081          LD      A,0DH
13B6  C9        0082          RET
                0083 ;
                0084 ;
                0085 ;
                0086 UFOPEN                   ;CALL HL=FILE NAME ADDR
                0087                          ;     DE=FILE SIZE,BYTES
                0088                          ;     BC=UNIT
                0089                          ;     A=1 RD =2 WRITE
                0090                          ;RETURNS A=0 Z=TRUE IF
                0091                          ;     SUCCESSFUL OPEN
                0092                          ;RETURNS A=-1 Z=F IF
                0093                          ; UNABLE TO OPEN FILE
                0094                          ;RETURNS A>0 Z=F IF
                0095                          ; FILE ALREADY OPEN
13B7  C5        0096          PUSH    BC
13B8  D5        0097          PUSH    DE
13B9  E5        0098          PUSH    HL
13BA  320614    0099          LD      (UFOPRW),A ;SAVE R/W CODE
13BD  CD9614    0100          CALL    UCFCB   ;DE<-FCB ADDR
13C0  DD7E01    0101          LD      A,(IX+1) ;MAKE SURE FILE IS
13C3  B7        0102          OR      A       ;NOT ALREADY OPEN
13C4  203C      0103          JR      NZ,UFOP9 ;EXIT,ERROR
13C6  D5        0104          PUSH    DE      ;SAVE IT
13C7  01A300    0105          LD      BC,FCB2-FCB1
13CA  AF        0106 UFOP2    XOR     A       ;CLEAR FCB AND BUFFER
13CB  12        0107          LD      (DE),A
13CC  13        0108          INC     DE
13CD  0B        0109          DEC     BC
13CE  78        0110          LD      A,B
13CF  B1        0111          OR      C
13D0  20F8      0112          JR      NZ,UFOP2
13D2  D1        0113          POP     DE      ;RESTORE DE=FCB FBA
13D3  D5        0114          PUSH    DE
13D4  0E86      0115          LD      C,134   ;FORMAT FILE NAME
13D6  CD2B15    0116          CALL    CDOS
13D9  D1        0117 UFOP3    POP     DE      ;DE<-FCB ADDR
13DA  D5        0118          PUSH    DE
13DB  0E0F      0119          LD      C,15    ;OPEN DISC FILE
13DD  CD2B15    0120          CALL    CDOS
13E0  D1        0121          POP     DE      ;DE<-FCB ADDR
13E1  FEFF      0122          CP      0FFH    ;WAS OPEN SUCCESSFUL
13E3  2014      0123          JR      NZ,UFOP8 ;YES
                0124                          ;OPEN NOT SUCCESSFUL
13E5  3A0614    0125          LD      A,(UFOPRW) ;WAS THIS CALL FOR WRITE
13E8  FE02      0126          CP      2
13EA  2805      0127          JR      Z,UFOP7 ;YES
13EC  3EFF      0128          LD      A,0FFH  ;READ CALL,EXIT WITH ERROR
13EE  B7        0129          OR      A
13EF  1811      0130          JR      UFOP9
                0131 UFOP7                    ;DE<-FCB ADDR
13F1  D5        0132          PUSH    DE
13F2  0E16      0133          LD      C,22    ;CREATE NEW FILE
13F4  CD2B15    0134          CALL    CDOS
13F7  18E0      0135          JR      UFOP3   ;NOW GO OPEN IT
13F9  212200    0136 UFOP8    LD      HL,DSTAT1-FCB1
13FC  19        0137          ADD     HL,DE
13FD  3A0614    0138          LD      A,(UFOPRW) ;SET R/W CODE IN TABLE
1400  77        0139          LD      (HL),A
1401  AF        0140          XOR     A       ;A=0 Z=TRUE
1402  E1        0141 UFOP9    POP     HL
1403  D1        0142          POP     DE
1404  C1        0143          POP     BC
1405  C9        0144          RET
1406  00        0145 UFOPRW   DB      0       ;1=RD 2=WRITE
                0146 ;
                0147 ;
                0148 ;
                0149 UFREAD                   ;CALL HL=FBA I/O BUFFER
                0150                          ;     BC=UNIT
                0151                          ;1 RECORD IS READ 1-256 BYTES
                0152                          ;RETURNS A=0 Z=TRUE DE=# BYTES
                0153                          ;         A=-1 Z=F -> EOF
                0154                          ;         A>0 Z=F -> ERROR
                0155                          ;         HL PRESERVED
1407  E5        0156          PUSH    HL      ;SAVE FBA
1408  CD9614    0157          CALL    UCFCB   ;DE<-FCB ADDR
140B  DD7E01    0158          LD      A,(IX+1) ;IS FILE OPENED FOR READ
140E  FE01      0159          CP      1
1410  201F      0160          JR      NZ,UFRD9 ;NO,EXIT WITH ERROR FLAG
1412  CDB414    0161          CALL    RBYTE   ;READ BYTE COUNT INTO C
1415  48        0162          LD      C,B
1416  0600      0163          LD      B,0
1418  C5        0164          PUSH    BC      ;SAVE BYTE COUNT
1419  CDB414    0165 UFRD2    CALL    RBYTE   ;READ RECORD
141C  78        0166          LD      A,B
141D  FE1A      0167          CP      EOF     ;EXIT IF END OF FILE
141F  2809      0168          JR      Z,UFRD6
1421  70        0169          LD      (HL),B  ;STORE THIS BYTE
1422  79        0170          LD      A,C     ;DONE YET?
1423  B7        0171          OR      A
1424  2807      0172          JR      Z,UFRD8 ;YES,EXIT A=0 Z=T
1426  23        0173          INC     HL      ;BUMP BUFFER POINTER
1427  0D        0174          DEC     C       ;DECREMENT BYTE COUNT
1428  18EF      0175          JR      UFRD2
142A  3EFF      0176 UFRD6    LD      A,0FFH  ;EXIT,EOF
142C  B7        0177          OR      A
142D  D1        0178 UFRD8    POP     DE      ;FETCH BYTE COUNT
142E  13        0179          INC     DE
142F  E1        0180          POP     HL      ;RESTORE HL,AND EXIT
1430  C9        0181          RET
1431  3E01      0182 UFRD9    LD      A,1     ;EXIT,FILE NOT OPEN FOR RD
1433  B7        0183          OR      A
1434  E1        0184          POP     HL
1435  C9        0185          RET
                0186 ;
                0187 ;
                0188 ;
                0189 UFWRITE                  ;CALL HL=FBA
                0190                          ;     DE=LBA
                0191                          ;     BC=UNIT
                0192                          ;RETURN A=0 SUCCESS
                0193                          ;       A<>0 ERROR
1436  E5        0194          PUSH    HL      ;SAVE FBA
1437  7C        0195          LD      A,H     ;HL<-(-HL)
1438  2F        0196          CPL
1439  67        0197          LD      H,A
143A  7D        0198          LD      A,L
143B  2F        0199          CPL
143C  6F        0200          LD      L,A
143D  23        0201          INC     HL
143E  19        0202          ADD     HL,DE   ;GET BYTE COUNT(LBA-FBA)
143F  E5        0203          PUSH    HL      ;SAVE IT
1440  CD9614    0204          CALL    UCFCB   ;DE<-FCB ADDR
1443  C1        0205          POP     BC      ;C<-BYTE COUNT
1444  E1        0206          POP     HL      ;HL<-FBA
1445  DD7E01    0207          LD      A,(IX+1) ;FILE OPENED FOR WRITE?
1448  FE02      0208          CP      2
144A  2010      0209          JR      NZ,UFWR8 ;NO,EXIT
144C  41        0210          LD      B,C     ;WRITE BLOCK LENGTH
144D  CDF014    0211          CALL    WBYTE
1450  46        0212 UFWR2    LD      B,(HL)  ;WRITE RECORD
1451  CDF014    0213          CALL    WBYTE
1454  79        0214          LD      A,C     ;END OF RECORD?
1455  B7        0215          OR      A
1456  2808      0216          JR      Z,UFWR9
1458  23        0217          INC     HL
1459  0D        0218          DEC     C
145A  18F4      0219          JR      UFWR2
145C  3EFF      0220 UFWR8    LD      A,0FFH  ;EXIT WITH ERROR FLAG
145E  B7        0221          OR      A       ;FILE NOT OPEN FOR WRITE
145F  C9        0222          RET
1460  AF        0223 UFWR9    XOR     A       ;A=0 SUCCESSFUL
```

```
1461  C9            0224          RET
                    0225 ;
                    0226 ;
                    0227 ;
                    0228 UFCLOSE                      ;CALL BC=UNIT
1462  E5            0229          PUSH    HL
1463  D5            0230          PUSH    DE
1464  C5            0231          PUSH    BC
1465  CD9614        0232          CALL    UCFCB   ;DE<-FCB ADDR
1468  D5            0233          PUSH    DE      ;SAVE IT
1469  DD7E01        0234          LD      A,(IX+1) ;WAS THIS OPENED FOR WRT
146C  FE02          0235          CP      2
146E  2018          0236          JR      NZ,UFCLS4 ;NO, JUMP
1470  0600          0237          LD      B,0     ;WRITE RECORD LENGTH
1472  CDF014        0238          CALL    WBYTE
1475  061A          0239          LD      B,EOF   ;WRITE EOF FLAG
1477  CDF014        0240          CALL    WBYTE
147A  DD7E00        0241 UFCLS2   LD      A,(IX)  ;CHECK BUFFER POINTER
147D  FE00          0242          CP      0       ;IS BUFFER EMPTY?
147F  2807          0243          JR      Z,UFCLS4 ;YES
1481  0600          0244          LD      B,0     ;NO, FORCE BUFFER WRITE
1483  CDF014        0245          CALL    WBYTE
1486  18F2          0246          JR      UFCLS2
1488  DD360100      0247 UFCLS4   LD      (IX+1),0 ;SET TABLE IDLE
148C  D1            0248          POP     DE      ;CLOSE FILE
148D  0E10          0249          LD      C,16
148F  CD2B15        0250          CALL    CDOS
1492  C1            0251          POP     BC
1493  D1            0252          POP     DE
1494  E1            0253          POP     HL
1495  C9            0254          RET
                    0255 ;
                    0256 ;
                    0257 ;
                    0258 UCFCB                        ;CALC FCB ADDR
                    0259                              ;CALL BC=UNIT
                    0260                              ;RETURN DE=FCB FBA
                    0261                              ;       IX=DPTR ADDR
                    0262                              ;       IY=DBUFF ADDR
1496  C5            0263          PUSH    BC
1497  E5            0264          PUSH    HL
1498  21EF10        0265          LD      HL,FCB1
149B  11A300        0266          LD      DE,FCB2-FCB1
149E  0D            0267 UCFCB2   DEC     C
149F  2803          0268          JR      Z,UCFCB9
14A1  19            0269          ADD     HL,DE
14A2  18FA          0270          JR      UCFCB2
14A4  EB            0271 UCFCB9   EX      DE,HL   ;DE<-FCB ADDR
14A5  DD212100      0272          LD      IX,DPTR1-FCB1
14A9  FD212300      0273          LD      IY,DBUFF1-FCB1
14AD  DD19          0274          ADD     IX,DE
14AF  FD19          0275          ADD     IY,DE
14B1  E1            0276          POP     HL      ;RESTORE HL AND BC
14B2  C1            0277          POP     BC
14B3  C9            0278          RET
                    0279 ;
                    0280 ;
                    0281 ;
                    0282 RBYTE                        ;CALL DE=FCB ADDR
                    0283                              ; RETURN B=DATA
                    0284                              ;C,HL,DE PRESERVED
14B4  DDE5          0285          PUSH    IX      ;SAVE REGISTERS
14B6  FDE5          0286          PUSH    IY
14B8  C5            0287          PUSH    BC
14B9  E5            0288          PUSH    HL
14BA  D5            0289          PUSH    DE
14BB  DD7E00        0290          LD      A,(IX)  ;IS DISC ACCESS NEEDED
14BE  FE00          0291          CP      0
14C0  200F          0292          JR      NZ,RBYTE2 ;NO, JUMP
14C2  FDE5          0293          PUSH    IY
14C4  D1            0294          POP     DE      ;DE<- I/O BUFFER ADDR
14C5  0E1A          0295          LD      C,26
14C7  CD2B15        0296          CALL    CDOS    ;SET DISC DMA
14CA  D1            0297          POP     DE      ;DE<- FCB ADDR
```

```
14CB  D5            0298          PUSH    DE
14CC  0E14          0299          LD      C,20    ;READ BLOCK
14CE  CD2B15        0300          CALL    CDOS
14D1  DD4E00        0301 RBYTE2   LD      C,(IX)  ;FETCH BUFFER POINTER
14D4  DD7E00        0302          LD      A,(IX)  ;AND BUMP IT
14D7  3C            0303          INC     A
14D8  FE80          0304          CP      128     ;END OF BUFFER?
14DA  2001          0305          JR      NZ,RBYTE3 ;NO
14DC  AF            0306          XOR     A       ;YES, INITIALIZE PTR
14DD  DD7700        0307 RBYTE3   LD      (IX),A  ;STORE UPDATED POINTER
14E0  0600          0308          LD      B,0     ;ADD OFFSET TO BASE
14E2  FD09          0309          ADD     IY,BC   ;BUFFER ADDRESS
14E4  FD7E00        0310          LD      A,(IY)  ;FETCH DATA BYTE
14E7  D1            0311          POP     DE      ;RESTORE REGISTERS
14E8  E1            0312          POP     HL
14E9  C1            0313          POP     BC
14EA  FDE1          0314          POP     IY
14EC  DDE1          0315          POP     IX
14EE  47            0316          LD      B,A     ;B <- DATA
14EF  C9            0317          RET
                    0318 ;
                    0319 ;
                    0320 ;
                    0321 WBYTE                        ;CALL DE=FCB ADDR
                    0322                              ;      B=DATA
                    0323                              ;RETURN BC,DE,HL PRESERVED
14F0  DDE5          0324          PUSH    IX      ;SAVE REGISTERS
14F2  FDE5          0325          PUSH    IY
14F4  E5            0326          PUSH    HL
14F5  C5            0327          PUSH    BC
14F6  D5            0328          PUSH    DE
14F7  DD5E00        0329          LD      E,(IX)  ;FETCH BUFFER PTR
14FA  1600          0330          LD      D,0
14FC  FDE5          0331          PUSH    IY
14FE  FD19          0332          ADD     IY,DE   ;FORM ACTUAL BUFFER ADDR
1500  FD7000        0333          LD      (IY),B  ;STORE BYTE
1503  FDE1          0334          POP     IY
1505  DD7E00        0335          LD      A,(IX)  ;BUMP POINTER
1508  3C            0336          INC     A
1509  DD7700        0337          LD      (IX),A
150C  FE80          0338          CP      128     ;IS WRITE NEEDED?
150E  2013          0339          JR      NZ,WBYTE9 ;NO, JUMP
1510  DD360000      0340          LD      (IX),0  ;INITIALIZE POINTER
1514  FDE5          0341          PUSH    IY
1516  D1            0342          POP     DE      ;DE <- BUFFER ADDR
1517  0E1A          0343          LD      C,26
1519  CD2B15        0344          CALL    CDOS    ;SET DISC DMA
151C  D1            0345          POP     DE      ;DE <- FCB ADDR
151D  D5            0346          PUSH    DE
151E  0E15          0347          LD      C,21
1520  CD2B15        0348          CALL    CDOS    ;WRITE BLOCK
1523  D1            0349 WBYTE9   POP     DE      ;RESTORE REGISTERS,EXIT
1524  C1            0350          POP     BC
1525  E1            0351          POP     HL
1526  FDE1          0352          POP     IY
1528  DDE1          0353          POP     IX
152A  C9            0354          RET
                    0355 ;
                    0356 ;
                    0357 ;
152B  DDE5          0358 CDOS     PUSH    IX
152D  FDE5          0359          PUSH    IY
152F  D5            0360          PUSH    DE
1530  CD0500        0361          CALL    5
1533  D1            0362          POP     DE
1534  FDE1          0363          POP     IY
1536  DDE1          0364          POP     IX
1538  C9            0365          RET
                    0366 ;
                    0367 ;
1539  (0100)        0368          END     100H

ERRORS              0
```

ODT User's Manual

BY GARY D. GAUGLER
2276 Beaver Valley Rd.
Fairborn, OH 45324

ODT (on-line debugging tool) is a monitor level or stand-alone, line-at-a-time assembler for the MC6800 microprocessor. ODT allows the user to rapidly enter patches or small programs directly into memory using normal mnemonics and operands. The only limitations are that arguments must be numeric (no labels) and all direct versus extended decisions resolve to extended.

The on-line assembler processes each line of mnemonic information into opcode, operand, and mode, and stores this information directly into memory at a user selected location. As the data is stored, the pseudo PC is incremented accordingly to be ready to store new data at the next available memory location. The user may change the memory origin at any time to permit rapid entry of non-continuous patches.

By using an efficient mnemonic look-up routine, the assembler can be contained in only 1K. This permits it to be stored in EPROM as part of the monitor, for readily available use. Some external overhead must be provided by the monitor to support input/output character processing and also for RAM storage.

ODT is called by typing 'ODT, addr(c/r)' where 'addr' is the address in RAM where the assembly is to begin. This is equivalent to the ORG assembler directive. The memory location may be moved at any time (before typing mnemonics) by entering a slash (/) followed by the new target address.

ODT accepts all of the Motorola mnemonics and recognizes the following conventions for operands (arguments):

TYPE	PREFIX
HEX	$
OCTAL	@
BINARY	%
DECIMAL	none

When entering mnemonics which have no arguments (inherent mode) merely enter the mnemonic followed by a carriage return.

EXAMPLE:

```
TPA
SEC
CLI
```

If it is a valid mnemonic, ODT will store the machine code for the mnemonic at the next available memory location, increment the memory pointer (and display it) in preparation for the next instruction, and then wait for input from the user. When an illegal mnemonic is entered, ODT responds with 'ILL MNEMONIC' indicating that the mnemonic was not recognized as valid. Re-enter the mnemonic; the memory pointer will not be moved until valid input is received.

For mnemonics and modes which have two or three bytes, enter the mnemonic, followed by a space, followed by the argument (always terminated by a carriage return).

EXAMPLE:

```
CLR    $7000
TST    0,X
BITA   #%10010
CMPB   #'M
```

Note that the mode will never be direct. A two byte instruction will either be immediate, relative, or indexed; always three bytes for extended. When an argument of greater than 8 bits (1 byte) is entered for a two byte opcode, the least significant 8 bits will be used.

EXAMPLE:

```
BITA   #$1234
```

would be assembled as 85 34

```
CLR    $1A72,X
```

would be assembled as 6F 72

LITERALS

Literal arguments may be entered, as with other assemblers, by preceeding the ASCII character with an apostrophe.

EXAMPLE:

```
LDAA   #'M
CMPB   #'$
```

CONSTANT DATA

Constant data areas for tables, text, or similar purposes may be entered using the FCC or FCB directive. The FCC or FCB directive does not generate code itself; it merely prepares the assembler to accept the data which follows and then place it into memory. The FCC directive is used to store ASCII characters or strings. The FCB directive is used to store non-ASCII data.

©Copyright 1978 by Gary D. Gaugler. All rights reserved.

EXAMPLE:

```
FCC     ABCD#
```

would store a $41,$42,$43,$44,$23 in sequential memory locations.

```
FCB     $D
FCB     $A
FCB     4
```

would store a $0D, $0A, 04 in sequential memory locations. Note that only one constant byte per declaration may be stored using the FCB directive.

EDITING ODT

Since ODT is designed as a subroutine, it is exited via an RTS. This is accomplished by entering an 'EX' when a memory address is being displayed.

EXAMPLE:

```
013A    : EX(c/r)
```

returns to the monitor (or to DOS when called via the DOS monitor).

CONFIGURATIONS

ODT is available in two forms: monitor command or DOS transient. The monitor based ODT is ORGed at $F400 for all versions of HUMBUG and at $4000, or $5000, or $6000, or $6B00 as a disk transient. There is no need for both versions when the monitor level ODT is installed.

ODT fits in a 1K EPROM when used with HUMBUG, and in 1.4K as a transient.

Version 1. is for use with HUMBUG
Version 2. is for use with the BFD-68 disc as a transient

```
                        NAM     ODT     (ON LINE DEBUG TOOL)

                * VERSION 2.04

                        OPT     NOG,PAG

                * ON-LINE ASSEMBLER

                * (C) COPYRIGHT 1978 BY GARY D. GAUGLER.
                *     ALL RIGHTS RESERVED
                *
                * REPRODUCTION FOR NON-COMMERCIAL PURPOSES
                * IS PERMITTED.
                *
                * EDITED AND REVISED BY RONALD J. ROWLAND
                * PROGRAM TO ASSEMBLE MNEMONIC INPUT FROM THE
                * CONSOLE KEYBOARD AND STORE RESULTING HEX CODE
                * INTO MEMORY.

                * EXTERNALS

72FC            LINPTR  EQU     $72FC       POINTER INTO THE INPUT
                                            BUFFER
7000            BUFFER  EQU     $7000       THE INPUT BUFFER OF DOS
72A0            GETHN   EQU     $72A0       GET HEX ARGUMENT IN X-REG
7285            XLINET  EQU     $7285       BUFFERED INPUT IN MONITOR
7353            PCRLF   EQU     $7353       OUTPUT A CR/LF
7297            GNCHR   EQU     $7297       GET NEXT BUFFER CHARACTER
7286            OUTCH   EQU     $7286       OUTPUT CHAR
7289            INCH    EQU     $7289       INPUT CHAR
E0C8            OUT4HS  EQU     $E0CA
E0CA            OUT2HS  EQU     $E0CA
E0BF            OUT2H   EQU     $E0BF
E0CC            OUTSP   EQU     $E0CC
72A6            ZOUTST  EQU     $72A6


                        OPT     NOG,NOP
5000                    ORG     $5000


5000 BD 72A0 M1         JSR     GETHN       GET BASE ADDRESS
5003 25 2C              BCS     AGAIN       ENTRY ERROR
5005 FF 54CC            STX     TMPUPC      INIT USER PC & SCRATCH
5008 FF 54CA            STX     UPC
500B CE 54CA M2         LDX     #UPC        PRINT BASE ADDRESS
500E BD E0C8            JSR     OUT4HS
5011 86 3A              LDA A   #':         PRINT DELIMITER
5013 BD 7286 MX         JSR     OUTCH
5016 BD E0CC            JSR     OUTSP
5019 8D 1F              BSR     INPUT       GET USER INPUT
501B BD 5086 M3         JSR     VERIFY      GET OPCOD,BYTCNT,MNEMONIC
501E BD 513A            JSR     EVAL
5021 24 03              BCC     M4
5023 7E 50C7            JMP     VERBA2
5026 BD 5274 M4         JSR     GETMOD
5029 BD 5241            JSR     STORE
502C BD 50CF            JSR     LSTLIN      DISPLAY BUILT CODE
502F 20 0A      BM2     BRA     M2
5031 CE 5490 AGAIN      LDX     #MSG21      NASTY DATA .. TELL OPR
5034 BD 72A6            JSR     ZOUTST      THEN BACK TO CALLER
5037 20 C7              BRA     M1

                * GET USER INPUT AND CHECK FOR NON-MNEMONICS

5039 BD 7285 INPUT      JSR     XLINET      GET THE LINE
503C CE 7000            LDX     #BUFFER
503F FF 5406            STX     BUFPNT
5042 A6 00              LDA A   0,X
5044 81 03              CMP A   #3
5046 27 0D              BEQ     JZWARM
5048 E6 01              LDA B   1,X
504A 81 45              CMP A   #'E
504C 26 0A              BNE     FCX
504E C1 58              CMP B   #'X
5050 26 02              BNE     EXT         RETURN
5052 31                 INS                 RETURN TO DOS
5053 31                 INS                 OR TO MONITOR
5054 39         EXT     RTS

5055 7E 7283 JZWARM     JMP     $7283



                * CHECK FOR FCB OR FCC

5058 81 46      FCX     CMP A   #'F         F(XX)?
505A 26 F8              BNE     EXT         NOPE
505C C1 43              CMP B   #'C         FC(X)?
505E 26 F4              BNE     EXT         NOPE
5060 A6 02              LDA A   2,X
5062 81 42              CMP A   #'B         FCB?
5064 27 26              BEQ     FCB         YES
5066 81 43              CMP A   #'C         FCC?
5068 26 EA              BNE     EXT         NOPE

506A BD 7297 FCC        JSR     GNCHR
506D 81 0D              CMP A   #$D
506F 27 E3              BEQ     EXT
5071 81 20              CMP A   #$20
5073 26 F5              BNE     FCC         FIND DELIM BEFORE DATA
5075 20 02              BRA     FCC2        GET CHAR
5077 27 28      FCC1    BEQ     GETMOR
5079 BD 7297 FCC2       JSR     GNCHR       GET DATA
507C 81 0D              CMP A   #$D
507E 27 21              BEQ     GETMOR
5080 8D 27              BSR     SDBYTS
5082 20 F5              BRA     FCC2

5084 31         NEWST   INS                 SCRAP RETURN
5085 31                 INS                 ADDRESS
5086 BD 7297            JSR     GNCHR       SCRAP FIRST CHAR
5089 7E 5000 BM1        JMP     M1

508C BD 7297 FCB        JSR     GNCHR
508F 81 0D              CMP A   #$D
5091 27 C1              BEQ     EXT
5093 81 20              CMP A   #$20
5095 26 F5              BNE     FCB
5097 8D 517D            JSR     GETNUM
5099 25 09              BCS     NUMERR
509B A6 54D2            LDA A   ARG1
509E 8D 08              BSR     SDBYTS
50A0 31         GETMOR  INS
50A1 31                 INS
50A2 20 BA      BM3     BRA     BM2

50A4 31         NUMERR  INS                 SCRAP RETURN
50A5 31                 INS                 ADDRESS
50A6 20 1F              BRA     VERBA2      REPORT ERR

50A8 C6 01      SDBYTS  LDA B   #1
50AA F7 54DA            STA B   BYTES
50AD B7 54D9            STA A   OPCODE
50B0 8D 5241            JSR     STORE
50B2 20 19              BRA     LSTLIN

50B4 A6 00      VERIFY  LDA A   0,X         GET FIRST CHAR
50B6 81 2F              CMP A   #$2F        SLASH?
50B8 27 CA              BEQ     NEWST       YES.
50BA 8D 2B              BSR     DECODE      CHECK MNEMONIC
50BC 24 94              BCC     EXT


50BE 31         VERBAD  INS
50BF 31                 INS
50C0 CE 549F VERBA1     LDX     #MSG9       VERIFY ERROR
50C3 20 03              BRA     VXX
50C5 CE 5480 VERBA2     LDX     #MSG10
50C8 BD 72A6 VXX        JSR     ZOUTST
50CB 20 D4              BRA     BM3


                * DISPLAY THE CODE THAT IS ASSEMBLED IN MEMORY

50CD CE 54CC LSTLIN     LDX     #TMPUPC
50D0 BD E0C8            JSR     OUT4HS
```

```
5005 FE 54CC          LDX    TMPUPC     POINT TO CODE IN MEM
5008 F6 54DA          LDA B  BYTES      GET # BYTES WE HAVE
500B BD F0CA          JSR    OUT2HS
500E 5A         LSTLI1 DEC B             UPDATE # BYTES PRINTED
500F 27 05            BEQ    LSTRTS     RETN IF ALL BYTES DISPLYED
5011 BD F0BF          JSR    OUT2H      PRINT OPERAND
5014 20 F8            BRA    LSTLI1     CONTINUE

5016 7E 7353 LSTRTS JMP      PCRLF      CR/LF & RTS

                    * PARSE USER COMMAND FOR MODE, OPCODE, & OPERAND

5019 CE 537A DECODE LDX    #OPTBL
501C FF 54D4 DECOD0 STX    TBLPNT
501F A6 00          LDA A  0,X        GET TABLE KEY
5021 FE 54D6        LDX    BUFPNT
5024 A1 00          CMP A  0,X
5026 27 0F          BEQ    DECOD2     MATCHED FIRST CHAR
5028 FE 54D4        LDX    TBLPNT
502B 08       DECOD1 INX               SEARCH FOR GROUP DELIMTR
502C 6D 00          TST    0,X
502E 26 FB          BNE    DECOD1
5030 6D 01          TST    1,X        END OF TABLE?
5032 27 34          BEQ    DECOD9     YES, CMD NOT FUND
5034 08             INX
5035 20 F5          BRA    DECOD0
5037 FE 54D4 DECOD2 LDX    TBLPNT     UPDATE TABLE PNTR TO BLK
503A 08      DECOD3 INX                                    STRT
503B FF 54D4        STX    TBLPNT
503E FE 54D6        LDX    BUFPNT
5041 A6 01          LDA A  1,X        GET NEXT BUFFR CHAR
5043 E6 02          LDA B  2,X
5045 FE 54D4        LDX    TBLPNT
5048 6D 00          TST    0,X
504A 27 1C          BEQ    DECOD9
504C A1 00    DECOD7 CMP A  0,X        MATCHED?
504E 26 14          BNE    DECOD4     NO
5050 6D 01          TST    1,X
5052 2A 02          BPL    DECOD5
5054 CA 80          ORA B  #$80
5056 E1 01    DECOD5 CMP B  1,X        2ND CHAR MATCHED?
5058 26 0A          BNE    DECOD4     NO
505A F7 54D3        STA B  MFLG       INIT MODE FLAG
505D A6 02          LDA A  2,X        GET BASE OPCODE
505F B7 54D9        STA A  OPCODE
5062 0C             CLC
5063 39             RTS

5064 08       DECOD4 INX
5065 08              INX
5066 20 D2           BRA    DECOD3

5068 0D       DECOD9 SEC
5069 39              RTS

                    * GET MODE OF OPCODE & GET ARGUMENT

506A 7F 54D8 EVAL   CLR    REGNAM
506D 7F 54DB        CLR    MODE
5070 FE 54D6        LDX    BUFPNT
5073 08             INX
5074 08             INX
5075 08             INX                POINT POSSIBLE REG CHAR
5076 A6 00          LDA A  0,X
5078 A1 20          CMP A  #$20        IF SPACE OR C/R  3 CHAR
507A 27 16          BEQ    EVAL2                          MNEM
507C A1 0D          CMP A  #$D
507E 27 29          BEQ    EVAL6
5080 A1 41          CMP A  #'A         MUST BE A OR B
5082 27 04          BEQ    EVAL1
5084 A1 42          CMP A  #'B
5086 26 E0          BNE    DECOD9      SYNTAX ERROR
5088 B7 54D8 EVAL1  STA A  REGNAM
508B 08             INX
508C A6 00          LDA A  0,X
508E A1 0D          CMP A  #$D
5090 27 17          BEQ    EVAL6
5092 FF 72FC EVAL2  STX    LINPTR
5095 BD 5F          BSR    JGNCHR
5097 81 23          CMP A  #'#
5099 27 0C          BEQ    EVAL4
509B FE 72FC EVAL5  LDX    LINPTR
509E 09             DEX
509F FF 72FC        STX    LINPTR
50A2 86 30          LDA A  #$30
50A4 B7 54DB        STA A  MODE
50A7 20 04    EVAL4 BRA    GETNUM      GET NUMERIC ARGUMENT

50A9 0C       EVAL6 CLC
50AA 39             RTS

50AB 0D       EVAL8 SEC
50AC 39             RTS

                    * OBTAIN THE NUMERIC ARGUMENT, EITHER HEX,
                    * BINARY, OCTAL, OR DECIMAL.

50AD CE 54D1 GETNUM LDX    #ARG2
50B0 6F 00          CLR    0,X
50B2 6F 01          CLR    1,X
50B4 BD 3F          BSR    JGNCHR
50B6 81 24          CMP A  #'$
50B8 26 03          BNE    GETNU1
50BA 7E 5216        JMP    GETHEX
50BD 81 40    GETNU1 CMP A  #'@
50BF 27 6A          BEQ    OCTHEX
50C1 81 25          CMP A  #'%
50C3 27 6C          BEQ    BINHEX

50C5 81 30    DECHEX CMP A  #'0
50C7 2B 04          BMI    LITERL
50C9 81 39          CMP A  #'9
50CB 2F 2F          BLE    DECHE1
50CD 81 27    LITERL CMP A  #''
50CF 26 07          BNE    NOTDEC
50D1 BD 22          BSR    JGNCHR
50D3 B7 54D2        STA A  ARG1
50D6 0C             CLC
50D7 39             RTS

50D8 81 2C    NOTDEC CMP A  #',        INDEXED?
50DA 27 04          BEQ    INDX1      YES
50DC 81 58          CMP A  #'X         OR THIS THEN?
50DE 26 CB          BNE    EVAL8      NOPE, JUST JUNK
50E0 B6 54D9 INDX1  LDA A  OPCODE
50E3 81 6F          CMP A  #$6E        JMP?
50E5 27 08          BEQ    INDXD
50E7 81 AD          CMP A  #$AD        JSR?
50E9 27 04          BEQ    INDXD
50EB A6 20          LDA A  #$20
50ED 20 01          BRA    INDXD1

50EF 4F       INDXD  CLR A             SET MODE TO INDEXED
50F0 B7 54DB INDXD1 STA A  MODE
50F3 0C             CLC
50F4 39             RTS

50F5 BD 7297 JGNCHR JSR    GNCHR
50F8 81 0D          CMP A  #$D
50FA 39             RTS

50FB 84 0F    DECHE1 AND A  #$F         GET DIGIT (0-9)
50FD B7 54D0         STA A  TW
5100 A6 00           LDA A  0,X
5102 E6 01           LDA B  1,X
5104 58              ASL B
5105 49              ROL A
5106 BD 5C           BSR    SHIFT3
5108 EB 01           ADD B  1,X
510A A9 00           ADC A  0,X
510C EB 54D0         ADD B  TW
510F A9 00           ADC A  #0
5111 A7 00           STA A  0,X
5113 E7 01           STA B  1,X
5115 BD 0E           BSR    JGNCHR
5117 26 AC           BNE    DECHEX
5119 20 37           BRA    GETHE2      CHECK FOR JMP, JSR

511B 81 30    OCTHE1 CMP A  #'0
511D 2B B9           BMI    NOTDEC
511F 81 38           CMP A  #'8
5121 2A B5           BPL    NOTDEC
5123 BD 3F           BSR    SHIFT3
5125 84 07           AND A  #7
5127 AB 01           ADD A  1,X
5129 A7 01           STA A  1,X
512B BD C8    OCTHEX BSR    JGNCHR
512D 26 FC           BNE    OCTHE1
512F 0C       CLREXT CLC
5130 39              RTS

5131 BD C2    BINHEX BSR    JGNCHR
5133 27 FA           BEQ    CLREXT
5135 81 30           CMP A  #'0
5137 2B 9F           BMI    NOTDEC
5139 81 32           CMP A  #'2
513B 2A 9B           BPL    NOTDEC
513D B0 30           SUB A  #'0         STRIP BIAS
513F 47              ASR A
5140 69 01           ROL    1,X
5142 69 00           ROL    0,X
5144 20 EB           BRA    BINHEX

5146 BD 72A0 GETHEX JSR    GETHN
5149 FF 54D1        STX    ARG2
514C 25 15          BCS    GETHE1      NO DATA
514E 81 0D          CMP A  #$D
5150 26 A6          BNE    NOTDEC
5152 B6 54D9 GETHE2 LDA A  OPCODE
5155 81 6F          CMP A  #$6E
5157 27 04          BEQ    GETHE3
5159 81 AD          CMP A  #$AD
515B 26 05          BNE    GETHE4
515D C6 10    GETHE3 LDA B  #$10
515F F7 54DB        STA B  MODE
5162 0C       GETHE4 CLC
5163 39       GETHE1 RTS


                    * SHIFT 16-BIT VALUE THREE PLACES LEFT.

5164 68 01    SHIFT3 ASL    1,X
5166 69 00           ROL    0,X
5168 68 01           ASL    1,X
516A 69 00           ROL    0,X
516C 68 01           ASL    1,X
516E 69 00           ROL    0,X
5170 39              RTS


                    * PUT CODE INTO MEMORY

5171 37       STORE  PSH B             SAVE BYTCNT
5172 B6 54D9         LDA A  OPCODE
5175 FE 54CA         LDX    UPC
5178 FF 54CC         STX    TMPUPC
517B A7 00           STA A  0,X         DUMP OPCODE
517D 08              INX
517E C1 03           CMP B  #3
5180 27 17           BEQ    STORE2
5182 5A              DEC B
5183 27 0F           BEQ    STORE1
5185 B6 54D2         LDA A  ARG1
5188 A7 00           STA A  0,X
518A 08              INX
518B 5A              DEC B
518C 27 06           BEQ    STORE1
518E B6 54D1         LDA A  ARG2
5191 A7 00    STORE0 STA A  0,X
5193 08              INX
5194 FF 54CA STORE1  STX    UPC         NEW PC NOW
```

```
                    PUL B
                    RTS

                * PROCESS 2 BYTE ARGUMENTS
STORE2  LDA A   ARG2
        STA A   0,X
        INX
        LDA A   ARG1
        BRA     STORE0

                * GET MODE OF OPCODE AND WHETHER A OR B-REG ONLY

GETMOD  LDA A   OPCODE      GET THE OPCODE
        JSR     BYTCNT
        DEC B
        BNE     GETM01      NO
        PSH A
        LDA A   REGNAM
        CMP A   #'A
        BEQ     GETM
        CMP A   #'B
        BEQ     GETM00
GETM    PUL A
        BRA     GETM01

GETM00  LDA A   OPCODE
        CMP A   #$32
        BEQ     GETM1
        CMP A   #$36
        BNE     GETM2
GETM1   SUB A   #$F
GETM2   ADD A   #$10
        STA A   OPCODE
GETM02  INS
        BRA     ABREG2

GETM01  TST     MFLG        SEE IF A OR B ONLY
        BMI     ABREG       YES
        CMP A   #$8D        CHECK FOR BRANCH
        BEQ     BRANCH      TO SUBROUTINE
        CMP A   #$20        CHECK FOR BRANCHES
        BLT     GETM03
        CMP A   #$2F
        BLE     BRANCH
GETM03  BRA     ABREG2

ABREG   LDA A   REGNAM
        CMP A   #'A
        BNE     ABREG1
        LDA A   OPCODE
        BRA     ABREG2

ABREG1  CMP A   #'B
        BNE     ERRZRO      ERROR
        LDA A   OPCODE
        ADD A   #$40
ABREG2  ADD A   MODE
        JSR     BYTCNT
        TST B
        BEQ     ERRZRO

EXIT    STA A   OPCODE      FINAL COMPUTED OPCODE
        STA B   BYTES       DITTO THE BYTE COUNT
        RTS
                * PROCESS THE BRANCH COMMAND

BRANCH  LDX     UPC
        INX
        INX
        STX     HIP         HERE NOW
        LDX     ARG2
        STX     HID         WANT TO GO HERE
        LDX     #HIP
        LDA A   5,X
        SUB A   1,X
        STA A   3,X
        LDA A   4,X
        SBC A   0,X
        STA A   2,X
        BEQ     BRAN2
        CMP A   #$FF
        BEQ     BRAN1
RAERR   LDX     #MSG6       RANGE ERROR
        BRA     RAEXX
ERRZRO  LDX     #MSG0
RAEXX   JSR     ZOUTST
        JMP     GETMOR

BRAN1   LDA A   3,X
        CMP A   #$80
        BMI     RAERR
BRAN3   STA A   ARG1        THE OFFSET
        LDA B   #2
        STA B   BYTES
        RTS

BRAN2   LDA A   3,X
        CMP A   #$7F
        BHI     RAERR
        BRA     BRAN3

BYTCNT  PSH A
        PSH A
        TAB
        BSR     BYTCN0
        TAP
        FDB     $BC00,$3CAF,$4000
        FDB     $00AC,$6412,$6412,$6410
        FDB     $6410,$1101,$1004,$1000
        FDB     $1000,$110D,$100C,$100C,$100C
BYTCN0  TSX
        LDX     0,X
        PUL A
        PUL A
BYTCN1  INX
        SUB B   #8
        BCC     BYTCN1
        LDA A   0,X
BYTCN2  ROR A
        INC B
        BNE     BYTCN2
        PUL A
        BCS     ILLOP
        CMP A   #$30
        BCC     BYTCN3
        CMP A   #$20
        BCC     BYTS2
BYTCN3  CMP A   #$60
        BCS     BYTS1
        CMP A   #$8D
        BEQ     BYTS2
        AND A   #$BD
        CMP A   #$8C
        BEQ     BYTS3
        AND A   #$30
        CMP A   #$30
BYTS3   SBC B   #$FF
BYTS2   INC B
BYTS1   INC B
ILLOP   PUL A
        RTS

                * MNEMONIC TABLE
                *
                *  CONVENTION:
                *           CHAR(XX)--> TOP OF TABLE
                *           X1,X2,OPCODE
                *
                * X2 WILL BE NEGATIVE IF OPCODE IS FOR A OR B
                * REG ONLY.

OPTBL   EQU     *

                *  A(XX) MNEMONICS
        FCB     'A
        FCB     'S,'L,$48   ASL
        FCB     'S,'R,$47   ASR
        FCB     'N,'D+$80,$84 AND
        FCB     'D,'D+$80,$8B ADD
        FCB     'D,'C+$80,$89 ADC
        FCB     'B,'A,$18   ABA
        FCB     0,'B

                *  B(XX) MNEMONICS
        FCB     'C,'C,$24   BCC
        FCB     'C,'S,$25   BCS
        FCB     'E,'Q,$27   BEQ
        FCB     'G,'E,$2C   BGE
        FCB     'G,'T,$2F   BGT
        FCB     'H,'I,$22   BHI
        FCB     'I,'T+$80,$85 BIT
        FCB     'L,'E,$2F   BLE
        FCB     'L,'S,$23   BLS
        FCB     'L,'T,$2D   BLT
        FCB     'M,'I,$2B   BMI
        FCB     'N,'E,$26   BNE
        FCB     'V,'C,$28   BVC
        FCB     'V,'S,$29   BVS
        FCB     'P,'L,$2A   BPL
        FCB     'R,'A,$20   BRA
        FCB     'S,'R,$8D   BSR
        FCB     0,'C

                *  C(XX) MNEMONICS
        FCB     'B,'A,$11   CBA
        FCB     'L,'C,$C    CLC
        FCB     'L,'I,$E    CLI
        FCB     'L,'V,$A    CLV
        FCB     'L,'R,$4F   CLR
        FCB     'M,'P+$80,$81 CMP
        FCB     'O,'M,$43   COM
        FCB     'P,'X,$8C   CPX
        FCB     0,'D

                *  D(XX) MNEMONICS
        FCB     'A,'A,$19   DAA
        FCB     'E,'C,$4A   DEC
        FCB     'E,'S,$34   DES
        FCB     'E,'X,9     DEX
        FCB     0,'E

                *  E(XX) MNEMONICS
        FCB     'O,'R+$80,$88 EOR
        FCB     0,'I

                *  I(XX) MNEMONICS
        FCB     'N,'C,$4C   INC
        FCB     'N,'S,$31   INS
        FCB     'N,'X,8     INX
        FCB     0,'J

                *  J(XX) MNEMONICS
        FCB     'M,'P,$6E   JMP
        FCB     'S,'R,$AD   JSR
        FCB     0,'L

                *  L(XX) MNEMONICS
        FCB     'D,'A+$80,$86 LDA
        FCB     'D,'S,$8E   LDS
        FCB     'D,'X,$CE   LDX
        FCB     'S,'R,$44   LSR
        FCB     0,'N

                *  N(XX) MNEMONICS
        FCB     'E,'G,$40   NEG
        FCB     'O,'P,1     NOP
        FCB     0,'O

                *  O(XX) MNEMONICS
        FCB     'R,'A+$80,$8A ORA
```

Continued on pg. 45

SOFTWARE STANDARDS

Who Needs Them?

BY TOM PITTMAN
PO Box 23189
San Jose, CA 95153

Any time someone gets up the energy to spend some effort on standardization, opinion polarizes. On the one hand we hear cheers and encouragement and on the other hand boos and discouragement. Since I am sitting on a Standards committee, you know how I feel. Let me therefore answer the detractors of standardization in general, and of software standards in particular.

The first argument raised against standards is, "They will inhibit progress." That may be true. Standards certainly slow down the proliferation of different ways of doing things. Some of those different ways are indeed beneficial and good; most of them are merely different. I suppose it is true that the larger the number of different ways there are for doing things, the more chance there is that a better way (or at least the better for any one particular application) is represented among them. At the same time the chance is reduced that any one of those ways is sufficiently developed to be of any use. This is because the total development effort is dissipated over too many different thrusts. We can see what software standards have done in the past: FORTRAN is such a standard, and because it is so, virtually every mainframe manufactured in the country is obliged to supply a FORTRAN compiler. The language is usable, its problems are well known, and those who need a language as a tool can generally depend on (mostly) predictable results when they use the ANS version of the language. Because there is widespread agreement among language designers that FORTRAN is a crummy language, each of them is designing another, better, language. We have dozens of the languages and they are all better than FORTRAN. None of them are very useful, because they lack the widespread support accorded to a *standard* language. I claim there is no reason to suppose that software standards in the future will be any different. Standards do not inhibit progress so much as they make the results of that progress available to the users.

"Reprinted from the Homebrew Computer Club Newsletter, Mountain View, CA."

The next argument reminds us that all microscopes are different and insists that the incompatibilities make standards either meaningless or impossible. I will admit to incompatibilities in the different CPUs, but among the most popular of them, I think the similarities outweigh the differences, at least in the areas we have chosen to standardize. To be sure, there are processors which will have a great deal of difficulty fitting into any standard we come up with; you may note, however, that virtually all of these have no second source. I suggest this as evidence that there is a consensus of what a microprocessor should be like, and that the codification of that consensus is our standard. But the world in which we work will admit to a stronger response than that. Not only is it meaningful to speak of standards across microprocessor lines, but there is a whole family of microprocessors which execute substantially the same instruction set; they are incorporated into systems by seven different chip manufacturers and dozens of OEMs, yet almost universally software developed on one of these systems cannot be loaded into another. The users cannot go out and buy package software (not even source code!) and expect to use it without significant modification. Standards among large computers are those defined by the manufacturer, because there are no others; the users benefit from the availability of software from independent vendors because there is no compatibility problem. If we had the same uniformity in our software we could quit standing on each other's toes.

Our standards committee has selected two general areas of immediate value for standards work. One of these deals with relocatable code format. There is no valid reason why we cannot come up with a standard which is applicable to all 8080-class CPUs (i.e., 8080, 8085, Z80), when dealing with object modules generated by an assembler or a FORTRAN compiler. No doubt there are many features which modern languages and/or debugging aids would like to see in a loader (using information passed from the compiler). Some of these are probably totally impractical in the general sense which we need to keep our focus; many can be supported by permission (as opposed to by requirement). Thus we retain the "standard" format as a subset of the enhanced form which

supports all the goodies, just as nearly every FORTRAN compiler supports features which extend the ANS standards.

It is true that by the time our standards are adopted there will already exist a substantial amount of incompatible software. This is unfortunate and only serves to underscore the need to avoid fruitless delay. However, once the standards are adopted, there will be a great incentive to develop new software products in accordance with the standards, and I think it reasonable to believe that the conversion of existing software to conform will be less effort than might be expected from a complete remake. The burden is on us as standards makers, to build into them the flexibility for needed growth into the foreseeable future. I do not see this to be impossibly difficult for the definition of relocatable code.

The other software area of immediate concern is much more controversial: assembly language code. There is obviously no way we can define a standard assembly language which encompasses all microprocessors. We can, however, agree on a standard for a macro assembler. We can define a standard representation for the common addressing modes, and suggest guidelines for the incorporation of unusal addressing modes. We can even suggest guidelines for the naming of instruction mnemonics. Thus where two different processors each have an instruction which performs substantially the same function in their respective repertoires, a user can expect to use the same name. Clearly this does not apply to widely differing functions, but my experience is that the similarities outweigh the differences.

Ben Franklin once said, "We must all hang together, or surely we shall all hang separately." I think the wisdom of this aphorism can be appplied directly to our own work.

ADDENDUM

After four months of meetings, the IEEE microprocessor standards effort is beginning to show some progress. The greatest concentration of talent, energy, and warm bodies continues to be in the hardware area, dealing with bus specifications. So far this is still in the fact-finding stage, but we have heard presentations on the S-100, SBC-80, and PI busses. A major problem area in the S-100 bus definition lies in the specification of DMA protocol; this is on the agenda for next month.

In the software area, we have heard presentations on the Intel floating point format and first draft of a mnemonic standard. We are still attempting to get better support for the work on relocatable code.

One of the continuing problems in every specialty area of this effort is the turnover rate of participants. As new people come to the successive meetings, they often bring specific objections to the standardication efforts which have already been discussed in previous meetings; reopening the discussion each month only wastes valuable meeting time. I have responded to some of the objections in the software area in the essay titled, "Software Standards – Who Needs Them?" The objections raised with respect to bus standards are often very similar. I say, "Work with us or stay home."

Continued from pg. 43

```
619 541D 00             FCR     0,'P
621                 * P(XX) MNEMONICS
622 541F 55             FCR     'U+'L+$80,$32 PUL
623 5422 53             FCR     'S+'H+$80,$36 PSH
624 5425 00             FCR     0,'R
626                 * R(XX) MNEMONICS
627 5427 4F             FCB     'O+'L,$49  ROL
628 542A 4F             FCB     'O+'R,$46  ROR
629 542D 54             FCR     'T+'I,$3B  RTI
630 5430 54             FCR     'T+'S,$39  RTS
631 5433 00             FCR     0,'S
633                 * S(XX) MNEMONICS
634 5435 42             FCR     'B+'A,$10  SBA
635 5438 42             FCR     'B+'C+$80,$82 SBC
636 543B 45             FCR     'E+'C,$D   SEC
637 543E 45             FCR     'E+'I,$F   SEI
638 5441 45             FCR     'E+'V,$B   SEV
639 5444 54             FCR     'T+'A+$80,$87 STA
640 5447 54             FCR     'T+'S,$8F  STS
641 544A 54             FCR     'T+'X,$CF  STX
642 544D 55             FCR     'U+'B+$80,$80 SUB
643 5450 57             FCB     'W+'I,$3F  SWI
644 5453 00             FCR     0,'T
646                 * T(XX) MNEMONICS
647 5455 41             FCR     'A+'B,$16  TAB
648 5458 41             FCR     'A+'P,6    TAP
649 545B 42             FCB     'B+'A,$17  TBA
650 545E 50             FCB     'P+'A,7    TPA
651 5461 53             FCR     'S+'T,$40  TST
652 5464 53             FCR     'S+'X,$30  TSX
653 5467 58             FCR     'X+'S,$35  TXS
654 546A 00             FCR     0,'W
656                 * W(XX) MNEMONICS
657 546C 41             FCR     'A+'I,$3F  WAI
658 546F 00             FCR     0,0        MARK END OF TABLE

660 5471 07     MSG0    FCR     $7
661 5472 49             FCC     'INPUT ERROR'
662 547D 0D             FCR     $D,$A,0

664 5480 07     MSG10   FCR     $7
665 5481 49             FCC     'INVALID DATA'
666 548D 0D             FCR     $D,$A,0

668 5490 07     MSG21   FCR     $7
669 5491 49             FCC     'INPUT ERROR'
670 549C 0D             FCR     $D,$A,0

672 549F 07     MSG9    FCR     7
673 54A0 49             FCC     'ILL MNEMONIC'
674 54AC 0D             FCR     $D,$A,0

676 54AF 07     MSG6    FCR     7
677 54B0 52             FCC     'RELATIVE BRANCH TOO FAR'
678 54C7 0D             FCR     $D,$A,0

680                 * RAM STORAGE AREA
681 54CA        UPC     RMB     2
682 54CC        TMPUPC  RMB     2
683 54CE        TEMPX1  RMB     2
684 54D0        TW      RMB     1
685 54D1        ARG2    RMB     1
686 54D2        ARG1    RMB     1
687 54D3        MFLG    RMB     1
688 54D4        TRLPNT  RMB     2
689 54D6        BUFPNT  RMB     2
690 54D8        REGNAM  RMB     1
691 54D9        OPCODE  RMB     1
692 54DA        BYTES   RMB     1
693 54DB        MODE    RMB     1
694 54DC        HIP     RMB     1
695 54DD        LOP     RMB     1
696 54DE        HIN     RMB     1
697 54DF        LON     RMB     1
698 54E0        HID     RMB     1
699 54E1        LOD     RMB     1
700 54E2        LRAM    EQU     *

702                     END
        SYMBOL TABLE:
ABREG   5284    ABRFG1  52C0    ABREG2  52C9    AGAIN   5071    ARG1    54D2
ARG2    54D1    BINHEX  5201    BM1     5089    BM2     502F    BM3     50A3
BRAN1   530A    BRAN2   5319    BRAN3   5310    BRANCH  5209    BUFFER  7000
BUFPNT  54D6    BYTCN0  5347    BYTCN1  534C    BYTCN2  5353    BYTCN3  5362
BYTCNT  5321    BYTES   54DA    BYTS1   5377    BYTS2   5376    BYTS3   5374
CLREXT  51FF    DECHE1  51CB    DECHEX  5195    DECOD0  50EC    DECOD1  50FB
DECOD2  5107    DECOD3  510A    DECOD4  5134    DECOD5  5126    DECOD7  511C
DECOD9  5138    DECODE  50E9    ERRZRO  5301    EVAL    513A    EVAL1   5158
EVAL2   5162    EVAL4   5177    EVAL5   516B    EVAL6   5179    EVALB   517B
EXIT    52D2    EXT     5054    FCB     508C    FCC     506A    FCC1    5077
FCC2    5079    FCX     5058    GETHE1  5233    GETHE2  5222    GETHE3  522D
GETHE4  5232    GETHEX  5216    GETHN   72A0    GETM    5289    GETM1   5297
GETM2   5299    GETM00  52BC    GETM01  52A1    GETM02  52BE    GETM03  52B2
GETM00  5274    GETM0R  50A1    GETNU1  51B0    GETNUM  5170    GNCHR   7297
HID     54E0    HIN     54DE    HIP     54DC    ILLOP   5378    INCH    7289
INDX1   51B0    INDXD   51BE    INDXD1  51C0    INPUT   5039    JGNCHR  51C5
J7WARM  5055    LINPTR  72FC    LITERL  5190    LOD     54E1    LON     54DF
LOP     54DD    LRAM    54E2    LSTLI1  50DE    LSTLIN  50CE    LSTRTS  50E6
M1      5000    M2      5008    M3      5018    M4      5026    MFLG    54D3
MODE    54DB    MSG0    5471    MSG10   5480    MSG21   5490    MSG6    54AF
MSG9    549F    MX      5013    NEWST   50B4    NOTDEC  51AB    NUMERR  50A5
OCTHE1  51E0    OCTHEX  51F0    OPCODE  54D9    OPTRL   5378    OUT2H   E0BF
OUT2HS  E0CA    OUT4HS  E0C8    OUTCH   7286    OUTSP   E0CC    PCRLF   7353
RAERR   52FC    RAEXX   5304    REGNAM  54D8    SDBYTS  50A9    SHIFT3  5234
STORE   5241    STORE0  5261    STORE1  5264    STORE2  5269    TRLPNT  54D4
TEMPX1  54CE    TMPUPC  54CC    TW      54D0    UPC     54CA    VERBA1  50C2
VERBA2  50C7    VERBAD  50C0    VERIFY  50B6    VXX     50CA    XLINE1  7285
70UTST  72A6
```

Letters

GOODBYE TO ACKERMANN'S FUNCTION

Gentlepeople:

Let me add another chapter to the discussion of Ackermann's function that appeared in the March 79 issue (No. 33 page 46) of your magazine. As Les Hancock pointed out, this function is a good test of the capability of a language. Hence . . . I enclose a STRUBAL+ version of his examples. To those not yet acquainted with STRUBAL+ (you haven't been reading *Dr. Dobb's* recently), it is a compiler which runs on a 6800 computer. It supports integer, floating-point, string, and mixed data types . . . crutch-coding . . . BASIC-type statements plus FORTRAN-like I/O formatting and structured forms (WHILE, IF-THEN-ELSE) and file I/O, etc. (See *Dr. Dobb's* No. 23 page 10 and No. 28 page 28.) The example was run on a SWTPC computer with 32K of memory. The example compiled to about 500 bytes . . . but the minimal runtime package adds about 3.7 K. Note on line 10 the ease of formatting the output compared to the tiny-C example which required a special function. The example used integer variables for speed, but floating-point could be used if the numbers would get too large. STRUBAL+ does not directly support recursion. It does, however, supply the primitives "PUSH" and "POP" which allow any variable to be stacked. Once M gets to 3, things run fairly slow. I let the example run for about a half-hour and it got to A(3,8). By then, over 6K bytes of stack had been used. The use of crutchcoding in line 5 illustrates the power of this feature to make the machine do exactly what is needed . . .in this case, get lots of stack space. I exhibit this example as an illustration of the clean syntax of STRUBAL+ and how easy it is to read a STRUBAL+ program.

Sincerely yours,
Robert D. Grappel

151 Tremont Street, Suite 8P
Boston, Massachusetts 02111

```
0001.          * DEMO OF ACKERMAN'S FUNCTION IN STRUBAL+
0002.          * R. D. GRAPPEL     MARCH 1,1979
0003.          *
0004.                INTEGER M,N,A
0005.                LDS #$7000              GET LOTS OF STACK SPACE
0006.                FOR M=0 TO 9
0007.                PRINT /
0008.                FOR N=0 TO 9
0009.                GOSUB AKRMAN
0010.                PRINT [6],A
0011.                NEXT N
0012.                NEXT M
0013.                STOP
0014.          *
0015.          *
0016.          * NOTE USE OF "PUSH" AND "POP" FOR RECURSION
0017.          *
0018.          AKRMAN PUSH M
0019.                PUSH N
0020.                IF M .NE. 0 THEN S2
0021.          *
0022.          S1    A=N+1
0023.                POP N
0024.                POP M
0025.                RETURN
0026.          *
0027.          S2    IF N .NE. 0 THEN S3
0028.          *
0029.                M=M-1
0030.                N=1
0031.                GOSUB AKRMAN
0032.                POP N
0033.                POP M
0034.                RETURN
0035.          *
0036.          S3    N=N-1
0037.                GOSUB AKRMAN
0038.                N=A
0039.                M=M-1
0040.                GOSUB AKRMAN
0041.                POP N
0042.                POP M
0043.                RETURN
0044.          *
0045.                END
```

Dear *Dr. Dobb*:

The accompanying programs written in ALTAIR 4.0 BASIC compute and list values of Ackermann's function as discussed by Gabrielson, Hancock, et al. in *DDJ* #31 and #33.

In line #310 of the first version, the computing subroutine calls itself recursively, after first "pushing" the first argument, M, onto the array STACK. The second argument, N, need not be saved, since it is to be replaced by the returned value. Seven bytes are needed per level of recursion, five for the subroutine call and two for the saved argument. For M greater than zero, the stack depth required to compute AK(M,N) appears to be exactly AK(M,N) – 2, so that workspace needed varies in proportion to the value of the function.

In the second version, the computing subroutine does not call itself, but instead uses the array, AK(M,N), to store function values as they are computed. Note that to compute AK(M,N), the value of AK(0,AK(M,N)–1) must be computed, so that the size of the needed array also varies in proportion to the value of the function.

The results for a few values of M and N are tabulated below. They indicate that the non-recursive solution works much faster while using about as much memory as the recursive solution.

Sincerely,
James Monagan

806 Clark Street
Iowa City, Iowa 52240

Recursive Solution

M,N	AK(M,N)	Subroutine calls	Maximum stack depth	Workspace (bytes)
3,3	61	1187	59	413
3,4	125	5093	123	861
3,5	253	21095	251	1757

Iterative Solution

M,N	AK(M,N)	Number of iterations	Array size needed	Workspace (bytes)
3,3	61	81	4 X 61	488
3,4	125	161	4 X 125	1000
3,5	253	321	4 X 253	2024

```
10 'RECURSIVE        02/22/79
20 PRINT"RECURSIVELY COMPUTES & LISTS VALUES FOR ACKERMANN'S FUNCTION
30 CLEAR100:DEFINTA-Z:SX=(FRE(0)-200)\7:DIM STACK(SX)
40 '
50 PRINT:PRINT"DEFINE UPPER LIMITS FOR ARGUMENTS OF ACKERMANN(M,N)
60 INPUT"M MAX = ";MX
70 INPUT"N MAX = ";NX
80 '
90 'MAIN LOOP
100 '
110 FORMA=0TOMX
120 FORNA=0TONX
130 GOSUB220
140 PRINTUSING"######";AK;
150 NEXTNA:PRINT
160 NEXTMA
170 END
180 '
190 'COMPUTE ACKERMANN(M,N)
200 '
210 M=MA:N=NA
220 IFN=0THENAK=N+1:RETURN
230                                    'M>0
240 IFN=0THENM=M-1:N=1:GOTO230
250 '
260                                    'M>0, N>0
270 N=N-1
280 STACK(S)=M:S=S+1:GOSUB230:S=S-1:M=STACK(S)   'AK = ACKERMANN(M,N-1)
290 M=M-1:N=AK:GOTO230
READY
```

```
10 'ACKERMAN    -    02/22/79
20 PRINT"ITERATIVELY COMPUTES & LISTS VALUES FOR ACKERMANN'S FUNCTION
30 CLEAR100:DEFINTA-Z
40 '
50 PRINT:PRINT"DEFINE UPPER LIMITS FOR ARGUMENTS OF ACKERMANN(M,N)
60 INPUT"M MAX = ";MX
70 INPUT"N MAX = ";NX
80 NA=(FRE(0)-300)\(2*MX+2):DIM AK(MX,NA)
90 '
100 'MAIN LOOP
110 '
120 FORMA=0TOMX
130 FORNA=0TONX
140 GOSUB220
150 PRINTUSING"######";AK(MA,NA);
160 NEXTNA:PRINT
170 NEXTMA
180 END
190 '
200 'COMPUTE ACKERMANN(M,N)
210 '
220 M=MA:N=NA
230 IFM=0THENAK(MA,NA)=N+1:GOTO340
240                                    'M>0
250 IFN>0THEN300
252 IFAK(M-1,1)=0THENM=M-1:N=1:GOTO230
254 AK(M,N)=AK(M-1,1):GOTO340
260 '
270                                    'M>0, N>0
280 IFAK(M,N-1)=0THENN=N-1:GOTO250
290 IFAK(M-1,AK(M,N-1))=0THENN=AK(M,N-1):M=M-1:GOTO230
300 AK(M,N)=AK(M-1,AK(M,N-1))
310 '
320 IFN=NAANDM=MATHENRETURNELSE220
READY
```

ANOTHER RECORD BROKEN

Dear *Dr. Dobb's*:

As there seems to be an unofficial contest for the shortest, fastest, reentrant subroutines to save and restore all 8080 registers (re: *Dr. Dobb's* #29, page 3 and #33, page 45), I submit the following which totals 11 bytes and 124 states, excluding stack and calling code (which may be either a call or RST X). This compares to 20 bytes and 172 states on the most recent submittal.

Sincerely,
Robert J. Minnihan

Dynamic Systems, Inc.
6975 Washington Avenue South
Minneapolis, MN 55435

```
PUSHER    XTXL        ; SAVE (HL) BY EXCHANGING WITH RETURN
          PUSH  PSW   ; ADDRESS ON STACK TOP
          PUSH  D     ; SAVE OTHER REGISTERS
          PUSH  B     ;
          PCHL        ; JUMP BACK TO CALLING LOCATION

PULLER    POP   H     ; (HL) = RETURN ADDRESS TO CALLING LOCATION
          POP   B     ; RESTORE OTHER REGISTERS
          POP   D     ; EXCEPT (HL)
          POP   PSW   ;
          XTHL        ; NOW RESTORE ORIGINAL (HL) & PLACE
          RET         ; RETURN ADDRESS ON STACK, THEN RETURN
```

AN EFFICIENT REENTRY ROUTINE

Dear *Dr. Dobb's*:

On page 45 *DDJ* 33 you published a small subroutine by W. W. Moss which is claimed to be reentrant but is not. The routine published is *serially* reentrant only.

Any routine used for interrupt processing can bomb this routine out. If and only if you are aware of this potential problem should you use the routine. If you really want a serially reentrant routine, then a routine one is both shorter and faster.

You should also be aware of a bug in the original routine that causes the carry flag to be cleared. Routine two corrects this bug. The Dad instruction always destroys the previous value of the carry flag and therefore it must be restored in some manner.

Lastly, I have changed the Push order so that the PSW is on the top (bottom?) of the stack. This allows either the flags or the accumulator to be easily accessed to return parameters.

Yours Truly,
Charles E. Marvin, Jr.

40 W. Long St.
Columbus, OH 43215

ROUTINE ONE — REENTRANCY SERIAL — RELOCATABILITY NON-IMMEDIATE

HEX CODE	LABEL	INSTRUCTION	BYTES	CYCLES
22????	PUSHER:	SHLD ADDR	3	16
E3		XTHL	1	18
D5		PUSH D	1	11
C5		PUSH B	1	11
F5		PUSH PSW	1	11
E5		PUSH H	1	11
21XXXX		LXI H,0	3	10
	ADDR:	EQU $-2		
C9		RET	1	10
TOTALS			12	98

ROUTINE TWO — REENTRANCY TOTAL — RELOCATABILITY IMMEDIATE

HEX CODE	LABEL	INSTRUCTION	BYTES	CYCLES
E3	PUSHER:	XTHL	1	18
D5		PUSH D	1	11
C5		PUSH B	1	11
F5		PUSH PSW	1	11
E5		PUSH H	1	11
F5		PUSH PSW	1	11
210A00		LXI H,10	3	10
39		DAD SP	1	10
7E		MOV A,M	1	7
23		INX H	1	5
66		MOV H,M	1	7
6F		MOV L,A	1	5
F1		POP PSW	1	10
C9		RET	1	10
TOTALS			16	137

```
ROUTINE THREE        REENTRANCY      RELOCATABILITY
                     TOTAL           IMMEDIATE

HEX CODE    LABEL    INSTRUCTION     BYTES     CYCLES
E1          PULLER:  POP   H           1         10
F1                   POP   PSW         1         10
C1                   POP   B           1         10
D1                   POP   D           1         10
E3                   XTHL              1         18
C9                   RET               1         10
TOTALS                                 6         68
```

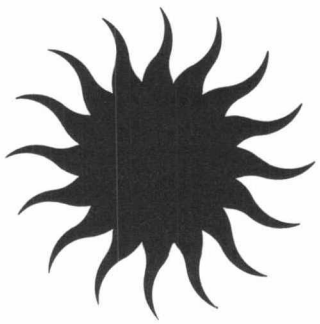
Dear *Dr. Dobb's*:

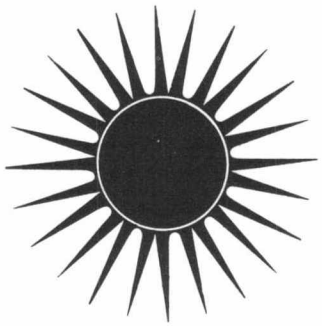
The following modifications to the Ithaca Audio Z-80 board will allow it to use the 5 Volt only 2716. I have done this mod to 5 boards now and they all work fine.

Cut trace at U36 pin 21
Cut trace at U36 pin 19
Cut trace at U24. This trace is seen looking down at the front of the board and runs between pins 3 and 4 of U24.

Jumper pin 12 U23 to pin 1 U1
Jumper pin 19 U30 to pin 4 U29
Jumper pin 40 U30 to pin 19 of U36
Jumper pin 12 of U36 to pin 18 of U36
Jumper pin 24 of U36 to pin 21 U36.

Thanks,
Peter W. Sargent





HINTS FOR GEORGE RISK KEYBOARD ASSEMBLY

Dear Sirs,

I recently assembled the GRI Model 756 keyboard kit. There was one item of confusion: the instructions do not mention where to install the 'return' pushbutton switch. Since the kit had 56 momentary buttons, I installed two of them in the space for the 'return' key. When the key tops were installed, I had to remove those two and put one of them in the proper location. No big thing, but a note in the documentation would have saved some confusion.

One other item: on the sheet of free keyboard tricks, there seem to be two errors in the bottom schematic. The 'strobe' signal should go low, not high. Also pin 7 of the 74165 should be left open, not grounded, since its an output.

All in all, I'm happy with the unit. We will be putting it to hard use in a word-processing system within the next month or so.

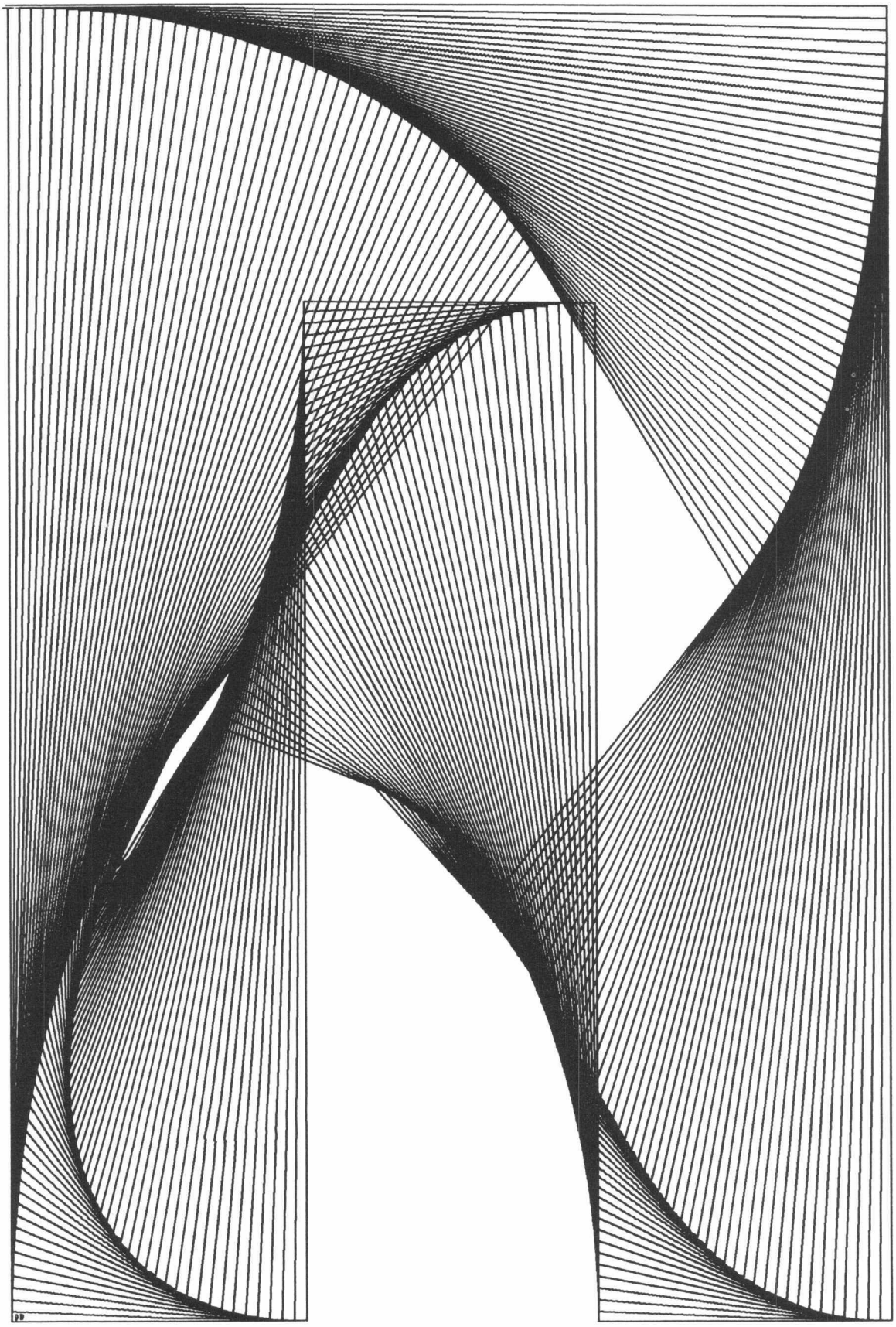
A little more detail that might help someone building one of these kits: there is room for two switches in the return key position. Only one switch should be installed. Put it in the center set of holes.

The other item mentioned concerned a suggestion for building a serial output circuit (a TX only 'UART') from four TTL chips.

Actually, their documentation is quite good. The board has a 'shift lock' key that is a real necessity in a word processing system.

Regards,
J.F. Sullivan

1330 S. Rosewood
Santa Ana, CA 92707



© David Dameron 1979

Dr. Dobb's Journal of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 36 | June/July 1979 | Volume 4, Issue 6

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

Discovering the 26th Prime 4
Curt Noll

XS-O: A Self-Explanatory School Computer 8
Nievergelt, Frei, Burkhart, Jacobi, Plattner, Sugaya, Weibel, Weydert

Selecting Business Software for Microcomputers 24
Greg B. Scott

What's a Cosmac? 28
Karl Mosgofian

EXOS-6500 Software Development Tool Kit 29
Don Colburn

How to Make Your Computer Talk 32
Charles M. Phelan

A Look at Computer Music 33
Curtis Roads

Growing, Pruning & Climbing Binary Trees with tiny-c 37
Les Hancock

BY CURT NOLL
25931 Fairview Avenue
Hayward, California 94542

Note: On October 30, 1978, two Hayward, California high school students achieved worldwide recognition when they discovered the largest known prime number: two to the 21,701st power minus one. The students, Laura Nickel and Curt Noll, had spent three years finding the number, which consists of 6,533 digits.

Nickel and Noll had become interested in computers in 1975 at Hayward High School when they learned computer programming on the school's small computers.

"When we saw a printout of the largest known prime on display in the math department at Cal State," Noll, 17, said recently, "that's when we decided we wanted to find the next one. Everybody said it couldn't be done. That encouraged us to prove 'em wrong."

After contacting Professor Arthur Simon, Chairman of the Mathematics Department at Cal State Hayward, and math lecturer Don Jurca, they were provided information and research material about primes. Jurca provided them with access to the university's computer, a CDC Cyber 174.

"We were pretty lucky to have a Cyber," Noll said. "Very few computers are fast enough to test Mersenne primes in a reasonable amount of time."

The students were constantly discouraged by professionals, including Jurca, who told them their project was "doomed." At times it seemed the pessimists were right, and the two had many "dark hours." They vowed to prove these pessimists wrong.

And prove them wrong they did – twice. Not only did Nickel and Noll discover the 25th prime, for which they received worldwide recognition, but, on Feburary 9, 1979, they discovered the 26th prime as well. This latest discovery went largely unreported by the media, but Curt Noll has provided us with a printout of the number (see inside back cover). We're pleased to bring you his personal story on the discovery of the 25th Mersenne prime, as well as the official paper and abstract which announced the discovery.

Nickel and Noll, by the way, are currently working on discovering the 27th prime. – SMR

About three years ago, Laura and I set out on a project that was to pay off with tremendous results. In fact, it seems that it will not level off for quite some time. The project to which I am referring is the search for the 25th Mersenne prime.

For those who are not familiar with Mersenne primes, or primes in general, I will give a brief refresher course. A prime number is one whose only factors are 1 and itself; 2, 3, 5, 7, 11, 13 are the first 6 prime numbers. A Mersenne prime is a prime of the form $2^n - 1$ where n is a positive integer. $2^2 - 1 = 3$, $2^3 - 1 = 7$, $2^5 - 1 = 31$ are the first 3 Mersenne primes.

The usual method for determining whether or not a number qualifies is to divide by numbers less than the square root of the number, or until you find a factor. This method works fine for numbers the size of 1,000,000, since you would only need to check for divisibility of 1000 numbers. Yet when numbers get extremely large, even the fastest computers are of no use. But thanks to the work of E. Lucas in 1876, and D. H. Lehmer in 1930, an algorithm called the Lucas-Lehmer test was devised for checking the primality of Mersenne numbers (numbers of the form $2^n - 1$). To test $2^p - 1$ for primality using the Lucas–Lehmer test, a sequence of numbers is generated beginning with 4, in which each term u_i is equal to the square of the preceding term minus 2, $(u_{i-1}{}^2 - 2)$, reduced modulo $2^p - 1$ (that is, u_5 = the remainder of $u_4{}^2 - 2 \div 2^p - 1$). If the pth term in the sequence is 0, then the number is prime, otherwise it is not prime. Here are two examples: (This works for $P \geqslant 3$).

The test for $2^5 - 1 = 31$.

$$u_2 = 4$$

$$u_3 = u_2{}^2 - 2 \pmod{2^5 - 1} = 14$$

(because the remainder of the 14 ÷31 is 14)

$$u_4 = u_3{}^2 - 2 \pmod{2^5 - 1} = 8$$

(because the remainder of 194 ÷ 31 is 8)

$$u_5 = u_4{}^2 - 2 \pmod{2^5 - 1} = 0$$

(because $u_5 = 0$, $2^5 - 1$ is prime)

The test for $2^{11} - 1$

$u_2 = 4$
$u_3 = 14 \quad (4^2 - 2 \pmod{2^{11} - 1})$
$u_4 = 194 \quad 14^2 - 2 \bmod 2^{11} - 1)$
$u_5 = 788 \quad 194^2 - 2 \bmod 2^{11} - 1)$
$u_6 = 701 \quad$ etc.
$u_7 = 119$
$u_8 = 1877$
$u_9 = 240$
$u_{10} = 282$
$u_{11} = 1736$

because $u_{11} = 1736$ and *not* zero, $2^{11} - 1$ is *not* prime.

$$89 \cdot 23 = 2047 = 2^{11} - 1.$$

The Lucas-Lehmer test does not tell you what the factors are; it merely tells you if the Mersenne number is prime or not. But this test grows considerably faster for large numbers. For example, where I would spend 10^{1214} years to test $2^{8191} - 1$ by the regular method, I need only 23 minutes with the Lucas-Lehmer test!

I wish to say that even though the Lucas-Lehmer test is much faster, it is still a major task for fast computers when you get into the range of $2^{20000} - 1$. Here the test requires the generation of 20000 terms in the u_i sequence. Each term generation requires squaring a 20000 bit number (approximately 6000 digits) and taking the 40000 bit result (approximately 12000 digits) and subtracting 2 and then reducing modulo of another 20000 bit number. To put this in human terms, if a person could multiply two 15-digit numbers together in one minute, then that person could perform the Lucas-Lehmer test for $2^{21701} - 1$ in about 3,888 years.

California State University at Hayward was nice enough to let us use their CDC Cyber 174. The Cyber 174 was an ideal machine for the Lucas-Lehmer test for several reasons:

1. The Cyber 174 has two very fast CPU'S.
2. The Cyber 174 has a word size of 60 bits which makes it ideal for multiprecise multiplication.
3. The Cyber 174 is a very reliable machine because of its single correct double detection parity error feature.

Another good reason was that the Cyber 174 is a timeshare system for the California State University system, and usually one of the two CPU's is idle due to the inherent nature of timeshare on a fast machine. Therefore the program could reside as an idle time job and run about 90% of the time in the CPU. The last and best reason was that the computer time was *free*.

I would like to return for a moment to the actual program. The program that was used was written in COMPASS (Cyber assembly language). High level languages had too much overhead and required much more core than a COMPASS Lucas-Lehmer test. Our 3K program multipled the required numbers in base 2^{47}. The modulo $2^n - 1$ reduction was done with binary shifts and adds. Example:

$$104 \pmod{2^4 - 1}$$
$$104 = 1101000_2$$

now break the number after 4 bits from the right

$$110 \quad 1000$$

now add the two numbers to obtain the answer

$$\begin{array}{r} 1000_2 \\ \underline{110_2} \\ 1110_2 \end{array} = 14 \quad [104 \div 15 = 6 \text{ remainder } 14].$$

Therefore the program only needs to multiply, add, subtract, and shift. Yet the major task is in the squaring, which comes down to just careful programming.

We spent 2½ years looking over several ways to approach the problem, and after four versions failed to give us good speed, our fifth version of the program started on October 11, 1978. We started testing at $2^{21000} - 1$, where Tuckerman (he found $2^{19937} - 1$ to be prime) had left off. After 44 tests, on October 30, 1978, we found $2^{21701} - 1$ to be prime. The test for this number took seven hours, eight minutes, and twenty seconds.

I should stop to say that one need only look at $2^p - 1$, where p is prime, because $2^n - 1$ where n is not prime can never be prime. We also were able to cut down our tests even more by discarding the numbers for which small factors had been found. We used a list that Dr. Wagstaff had made of Mersenne numbers that had factors less than $2^{35} - 1$.

I said in the beginning that I was reaping tremendous results as a result of the discovery. By this I don't mean publishing a paper, or getting worldwide coverage, etc. This is not why I spent months of study and work on the project. The reason was the personal satisfaction of doing something for me and the world of mathematics. All those hard hours were made worth it when I was able to laugh in the faces of those who kept saying "It will never be done, why waste your time?" From all this I have learned one valuable thing: *never* let them tell you, "You can't."

```
9856140612934423376292800357002337948291195231891916319254587638023719991739894135357667509038906
9376224029198033655100624308247503737427045463405558995482952926765329443125591968186626470815051 5
1821208670072062662244135785787663162882922000802040699085958808649045935301957919928360662713973 2
2016336401432076989173442258953647032430558877151892524852554146167146883853769910561246320694705 5
1245708892171197864405670313577094738663472978765024173697554322425514722377074162976565017144454 1
1153411865771642670096665217784040079089425722044373575364470132023204490873743591198846888965706 5
3153635965699349888166816675325403419739923012069285047750708058098369546148066530512705922505763 3
3936397344874355127528362865683467040277033890120353882586388527337272529680233117571122695445036 3
4933557823964169012365540952244084485151548107189417781442530197357722865554948884829926768347366 0
0663066924318505506867453994455209613903356661381551303170707697066559694566821068447680650509746 0
3538440573896895229527083162847671360209681320052166229770222864138303772012744507729541652286725 1
0862593089242569491305464825292051070670232638922508876510803630212114342714333976089893083303643 0
0732585576535892044008832875413115983363441152995391893368657891779064964168643604233277423002160 2
8240865056235601373753305378621159873680555706434488664079585221580109298864192394145593692823239 9
5582350982708215349265311035134009700812402989211058114698144382506102890169695315562317736449187 1
3398451013610901563088260312670567852392406888546913219408585158313060052720828019956940355099824 7
0822389954981571973382421635115832184667560777181033413857160655904157765106059858641094524241435 1
0105710136533420290951417937038794020551239629386981798285845890434316234615652170924747125041080 9
5507185137084361406290374948975337884073889633379670194520143162505535982480557240745650789504233 9
```

THE 25TH MERSENNE PRIME

Curt Noll, Laura Nickel
California State University at Hayward
Hayward, CA 94542

Abstract: The 25th Mersenne prime is $2^{21701}-1$. The primality was determined with an implementation of the Lucas-Lehmer test on a CDC Cyber 174 computer. The 25th even perfect number is

$$2^{21700}(2^{21701}-1).$$

A Mersenne number is an integer of the form $M_n = 2^n - 1$. If $2^n - 1$ is prime, it is called a Mersenne prime.

A perfect number is a natural number n such that the factors of n less than n sum to n. Euclid showed that if 2^p-1 is prime, then $2^{p-1}(2^p-1)$ is perfect; conversely, Euler showed that an even perfect number must be of the form $2^{p-1}(2^p-1)$ where 2^p-1 is prime [2].

It is easy to show that if M_n is prime, then n is prime; the converse is false. Marin Mersenne (1644) claimed that for $p \leq 257$, 2^p-1 is prime for p = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257, [6]. It was later shown that M_{67} and M_{257} are composite, while M_{61}, M_{89}, M_{107}, which had not been listed by Mersenne, are also prime.

In 1876 Lucas discovered a new way to check the primality of Mersenne numbers. This test was later improved by Lehmer in 1930. The Lucas-Lehmer test for primality is:

Given an odd prime p
let $u(2) = 4$
$u(3) \equiv u(2)^2 - 2 \pmod{2^p - 1}$
.
.
.
$u(n) \equiv u(n-1)^2 - 2 \pmod{2^p - 1}$;
then $2^p - 1$ is prime, if $u(p) = 0$ [4].

It is obvious that this test is much shorter than the usual division test, which is one reason why Mersenne numbers are so tractable.

The 25 known prime M_p's, along with finder and date of discovery, are as follows:

p = 2, 3, 5, 7	around 200 BC	unknown
p = 13	1456	unknown
p = 17, 19	1588	Cataldi
p = 31	1722	Euler
p = 61	1883	Seelhoff & Pervushin
p = 89	1911	Powers & Cunningham
p = 107	1914	Powers & Cunningham
p = 127	1876	E. Lucas
p = 521, 607, 1279, 2203, 2281	1952	Robinson & D.H. Lehmer
p = 3217	1957	Riesel
p = 4253, 4423	1961	Hurwitz &Selfridge
p = 9689, 9941, 11213	1963	Donald B. Gillies
p = 19937	1971	Bryant Tuckerman [1, 2, 3, 5]
p = 21701	1978	Noll & Nickel

The 25th Mersenne prime was found using a CDC Cyber 174 on October 30, 1978 at 9:40 pm. The CPU time used in the test was 7 hours, 8 minutes and 20 seconds. Both Bryant Tuckerman and D.H. Lehmer provided confirmation shortly afterwards [5, 7]. The 6533 digit prime begins 4486 . . . and ends . . . 2751. The perfect number is 13066 digits long, beginning 1006 . . . and ending . . . 5376.

Representation of Mersenne numbers is especially nice in computers because M_n is a string of n ones in base 2. To implement the test, we represented the required numbers in $d = n/47$ digits in base $2^{47} = \beta$.

The squaring of the multiple-precision integers,

$$\left(u = \sum_{i=0}^{d-1} x_i \beta^i\right),$$

is accomplished by the following method.

$$u^2 = \sum_{k=0}^{d-1} x_k^2 \beta^{2k} + \sum_{k=1}^{2d-3} \beta^k \cdot 2 \sum x_i x_j$$

(where the right-most summation runs over all (i, j) such that $i+j = k$ and $0 \leq i < j \leq d-1$). By using both integer and floating point arithmetic we were able to achieve the required 94 bit cross products.

Tuckerman established that M_p for $19937 < p < 21000$ is composite [7]. Wagstaff found thirty-one of the 75 M_p's for prime p, $21000 \leq p \leq 21701$, to be composite by testing possible factors $< 2^{35}$ [8]. We checked the primality of the 44 remaining M_p's and found M_{21701} to be the 25-th Mersenne prime.

There have been several conjectures on the distribution of Mersenne primes. It is not known, for example, whether there are infinitely many Mersenne primes. Mersenne primes do, however, show a tendency to "twin," such as M_{9689} and M_{9941}, and there is some discussion as to whether or not M_{19937} and M_{21701} is another set of twins.

The amount of computation needed to check the primality of M_p is $O(p^3)$. Thus finding a 26th Mersenne prime may be a much more substantial task. It is clear that the major computational effort is in squaring the u(k) (division by $2^n - 1$ is readily accomplished by shifting, and this enables fast division by $2^n - 1$); and it may be possible to implement a faster multiplication method. For example, Schroeppel and Speciner have devised an algorithm that should cut down execution time [3], and Schonhage and Strassen have an algorithm based on fast Fourier transforms [3], which may be promising.

We would like to thank Dr. Jurca, Dr. Lehmer, and Dr. Simon for their help, and Dr. Wagstaff for the use of his tables. We would like to also thank California State University at Hayward for the use of their computer facilities.

References

1. Burton, David M., "Elementary Number Theory" (1976), p. 218-233, Allyn & Bacon, Inc.
2. Kirch, Allan M., "Elementary Number Theory" (1974), p.148, 149, 308, Intext Eductional Publishers.
3. Knuth, Donald E., "Art of Computer Programming, Volume 2" (1968), Section 4.3, Addison-Wesley.
4. Lehmer, D.H., "Lucas-Lehmer test", Journal of the London Math Society, Volume 10, (1935), p. 162-165.
5. Lehmer, D.H., private communication with, University of California at Berkeley, Department of Mathematics, 94720.
6. LeVeque, William J., "Fundamentals of Number Theory" (1977), p. 3, Addison-Wesley.
7. Tuckerman, B., private communication with, International Business Machines Corporation, Thomas J. Watson Research Center, P.O. Box 812, Yorktown Heights, New York 10589.
8. Wagstaff, Jr., S., private communication with, University of Illinois, Urbana, Illinois 61801.

XS-0: A Self-Explanatory School Computer

BY J. NIEVERGELT, H.P. FREI, H. BURKHART, CH. JACOBI, B. PLATTNER, H. SUGAYA, B. WEIBEL, J. WEYDERT
Institut für Informatik
ETH-Zentrum
8092-Zurich, Switzerland

Copyright by Institut für Informatik, ETH, Zurich

ABSTRACT

This report describes a project whose aim is to develop an interactive system that serves as a self-explanatory school computer. Particular attention has been devoted to making the man-machine dialog easy to follow for the inexperienced user. The system includes a course on computer programming, a programming system for writing, editing, executing, and debugging programs interactively, and a filing system containing private and public libraries. The language offered to the user is a version of PASCAL. The system is realized on a small stand-alone computer which supports a small number of graphic terminals. This hardware consists of the most cost-effective components currently on the market.

PREFACE

This paper is an abridged and preliminary version of a report intended as a detailed case study of a self-contained interactive system designed for one particular application. It describes what the XS-0 project is, why it is being done, and how the stated goals have been achieved.

The present paper, consisting of Part I and chapters 1 and 2 of Part II, is an overview of the project and its goals. The rest of the report will cover the following topics:

Part II, written in the style of a user manual, is a detailed description of the behavior of the system. This user manual is not required reading for the users, since XS-0 will contain operating instructions as a part of the system.

Part III is a documentation of the software implementation, with a justification for all the major design decisions.

Part IV is a documentation of the hardware.

PART I: THE XS-0 PROJECT AND ITS GOALS

1. Goals of the Project

The XS-0 project was started in summer 1975 at the Institute for Informatics of the Swiss Federal Institute of Technology, with partial support of the Nationalfond. It has three major goals:

- to provide a focus for research on interactive systems and man-machine dialogs;
- to make a contribution to computer science education and computer literacy, by creating a tool for teaching computer programming which is superior to current practice;
- to develop a low-cost but flexible computer-assisted instruction system, that supports the interactive preparation and delivery of computerized course material on any subject.

1.1 Research on Interactive Systems for the Casual User

We are currently in the midst of a rapid change in the computing environment. The price-drop in computer hardware, which has averaged somewhere between a factor of 2 and 10 every five years, is changing the computer from an expensive institutional piece of equipment to a consumer item. This transformation has occurred in recent years in the field of numerical computation, where a personal pocket calculator often provides better service than a computer center does. Most people have little use for numerical computation, but almost everybody had daily chores of a textual nature that could be eased by a mini- or microcomputer with suitably designed user interface and application programs: handling one's mail and personal notes, filling out forms and writing reports, accumulating a personal data base of relevant facts. In addition, such a personal computer supplied with a rich collection of programs is an attractive device for entertainment and education. A personal computer will obviously be used interactively, and the quality of the dialog it provides is crucial for its acceptance. Many of today's "command langauges" (for example, job control languages) exhibit a primitive level of communication. In order to turn a computer into a personal tool, the

man-machine dialog must be easy to follow, interesting, and informative. The nature of the interaction has to be much better tuned to the novice user and to the casual user than it is in most current systems. Hence, research on man-machine dialogs is at the center of the current trend towards turning a computer into a useful tool for everybody.

The challenge of this project is to develop a system on which novices can find their way easily, without any instruction besides that provided by the system itself; and which remains a convenient tool for the user who has been exposed to it for a while, even if he uses the system only occasionally. There is currently little systematic knowledge relevant to the task of designing effective human interfaces for interactive systems—at best, there are a number of isolated experiences that are relevant. Our aim is to discover and understand the principles that govern the design of an effective communication between a casual user and a machine.

1.2 A Self-Explanatory School Computer

Concerning the second goal, the creation of a superior tool for teaching computer programming, the following considerations apply. Computers will play such a pervasive role in the society of the near future, that some knowledge of the principles on which they operate, and some skill in their use, will soon be required of every educated person. Hence, in order to prepare our children for the society they will have to live in, our schools should teach some aspects of computing to every pupil. Among several reasons why this is not done today, the following are prominent: first, that we have few teachers with the requisite knowledge; second, that computing equipment currently available is costly in relation to the computing budget of most schools; third, that the software and application programs on current equipment have rarely been designed for use in schools, and hence are often inconvenient for instructional purposes. The second reason is rapidly disappearing: one can foresee the day when cost will no longer be a deterrent to the spread of small-scale computation. We want to overcome the other two problems by providing an *ideal school computer*, which has the following properties:

- novices and occasional users can use it conveniently, without any prolonged training;
- it is completely self-explanatory: all the information necessary to use the system is contained within the system;
- it contains a course on programming, from which a motivated user can learn on his own to write sizable programs;
- it supports a modern, high-level programming langauge, and contains a programming system that allows the user to write, edit, file, and execute his programs;
- it is a stand-alone system, usable without link to any other computer;
- it consists of widely available, low cost hardware components.

We hope to have achieved these goals by developing systems software and applications programs that transform a small commercially available computer into a turn-key system designed for the school environment. Our emphasis on low cost has caused us to choose a prototype rather than a commercial product for the graphic terminals of XS-0.

1.3 A Small System for Computer-Aided Instruction

Computer-aided instruction (CAI) has experienced a drastic reorientation over the past decade (see [NIE 75a] for an assessment of the evolution of CAI). During the sixties the field was dominated by psychological/educational theories which emphasized a few rigid teaching strategies, in particular programmed instruction and drill. Courseware was to be prepared by educationalists who were to need no knowledge of computer hardware or programming. Consequently, the technical demands a CAI system had to satisfy were light: often typewriter input and output were deemed sufficient, and course material was typically written in "author languages" whose processing power was severely limited in comparison to general purpose programming languages, so that computational demands on the CAI system were light. Thus CAI was characterized as "electronic page turning" by its critics [OET 69].

Gradually a disillusionment with the restriction to a few rigid teaching strategies set in. The insight gained ground that in many ways the design of a CAI dialog is no different than the design of interactive man-machine communication for other computer applications: CAI lessons began to appear that include extensive simulation models, or permit the user to query a sizable data base. This led to increased demands on the computer resources (terminal, processor, software) necessary to implement an effective instructional dialog, such as: more processing per interaction, immediate response, animated graphic output, and a high level general purpose programming language that allows an author to express complex algorithms.

The XS-0 project wants to show that a good general purpose interactive system with graphic terminals is also a good CAI system. And that a conventional high level programming language which is rich in input/output facilities is a good author langauge—if the author is a good programmer. We consider the widespread view that an author of computerized courseware need not know much about computers to be an illusion: anybody who wants to express himself well should master the medium that delivers his thoughts—in this case, a computer-controlled screen.

2. The System XS-0

Our first realization of the school computer described above is now operational. We call it XS-0, Experimental System—Version 0. This section, an analysis of the requirements, describes the considerations that guided the design. We first characterize the task which XS-0 must be capable of performing, and the users for whom it is intended. Then we explain how our design attempts to match the two: how the system allows our typical user to perform one of the typical tasks.

2.1 Required Performance

2.1.1 Hardware

XS-0 must be realized on a small stand-alone computer system, which is portable in the sense that one person can easily move it from one place to another. It must support a small number (up to about 6) of graphic terminals, that are

located in the immediate vicinity of the computer, typically in the same room. The system must give instantaneous response (0.1 second) to most trivial user requests (those requiring little processing). It must allow on-line storage of sufficient amount of courseware and private files of users.

2.1.2 Software and Applications Packages

XS-0 must provide software for the following user services:

1) A "command language" that poses no difficulties for the novice, yet is also a convenient tool for the user who is exposed to it frequently, in particular the beginning programmer who wants to write, edit, run, and file small programs.
2) A programming system for one higher level programming language suitable for instruction and for writing interactive programs. This includes:
   – a language processor with good diagnostics at compile-time as well as at run-time;
   – an editor;
   – a filing system organized so that it can reflect any structure inherent in the population of users (for example, classes, groups of students within a class, individual users);
   – an on-line reference manual.
3) A course on programming, using the programming language supported by XS-0, including tutorial instruction, exercises, and exams.
4) Facilities to help with the administration of a course (for example, assigning the resources of the system to users, grouping users into classes), and for communication among the users.
5) A public library with a classification system so that interactive programs contributed by users are made available to all users.

All of these services must be provided in a way that an unskilled user can easily understand.

2.2 Types of Users

The system must make it easy for each of the following types of typical users to obtain the services they want.

2.2.1 The Casual User

He approaches the system without any previous instruction; he wants to obtain a general impression of the system, to use some of the programs it contains, and perhaps to be entertained. He needs no passwords or log-in procedure. The system must make it easy for him to move around in the public library, and to use any of the programs contained therein. If he writes a program, it is in order to experiment with the programming system; he does not need to have his private files.

2.2.2 The Student and/or Programmer

He may start as a casual user, but since he uses the system for a prolonged time, he will soon learn the mechanics of getting around in it. From that moment on he wants to communicate in an efficient, low redundancy command language – using only a few key presses to get to any place he wants. He operates in two main modes: either by interacting with an instructional program in the public library or else by writing and running his own programs. In the latter mode he has access to his private files, or to those of a group to which he belongs. He may frequently explore the public library in search of lessons of interest to him, or the register of users, in order to write or read messages.

2.2.3 The Instructor

He coordinates the activities of a class of students. He can leave messages for an entire class or for an individual student, he can enter a student in a class roster so that the student has access to all the files assigned to that class. In practice, the instructor may often be the manager of the system, whose functions are described next.

2.2.4 The Manager

He controls the resources of the system, and updates the public library and the register of users. When a user requests working space, the manager assigns him files of appropriate size. When a new program is added to the public library, the manager classifies it.

2.3 Design Strategy

2.3.1 The Notion of a Self-Contained System

A system is said to be *self-contained with respect to a given group of users and a given set of tasks* if it provides every service that a user in the group requires to perform a task in the given set. XS-0 attempts to be a self-contained system for the casual user who wants to browse, the student who wants to learn and practice programming, and the instructor and manager who need to perform their administrative functions (including the development of instructional material, examinations, etc).

2.3.2 Style of Man-Machine Dialog

A computer with graphics terminal is a medium with entirely different properties than those of conventional media, such as the printed page or film. The combined possibilities of animated graphics, frequent user interaction with immediate response, and access to large amounts of data, allow a style of dialog that is efficient for the trained user, and at the same time convenient for the novice who is learning to use the system. The computer displays a menu of options by means of mnemonically meaningful graphic symbols; the user responds with a single key press, which, if meaningful, accomplishes a lot of work for him. If the key press was illegal or incorrect, the user will notice this immediately, and a more detailed explanation is displayed either automatically, or at his request. Interactive use of computers makes trial-and-error an effective way of man-machine communication; this is in contrast to batch-processing systems, where users often find that a carefully formulated, lengthy input that requests a great amount of processing is the most effective way of communicating with machines.

2.3.3 The User Programming Language PASCAL′

Among high level programming languages, PASCAL is distinguished by its conceptual simplicity and elegance. In

addition, there exist good books on programming based on PASCAL (see [WIR 73] and [WIR 76],), and the language has been used successfully in many places to teach programming. Hence it is well suited as a language for instruction. Since full PASCAL is a rather sizable programming language that contains more features than are necessary for an introductory programming course, and requires a compiler difficult to run on a small machine, we have chosen a subset PASCAL-S [WIR 75] as the basis for the user language of XS-0. However, a language suitable for interactive use of computers, particulary when a graphic terminal is available, needs richer input/output facilities than PASCAL offers. Hence we have added to PASCAL-S facilities for text processing and graphic I/O; the resulting language, PASCAL′, is described in Part II, Chapter 6.

2.3.4 Hardware Configuration

The recent commercial availability of microcomputers makes the XS-0 project economically feasible. For reasons of program compatibility with the PDP-11 family of computers, the LSI-11 based PDP-11/03 is chosen as the main computer, with 28K words of memory, and a dual floppy disk drive. A Texas Instruments 733 ASR Silent, with hard copy output and dual cassette drive, is used as an operator console.

The cheapest graphic terminals available use a TV monitor driven from a refresh memory. XS-0 uses such a terminal developed at the Laboratoire de Calculatrices Digitales of the Swiss Federal Institute of Technology in Lausanne, called SMAKY. It contains an INTEL 8080 micro processor, 8K bytes of random access memory, from which the screen is refreshed, and 4K bytes of read-only program memory. SMAKY is sufficiently powerful that text editing is accomplished entirely within the terminal.

2.3.5 Implementation Language

The software is implemented in a high level system programming language called MODULA [WIR 77], in order to achieve as much machine-independence as possible. Modula permits a convenient specification of concurrent processes, and of structuring a program into a hierarchy of nested modules, such that the variables used by each module are protected from unauthorized access by other modules.

3. The XS-0 Project in the Context of Current Trends

The XS-0 project has aspects in common with a number of areas and research topics that have traditionally been considered to be unrelated to each other; however, all of these areas touch upon problems that are relevant for the design of interactive systems for casual users.

3.1 Trends in Interactive Computing

Interactive computing was historically the first mode of computer use—prior to the development and widespread use of operating systems, applications programmers would operate the computer's console. Soon batch processing systems were introduced for the sake of efficiency, and ever since interactive computing has remained the prerogative of a privileged minority of computer users. It was evident, however, that by removing the computer from the user's immediate control, efficiency was gained only with respect to equipment utilization, but not programmer productivity. Even before the existence of the modern computer, Vannevar Bush had described the visionary system "Memex," that would serve as an extension of its users memory [BUS 45]. In another well known early paper, Licklider speculated on the effectiveness of "man-machine symbiosis," a tightly coupled team consisting of a man and a computer [LIC 60].

During the sixties many research projects attempted to realize the potential of interactive computer use. Among the best known ones are the Culler Fried mathematical laboratory [CUL 68], Sutherland's Sketchpad system for graphic design [SUT 63], and Engelbart's project [ENG 73]. These projects proved that a well designed system, tailor made for a particular application area, is an effective tool that can measurably increase the productivity of an expert user. But interactive systems in the past have been expensive, and hence the transition from research project to production system has occurred only in a few cost-intensive applications, such as computer assisted design ([ALL 77] is a survey of typical applications of interactive systems for CAD). In order to be useful, such systems must provide powerful tools; ease of use, on the other hand, is of secondary importance—a long training required to master such a system may be acceptable.

The recent appearance of microprocessors, and the resulting price-drop in computer hardware, have started a trend towards the computer as a consumer item with the goal of providing portable personal computers with self-contained applications packages for different groups of users. The pocket calculator is merely the first example of this type of product; others, including systems for more demanding applications, will follow rapidly.

With the spread of personal computers, the majority of users will use the computer in an interactive mode again as was true in the early days. But a major condition of usage will be changed: many users will have little or no computer training and will not be willing to learn more about the computer than is absolutely necessary—obtaining the results is all that interests them. Hence the prime consideration in the design of personal computers is ease of use: the quality of the man-machine interface, of the dialog between user and system, is at the center of today's research in interactive computing.

3.2 Programming Languages and Systems Designed for Instruction

For at least the past 10 years a considerable amount of activity has been devoted to programming languages devoted explicitly to instruction. Some early examples are Dartmouth's BASIC [KEM 67] and the LOGO project at BBN and MIT [FEU 71, PAP 70]. APL [IVE 62] has also found a strong following in the educational community. More recent projects of importance to the issue of designing programming languages and systems for instruction include the SMALLTALK project at Xerox Research [GOL 76], and the French project LSE (Language Symbolique pour l'Enseignement) which runs on a MITRA 15 minicomputer in a time-sharing mode.

Among several commercial computer systems marketed primarily to educational institutions we mention DEC's CLASSIC (Classroom Interactive Computer), DEC's PDP-11/03, and Hewlett-Packard's 3000. Most commercial systems offer primarily BASIC as a programming language.

In recent years PASCAL has emerged as perhaps the foremost contender among programming languages suitable for instruction. PASCAL compilers for batch processing are now available for several minicomputers. The goal of XS-0 is to make PASCAL available to the novice programmer on a small interactive system.

3.3 Computer-Assisted Instruction

Concurrently with early research on interactive computing, the first computer-assisted instruction (CAI) systems came into use during the sixties. CAI was primarily introduced for the sake of the following two advantages: the differentiation of curriculum material to fit individual needs, and the possibility of providing immediate feedback to the student. Both of these goals are more easily realized with large numbers of students when a computer is involved than with any other medium. In addition, a great number of students and several schools have been involved in experiments to teach mainly arithmetic and English by means of CAI. There is an extensive documentation about the use of the well known Stanford Arithmetic Program [SUP 68].

Among current projects, the PLATO system developed at the University of Illinois [ALP 70] and MITRE's TICCIT system [BUN 73] are perhaps most widely known. These are large systems (one PLATO system supports about 1000 graphic terminals, one TICCIT system up to 128 color terminals): both are aimed at instruction on a large scale.

Several portable CAI systems have been developed, in the sense that they are not tied to specific hardware; PLANIT [FEI 67] and LEKTOR [SCH 73] are two examples. Such systems usually suffer from poor response time, and often from lack of graphic facilities. THALES [FRE 75] has no graphic facilities as well, but proved that it is possible to develop a viable CAI system with comparatively little effort and expenditure.

XS-0 represents a major departure from many of the existing CAI systems: first, because of its small size and low cost which make it accessible to institutions that need just one or a few terminals; second, because it uses a general purpose programming language (which entails the assumption that the author has considerable programming knowledge) rather than a special purpose author language; thirdly, because no specific teaching strategy is explicitly or implicitly built into the system.

3.4 Desk Top Computers

Commercial desk top computers that offer an applications package for some office activity (for example, accounting or letter writing), but no programming language, have existed for a number of years. More recently desk top computers that offer higher level programming languages, and increasingly also applications packages, have appeared on the market: for example, the Hewlett-Packard 9830 (BASIC), and the IBM 5100 (BASIC and APL). Such systems increasingly assume the function of instructing the user in the operation of the computer, by means of on-line manuals and computer-assisted instruction. The trend is clearly towards making these systems self-contained in the sense that a user needs no facilities other than those contained in the system to accomplish his task.

4. Remarks, References, and Acknowledgments

The design of XS-0 has been heavily influenced by the Automated Computer Science Education System, ACSES [NIE 75a, 75b], that has been developed on the PLATO system [ALP 70] at the University of Illinois. The text processing facilities of PASCAL′ have been influenced by THALES [FRE 74, 75]. The low cost graphic terminals, SMAKY, used by XS-0 were developed by J.D.Nicoud of the Swiss Federal Institute of Technology, Lausanne. The system programming language MODULA, of which an early implementation is used to write the XS-0 software, has been developed by N.Wirth and his group [WIR 77]; we acknowledge in particular the help of J. Hoppe, H.H. Naegeli, V.K. Le of this group. L. Chan, M. Clemens, A. Gorrengourt, and W. Ehinger have helped in developing system software. L. Geissmann, D. Kernberg, W. Ostasiewicz, and H. Schmidli have contributed many lessons to the library of XS-0. We are also grateful to A. Schmitt of the University of Karlsrune for useful discussions about the XS-0 project and on interactive systems in general.

References

[ALL 77] Allan, J. J. (ed.), *CAD Systems,* Amsterdam: North Holland Publ. Co., 1977.

[ALP 70] Alpert, D., and Bitzer, D. L., "Advances in Computer-Based Education," *Science*, Vol. 167 (1970), 1582-1590.

[BUN 73] Bunderson, C. V., "The TICCIT Project: Design Strategy for Educational Innovation," *ICUE Technical Report,* Provo, Utah: Brigham Young University, September, 1973.

[BUS 45] Bush, V., "As We May Think," *Atlantic Monthly,* 176, No. 1 (July 1945), 101-108.

[CUL 68] Culler, G. J., "Mathematical Laboratories: A New Power for the Physical Sciences," in [KLE 68], 355-384.

[ENG 73] Engelbart, D. C., *et al.,* "The augmented knowledge workshop," *AFIPS Conference Proceedings,* Vol. 42 (1973), 9-21.

[FEI 67] Feingold, S. L., "PLANIT—A Flexible Language Designed for Computer-Human Interaction," *AFIPS Conference Proceedings,* Vol. 31 (1967), 545-552.

[FEU 71] Feurzeig, W., *et al.,* "Programming-Languages as a Conceptual Framework for Teaching Mathematics," *Final Report on BBN Logo Project,* June 1971.

[FRE 74] Frei, H. P., "The Author Language and the System THALES," Zurich: *Berichte des Instituts fuer Informatik ETH,* No. 9 (September 1974).

[FRE 75] Frei, H. P., *THALES, an Interactive System and its Application to Teaching Programming,* Zurich: Juris Druck & Verlag, 1975.

[GOL 76] Goldberg, A., *et al., Personal Dynamic Media,* Palo Alto, Calif.: *Xerox Research Report SSL 76-1,* 1976.

[IVE 62] Iverson, K. E., *A Programming Language,* New York: John Wiley, 1962.

[KEM 67] Kemeny, J. G., and Kurtz, T. E., *BASIC Programming,* New York: John Wiley, 1967.

[KLE 68] Klerer, M., and Reinfels, J., *Interactive Systems for Experimental Applied Mathematics,* New York: Academic Press, 1968.

[LIC 60] Licklider, J. C. R., "Man-Computer Symbiosis," *IEEE. Trans. Human Factors in Electronics,* Vol. HFE-1 (March 1960), 4-11.

[NIE 75a] Nievergelt, J., "Interactive Systems for Education—the New Look of CAI," *Proc. IFIP 2nd World Conf., Computers in Education,* Amsterdam: North-Holland Publishing Co., 1975, 465-471.

[NIE 75b] Nievergelt, J., *et al.,* "ACSES, An Automated Computer Science Education System," *Angewandte Informatik 4/75,* (April 1975), 135-142.

[OET 69] Oettinger, A. G., *Run Computer Run,* New York: Collier, 1969.

[PAP 70] Papert, S., "Teaching Children Thinking," *Proc. IFIP World Conference on Computer Education,* (1970), 73-78.

[SCH 73] Schmitt, A., Duerre, K., and Hummel, H., "Entwurf und Realisierung des LEKTOR-Systems," *Lecture Notes in Computer Science,* Vol. 1, Berlin: Springer-Verlag, 1973, 210-218.

[SUP 68] Suppes, P., *et al., Computer-Assisted Instruction: Stanford's 1965-1966 Arithmetic Program,* New York: Academic Press, Inc., 1968.

[SUT 63] Sutherland, I. E., "Sketchpad, A Man-Machine Graphical Communication System," *Proc. Spring Joint Comp. Conf.,* New York: Spartan Books, 1963, 329-346.

[WIR 73] Wirth, N., *Systematic Programming: An Introduction,* Englewood Cliffs, N.J.: Prentice-Hall, 1973.

[WIR 75] Wirth, N., "PASCAL-S: A Subset and its Implementation," Zurich: *Berichte des Instituts fuer Informatik ETH,* No. 12 (June 1975).

[WIR 76] Wirth, N., *Algorithms + Data Structures = Programs,* Englewood Cliffs, N.J.: Prentice-Hall, 1976.

[WIR 77] Wirth, N., "MODULA: A Language for modular multiprogramming," *Software—Practice and Experience,* Vol. 7 (1977), 3-35.

PART II: THE SYSTEM AS SEEN BY THE USER

The first two chapters present an overall view of the XS-0 system from the user's point of view. Subsequent chapters present each major component of the system in detail. They are intended as functional specifications for the implementation, and as a detailed user manual for those users who want information beyond the operating instructions provided by the XS-0 system itself.

1. Structure of the System

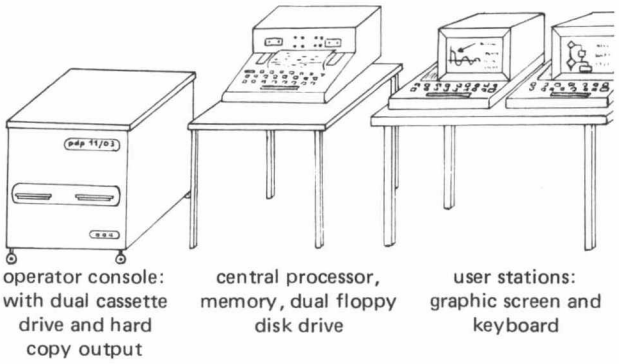
1.1 The Physical XS-0 System

The physical components of the XS-0 system are (see Fig.a.)

- the computer (PDP-11/03) containing a central processor (LSI-11), central memory (28K words of 16 bits), and a dual floppy disk drive;
- the system console and hard copy output device; in the current system this is a Texas Instruments 733 ASR Silent with dual cassette drive;
- a small number (up to 6) of user stations (SMAKY's) each with a graphic screen (TV monotor) and a keyboard; each user station contains a micro processor (INTEL 8080), 8K bytes of random access memory and 4K bytes of read only memory.

1. STRUCTURE OF THE SYSTEM

1.1 The Physical XS-0 System



(Fig. a) Physical components of the XS-0 system

The system is started by inserting a "system diskette" into the slot of the left disk drive, and loading the desired system software. Thereafter a "public diskette" may be inserted into the left slot; it allows users to execute programs contained in the public library (see section 1.3.2). When a "user diskette" is inserted in the slot of the right disk drive, containing the "private libraries" of a group of users, then the users may operate on their own files. A user may also keep private files on a cassette, and read these from the operator console into his private library on the user diskette for processing; he may write any of his files back onto a cassette or print it on the typewriter.

Once the XS-0 system has been started, a user simply needs to know which public and private diskettes contain the modules he wants in order to start working.

1.2 Making a Complex System Understandable: Sites and Modes

While the physical setup of XS-0 is simple, the software that runs on it is elaborate: once the user has turned on the terminal, a multitude of items and services are at his disposal. In order not to confuse the novice user, this multitude must be shown to him only gradually; in order to provide fast service to the expert user, this multitude must be accessible at the press of a few keys. XS-0 attempts to reach these two goals by organizing its collection of items and services in a systematic way based on the concepts of *site* and *mode*. This section explains and motivates this structuring concept. Section 1.3 and 1.4 show how it is used in the XS-0 system.

Many traditional interactive systems have an extremely simple structure, as far as the user is concerned; they are "one-state machines": the user enters a command, the system responds, and returns to the same state it was in before the

command was entered. This structure is sufficient for many simple inquiry systems. It breaks down, however, when the system has to perform several functions that are perceived by the user as being of a different nature. In a combined editing and filing system, for example, the user would like to distinguish a *mode* where he browses through a directory of files, from the mode where he edits a file, because the set of commands applicable in the browsing and the editing mode is completely different. Any attempt to merge these two distinct modes into one, for example by letting the system react to editing and browsing commands at the same time, breeds confusion. The only clear solution is to design the system so that it has distinct modes, and the user is always aware in which mode he is working, and which set of commands is currently active.

Let us now pursue the example of the filing system, and assume that its directory is hierarchically structured: the files belonging to one user form a set, all files belonging to a group of users form a larger set, and so on. When browsing through such a highly structured directory, we must be aware not only that we are in browsing mode, but also, in what region of this hierarchically structured space we are: we must be aware of what *site* we are currently at.

These ideas lead us to structure a complex interactive system according to two independent dimensions. One dimension distinguishes many *sites*, which correspond to different pieces of data that we are operating on; the other dimension distinguishes different *modes*, each mode corresponding to a different set of commands that the user can apply. If the system always lets the user be aware of what site he is at, and what mode he is in, then he knows what he can do at any moment.

The number of different modes of a system should be small, because a mode corresponds to a state of mind of the user, and he cannot conveniently switch between many different states of mind. He is either editing, or browsing, or running a program, or doing one of a *small* number of different things. On the other hand, a system may contain many different sites, i.e., many different pieces of data. It is no burden on the user's comprehension to be aware that a system has a million sites, of which 1000 are of potential interest to him. When he wants to find out what they are, he simply explores the space of sites, that is, he moves around the space while in the browsing mode. When he arrives at a site which is of interest to him, he switches to a mode appropriate to the processing he wants to do at that site.

In summary, our guiding principle for making a complex interactive system understandable to a user is the following: structure your system so that it has as large a number of sites as neccessary to accommodate conveniently all the things it has to offer; and that it has a small number of distinct modes, each mode corresponding to a distinct set of commands available to the user. One of these modes must allow the user to move around, to explore, the space of sites.

1.3 The Space of Sites

The space of sites of XS-0 is hierarchically structured. From the entry point to the system, the root of the tree of sites, the user can reach every site for which he has permission to visit by a sequence of systematic motion commands available in the mode explore. Near the root of the tree are sites that are present in every implementation of the XS-0 system: they are called *predefined sites*. The remaining sites are *user-defined*; they are added by the user community of each physical system.

Different users may have different degrees of permission to operate in the various subtrees of the space of sites. Every user has permission to explore the entire space of sites, except for the private libraries of other users, and to use the programs in the public library. By identifying himself by means of a password, a user may acquire more permissions—for example, the permission to edit in the public library.

1.3.1 Entry into the System

When the XS-o system is in operation, a user who turns on the terminal automatically (without any log-in procedure) finds himself in a mode called "Explore" at a site called "Entry into the XS-0 system". The screen display of this site shows just enough information to tell the novice user how to obtain more information on how to operate the system. The expert user will immediately apply one of the motion commands available in mode explore, and with one key press move on to another site.

The three sites that are directly accessible (sons of the entry site in the tree structure) are called: introduction to the XS-0 system, public library, and user register.

1.3.2 Public Library

The public library is a hierarchically ordered collection of programs that are available to any user of the system. Some of these are supplied with the XS-0 system – they correspond to predefined sites. Others may have been contributed by the user community of one physical system, and added to the public library by the manager of this physical system – a user with permission to edit the directory of the space of sites.

1.3.2.1 Lesson Library

Those programs in the public library that can be executed by themselves to carry on a meaningful dialog with the use are called *lessons*. The XS-0 system contains a collection of lessons on computer programming using the language PASCAL′, a few demonstration lessons on other subjects, and some games. A practically unlimited number of further lessons may be added by a user community.

1.3.2.2 Program Library

The program library contains procedures (subroutines) that are generally useful for writing programs, but cannot be meaningfully executed by themselves. Examples are procedures for analysing text input, for generating graphic output, or for computing numerical functions.

1.3.3 Register of Users

An XS-0 system, in its typical application as a device for teaching and practicing programming in a school, will be used by many different people. This sizable user community will naturally contain a variety of groups: teachers and students, the latter grouped according to classes; within a class one may wish to identify groups of students who work on a joint

project, and hence need to share a set of files. A large user community that shares resources has a need for communication, and the most efficient way for them to communicate is through the system that they jointly use.

The main purpose of the user register is to make such communication easy. Each individual user, and each registered group of users, is represented by a *site* in the user directory, and each such site has a mail box attached to it. For example, if a teacher wants to leave a message for all students in a certain class, he moves to the site of this class and writes a message into the mailbox.

Another function of the user register is to grant permissions to users, and to enforce protection of private libraries from unauthorized access. The private files belonging to an individual user or a group of users are attached to the corresponding site in the user register. They are accesssible, however, only if a user has acquired the permission to access them by typing in a password associated with this site. Notice that a user may accumulate the permissions associated with several sites: for example, the permissions associated with a group to which he belongs, and the permissions associated with his own site.

The register of users is also hierarchically structured: it is a subtree of the entire space of sites. From the point of view of the system the only difference between sites corresponding to groups and sites corresponding to individuals is that in the latter, only one user at a time may acquire the permissions associated with this site, while in the former case, any number of users may do so simultaneously.

1.4 Modes

A *mode* corresponds to a set of commands available to the user. The purpose of this concept is to facilitate the description of which commands available in the system are applicable on which sites, and by whom. On all sites of a similar nature the same set of commands should be active—for example, on all lessons being executed those commands that allow the user to interact with the interpreter must be applicable, and no others. So one introduces a mode for every set of commands that is useful over a wide range of the space of sites, and is then able to describe concisely the command options available to a user by listing the regions of the space of sites where each mode applies.

XS-0 distinguishes three main modes, corresponding to the major activities a user may be involved in:

- exploring and moving around the space of sites;
- editing, that is, making any modification anywhere in the space of sites;
- using a lesson, or more generally, interacting with any PASCAL′ program in execution.

Mode Edit and mode Use are broken into several submodes, as described below.

1.4.1 Explore

From the moment a user enters the system until he leaves it, he is in the hands of the program "EXPLORE": it leads him from site to site, always informing the user of where he is, where he can go, and how to get there. The dialog conducted by EXPLORE must be sufficiently descriptive to inform the inexperienced user of all the options available to him, and sufficiently concise so as not to delay and hamper the expert who knows what he wants.

The main commands applicable in mode EXPLORE are *motion commands.* Structural motion commands, denoted by the arrows ↑, ←, →, and the digits 1 through 9, rely on the tree structure of the space of sites: they move the user from his current site to its father site, left brother, right brother, or the first, second, . . . son, respectively. Other motion commands allow the user to go directly to any site whose name he knows, or to mark the current site he is at, so that later he can jump back to it with the command "return."

The mode EXPLORE is best understood as a set of commands that allow a user to browse efficiently through an automated directory or catalog. When he has found the desired site, the user changes either to mode EDIT or to mode USE, to accomplish his task. When that is finished, he is back in mode EXPLORE at the same site.

1.4.2 Use

The interactions possible between a user and a PASCAL′ program in execution are of two kinds: those which are explicitly written into this particular program, and those which apply to any PASCAL′ program, because they are designed into the system. The latter commands make up mode "USE." Because most PASCAL′ programs written for XS-0 will be written by novice programmers, the mode "USE" allows a rich set of commands for inspecting the status of a program in execution. In the running mode, the program can be interrupted at any time by pressing the HALT key. The user can then enter a submode called "VIVISECTION," in which the current value of any program variable can be displayed. Or, in a submode called "SPECIFICATION," he can specify that the program is to be executed step by step, or that certain variables should be traced. By reentering the running mode, program execution is resumed.

1.4.3 Edit

All the changes a user can do to XS-0 are considered to be editing operations, and are done in mode "EDIT." This includes editing a file in a user's private library; reading one's mailbox and deleting the messages read, or writing a message in someone else's mailbox; adding or deleting entries in the user register or in the public library, that is, creating new sites or deleting old ones; changing the information in a site; copying a file or compiling, which creates a new file.

Because of the great variety of editing operations, mode EDIT is broken into several submodes, each of which is a relatively small, coherent set of commands: text editing, graphics editing, program editing, copying, message reading and writing, site editing. Since editing operations can have drastic consequences, the permission to edit is granted sparingly—most users have permission to edit only on their private library, and on most mailboxes.

2. Sample Dialogs

The sample dialogs described in this section are intended to convey a more concrete picture of how different users might interact with XS-0. They also illustrate some general principles of dialog design that are used as guidelines for all interactive software of XS-0. Section 2.1 describes these principles.

2.1 The Art of Designing Effective Dialogs

Every medium requires its own style of communication. For a medium as novel as a computer controlled screen, there is as yet little experience to guide the design of interactive programs that carry on informative, easy to understand dialogs. However, a few principles are beginning to emerge; the following ones are used as guidelines for designing man-machine dialogs in the XS-0 system.

P1: Whenever possible, let the user control the dialog (that is, decide what should happen next) rather than the program. We assume XS-0 users are motivated to explore the system up to its limit of performance. So at any moment, many options should be open to a user; in particular, the option to change the entire context of the discussion. This encourages trial-and-error behavior on the part of the user. If the principles P2 and P3 are followed, trial-and-error can be the most effective way of communicating.

P2: Make sure the user can at all times get a quick answer to the questions:
– where am I?
– what can I do here?
– where can I go from here?
– how do I get there?
If this principle is followed, the user will never feel lost in a maze, even after having entered a command he did not understand (see the discussion on trial-and-error under P1).

Dialogs in XS-0 follow certain conventions that make it easy for the user to answer these questions. The current site and mode are displayed at the top left of the screen. The bottom line often reminds the user of his most important options, while the HELP key lists all commands active at this moment. Mode EXPLORE displays a map of a relevant portion of the space of sites.

P3: Let the dialog be highly interactive; that is, let many short inputs and outputs alternate at high frequency. Whenever possible, let the input consist of a single key press. If the system responds instantaneously (0.1 sec) to commands that require little processing, it is often preferable for the user to achieve a given effect by entering several one-key-press-commands, rather than one command several characters long. This facilitates trial-and-error: if a command is wrong, the smaller the command, the lesser is the resulting confusion. An equally important consequence is that command languages that need not contain long commands tend to be simpler (see P4).

P4: Command languages must be as simple as possible: there should be few commands, and their names must be short and mnemonically meaningful. The most important commands should be implemented as labeled keys on the keyboard.

P5: Communication from the system to the user must have a certain amount of redundancy. It is not enough to say something important only once in the course of a dialog; it must be repeated from time to time.

P6: Pictures and examples, in particular dynamic ones, convey information faster than verbal descriptions. Even if they do not express everything that could be said about the topic under discussion, in a fast-moving dialog it is better to convey the essential part of an idea quickly, leaving the rest to be guessed, than to specify every detail at the cost of the user's time.

P7: Design the dialog so as to use the entire area of the screen effectively. Don't use the screen as a scroll, where only the last few lines carry information that is still relevant, while the rest is cluttered with input and output left over from earlier interactions that may not pertain to the current task. Use the screen as an artistically designed animated page. A good screen layout is balanced, uncluttered, and attracts the eye to one spot where the currently most important information is.

Undoubtedly there are situations where these principles are not appropriate. We do not follow them dogmatically, but we believe that they are good guidelines, particularly for systems aimed at casual users, and that one should not deviate from them without good reasons.

2.2 Exploring

Upon entering the system the user is in mode EXPLORE. EXPLORE provides different kinds of dialogues for different kinds of users.

The novice who is using XS-0 for the first time must be guided through the system so that he never feels lost. He will be grateful for some redundancy in the man-machine dialog, such as repetition of the most important information, and complete sentences instead of abbreviations.

The student/programmer starts out as a novice and soon becomes an experienced XS-0 user who is able to use the entire set of commands, and wants to enter these with few key presses.

The installation manager who registers students and assigns files to them is only interested in executing his tasks as quickly as possible.

All these different users enter the system in the same way. On the left part of the screen the user gets a menu of the successor sites he may select. On the right part of the screen there is information which is particularly useful for the casual user. (Fig. a)

```
MODE EXPLORE   SITE X50   XS  Top of the hierarchy of sites

NO ABSTRACT MODULE          SHORT ABSTRACT INFORMATION:

NO ALLOCATED MODULES        If this is your first visit,
                            press 1.
SUCCESSOR SITES ARE:        To go to the PASCAL' course,
1. INTRO                    type G PASCAL.
2. PUBLIB                   For entertainment type
3. USERREG                  G DEMO.

↑
```

Fig. a) Entry to the system

After this common entry point the subsequent dialog is likely to differ depending on the type of user and his goal.

2.2.1 Casual User

The text-information contained on the right part of the screen is intended for the novice. Let us assume the user types '1', the reaction recommended by the system. This causes a motion to the site 'Intro'. (Fig. b)

```
MODE EXPLORE   SITE INTRO   INTRODUCTION TO THE SYSTEM XS0

NO ABSTRACT MODULE          SHORT ABSTRACT INFORMATION:

ALLOCATED MODULES:          Press 0 for the most basic
4 MODES XS  C               information.
                            Press 4 for a pictorial
SUCCESSOR SITES ARE:        description of modes.
1 EXPLORE                   More detailed information
2 EDIT                      available at successor sites.
3 USE

GOTO INTRO
```

Fig. b) Moving by selection

Again the user gets some information and decides to type '4' according to the recommendation given on the screen. This causes the mode USE to be activated and the corresponding lesson to be executed. Even if the user does not understand these first key presses he will learn the concepts of sites and modes and recognize what happened. After the termination of the lesson the user is automatically returned to mode EXPLORE.

If the user is familiar with the command set he may type 'D' (for DISPLAY), which changes the view of the current site. Now a part of the space of sites is displayed on the screen. The current site is marked and located in the middle of the screen. (Fig. c)

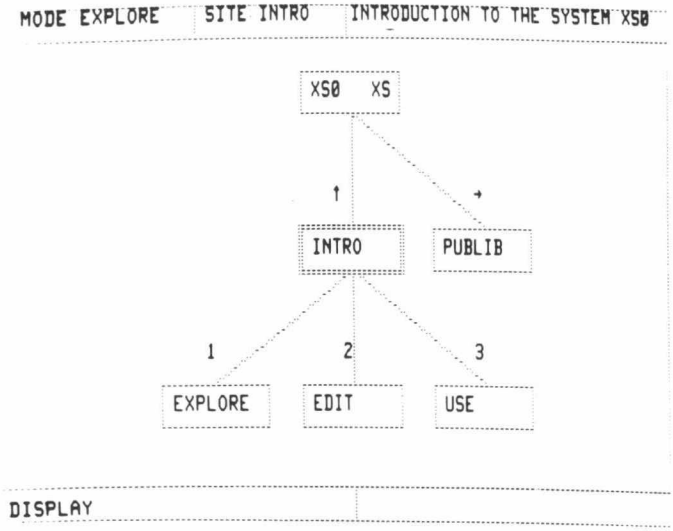


Fig. c) Graphical view of neighbourhood

The user decides to go to the right neighbour site, the public library, and types '→'. He explores the library by moving from site to site with the "arrow-commands" ↓, ←, →, ↑ and with the selectors 0, 1, . . . , 9. He finds a site called 'Demo' (Fig. d).

```
MODE EXPLORE   SITE DEMO   ENTERTAINMENT

NO ABSTRACT MODULE          SHORT ABSTRACT INFORMATION:

NO ALLOCATED MODULES        Experiments designed to show
                            the power of this new medium:
SUCCESSOR SITES ARE:        *** the computer-driven  ***
1 MUSIK                     ***          screen      ***
2 SPIELE
3 WITZ

5
```

Fig. d) A site as entry point into a collection of related material

He reads the text, selects the most interesting demonstration program and changes to mode USE to run it. After the lesson is terminated he decides to end the session. For the casual user there is no log-out procedure necessary. He just stands up and leaves the terminal.

2.2.2 The Student/Programmer

In the following we see student Miller working with XS-0. Upon entry to the system he moves inmmediately to his "home site" with the command 'G' (for goto) 'Miller' (see Fig. e). There he enters his password with the command 'P'. The system responds with 'Permissions allocated.' This gives him all the privileges associated with his site, in particular, permission to edit the modules attached to this site. Miller currently owns three modules: a mailbox module, where other users can leave messages for him, a text module, called 'Exercis1' where he has written a source program, and a code module of the same name which contains the compiled program.

```
MODE EXPLORE   SITE MILLER   MILLER JOHN / SEMESTER 2

NO ABSTRACT MODULE          SHORT ABSTRACT INFORMATION:

ALLOCATED MODULES:          ADRESS:
1 MAILBOX   M               -------
2 EXERCIS1  T               Clausiusstr. 55
3 EXERCIS1  C               8007 Z U E R I C H

NO SUCCESSOR SITES          PHONE NO:
                            ---------
                            01/242 21 33

GOTO MILLER
```

Fig. e) Home site of student Miller

Miller selects the text module and thus changes to mode EDIT. He continues writing a PASCAL′ program. Suddenly, he has the problem whether there exist pointers in PASCAL′. He may get this information by using the PASCAL-INFOR–MATION-SYSTEM (see section 2.5). Therefore he types 'X' and returns to mode EXPLORE. Since he wants to return to site 'Miller' later he marks it with the command 'M', and jumps to the information system with the command 'G' 'Pas'. Later he returns to the stacked site 'Miller' by typing the command 'R'.

When Miller has finished editing his program he asks for compilation. If compilation is successful the code module with the same name contains the newly compiled code.

To run the program he simply selects this code module; this causes the mode to be changed to USE (see 2.4). As the program works satisfactorily he returns to mode EXPLORE, and logs out by keeping his finger pressed on the key '↑', which corresponds to repeated execution of the command "move one level up in the tree." By pressing '↑' one last time at the root of the tree he logs out, and the permissions allocated to this terminal are withdrawn.

2.2.3 The Manager

After entering the system the manager gives his password. He will then have the permission to change all the information which does not belong to a predefined site. Then he will move to the site where he wants to change the descriptor space and will switch to mode EDIT by typing 'E'. All the changes are done by the site-editor (see 2.3.2).

After return to mode EXPLORE, he will logout or move to another site to proceed with other changes.

2.3 Editing

2.3.1 Text-Editing

Assume student Miller wants to modify a program he has written earlier. He goes to the site MILLER which is associated with his name, and there he selects the text module called EXERCIS1. This causes a change of mode from EXPLORE to EDIT, and a choice of editors to be displayed on the screen. The student selects the text-editor; instantly the top of the text-module EXERCIS1 appears on the screen (Fig. a).

```
MODE  TEXT-EDIT   SITE: MILLER    MODULE: EXERCIS1
-----------------------------------------------------------
  PROGRAM TOWERSOFHANOI(INPUT,OUTPUT);
  (*THIS PROGRAM PRESENTS THE PROBLEM OF THE TOWERS OF HANOI,
    IN A FIRST PART THE SOLUTION OF THE PROBLEM IS SHOWN ON
    THE SCREEN, IN A SECOND PART THE USER IS OFFERED THE
    POSSIBILITY TO DISCOVER THE ALGORITHM *)
  CONST MAXHEIGHT=5; (*MAXIMUM HEIGHT OF THE TOWERS*)
  TYPE TOWER=RECORD
               DISK: ARRAY[1..MAXHEIGHT] OF INTEGER;
                                    ↑
               HEIGHT: INTEGER;
             END;
  VAR ITEM: ARRAY['A'..'C'] OF TOWER;
      (*3 TOWERS AT LOCATIONS A,B AND C*)
      COMMAND: STRING(10);
      OPTION: CHAR;
      COUNTER: INTEGER;
-----------------------------------------------------------
```

Fig. a) Window onto a text module

First the student quickly scans the entire program. For this purpose he moves the cursor over the text. The special keys for cursor-motion (↑, ↓, →, ←) can be used in a repetitive way: holding the key ↓ makes the cursor move continuously down on the screen. As soon as the cursor "tries to leave" the screen, the text is rolled upwards.

The student, after having scanned the entire program, wishes to modify the procedure MOVEONEDISK. He enters the command 'F' (find), types the string MOVEONEDISK and the editor finds the first occurence of this string, searching the text from the current cursor position in a circular way (Fig. b).

```
MODE  TEXT-EDIT   SITE: MILLER    MODULE: EXERCIS1
-----------------------------------------------------------
  PROCEDURE HANOI(A,B,C:CHAR; N:INTEGER);
  (*MOVES A TOWER FROM LOCATION A TO LOCATION C,
    USING LOCATION B*)
      PROCEDURE MOVEONEDISK(LOC1,LOC2:CHAR);
                      ↑
      BEGIN
        TRANSFER(ITEM[LOC1],ITEM[LOC2]); DRAWTOWERS;
      END;
  BEGIN
    IF N>0 THEN
    BEGIN
      HANOI(A,C,B,N-1);
      MOVEONEDISK(A,C);
      HANOI(B,A,C,N-1);
    END;
  END (*HANOI*);
-----------------------------------------------------------
```

Fig. b) Character string has been located

Now he wants to change the name of the procedure from MOVEONEDISK to SIMPLETOWER. Keeping the BACK SPACE-key depressed he watches the characters of the name disappear one by one. By typing 'I' he opens the insert-command and directly inserts the new name of the procedure.

Having renamed the procedure, he wants to move it further up in the program text. To do this the cursor is positioned below the last line to be moved. The command 'M' (move) then creates a subcursor which is positioned above the first line of the procedure. Closing the move command deletes the lines bracketed by the two cursors from the text and saves them into a buffer.

Since the student does not remember the command for transferring the contents of the buffer into the text, he presses the HELP-key (Fig. c).

```
MODE  TEXT-EDIT   SITE: MILLER    MODULE: EXERCIS1
-----------------------------------------------------------
              4 OPERATIONS ON BUFFER

  S: SAVE LINES
     CREATES A SUBCURSOR (DOTTED LINE), WHICH MAY BE MOVED
     BY THE CURSOR-MOTION-COMMANDS, NEXT CLOSES THE COMMAND
     AND COPIES THE LINES BETWEEN THE CURSOR AND THE SUB-
     CURSOR INTO THE BUFFER
  M: MOVE LINES
     HAS THE SAME EFFECT THAN S; MOREOVER THE LINES SAVED
     INTO THE BUFFER ARE DELETED FROM THE TEXT
  C: COPY BUFFER
     CAUSES THE CURRENT BUFFER CONTENTS TO BE COPIED ABOVE
     THE ACTUAL CURSOR POSITION INTO THE TEXT

                    -PRESS NEXT-
-----------------------------------------------------------
HELP
```

Fig. c) HELP displays a list of commands

By typing '↑' he resumes text editing in the same situation as before. He now positions the cursor at the correct place and gives the command C to copy the procedure saved in the buffer into the text.

Finally the student leaves the text-editor; he sees a menu of all submodes of mode EDIT, and selects the compiler to compile his program.

2.3.2 Site-Editing

The site-editor can only be used by the management to change the descriptor space. This is guaranteed by the built-in protection mechanism. The use of the site-editor is demonstrated by 2 examples which are typical for management tasks.

Example 1:

Let us assume that the manager has to register a new student, say Miller. He shall be a member of class 1. In mode EXPLORE he will then move to the site 'Class 1' and by typing 'E' switch to mode SITE-EDIT (Fig. a).

```
MODE: SITE-EDIT              SITE: CLASS1
------------------------------------------------------------

Abstract:
    0  CLASS 1          T

Allocated modules:
    1  Cl.Mail          M
    2  Cl.Text          T

Successor sites:
    3  Forster
    4  Hall
    5  Smith
    6  Williams

------------------------------------------------------------
EDIT
```

Fig. a) Editing a site in the user register

Now the authorized user (manager) can type a site-editor command. In this case he types 'N' for 'Newmodule.' Instantly the screen changes and he is asked to enter the necessary information, that is, the name of the new module (Miller) and its type ('D' for descriptor).

If the module has been created successfully the system answers by the message 'Module created', otherwise an error message is issued. Let us assume that the creation was possible which means that the new descriptor is automatically linked into the descriptor space as fifth successor site of the site 'Class 1.' By typing 'X' the manager now switches to mode EXPLORE, moves to the new site 'Miller' (for instance by typing '7') and again switches to mode SITE-EDIT to continue the registration of Miller (Fig. b).

```
MODE: SITE-EDIT              SITE: MILLER
------------------------------------------------------------

No abstract module

No allocated modules

No successor sites

------------------------------------------------------------
EDIT
```

Fig. b) View of the new site

Upon creation of a descriptor module only the linkage to the neighbor nodes is done automatically. Other descriptor fields must be filled explicitly. If the manager types 'F' for 'Fill' the system reacts by asking for the corresponding information, in particular a descriptive title for this site, a short abstract, and a password. Any of these fields can be left blank or changed later.

The last step in the registration of Miller is the creation of private files (Text- and mailbox modules). The manager types 'N' for 'Newmodule' and then specifies the names, types and sizes of the modules to be attached to this site. When he is finished the display of this site looks as in Fig. e), Section 2.2.2.

With the command 'X' he leaves the site-editor and returns to mode EXPLORE. He can now move to other sites, adding or deleting users and their files.

Example 2:

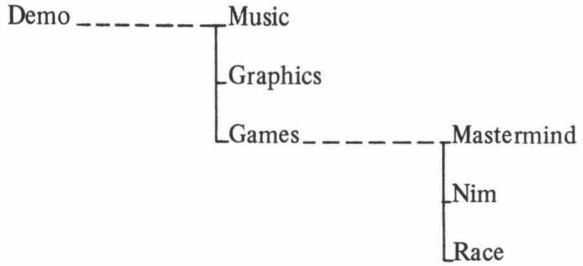
Assume the manager wants to restructure parts of the lesson library.

Among several successors of the site 'Demo' there are some games, for instance Mastermind, Nim, Race. Now these games shall be grouped as successor nodes of a new site'Games'. The manager moves to the site 'Demo' and creates the new descriptor 'Games'. Then he types 'S' for 'Subtree'; the system asks for the node which will be the root of the new subtree, and for the numbers of the first and the last nodes to be grouped under the new site (Fig. c).

```
MODE: SITE-EDIT     SITE: DEMO              ENTERTAINMENT
------------------------------------------------------------

Successor sites:
    1  Music
    2  Mastermind
    3  Nim
    4  Race
    5  Graphics
    6  Games

------------------------------------------------------------
SUBTREE   ROOT 6   FIRST 2   LAST 4
```

Fig. c) Before building a subtree

The result of this editing operation is to change the structure of the subtree rooted at site 'Demo' as follows:



The previous state of the tree can be restored by "killing" the new site 'Games'. Another command allows permuting subtrees of a common site, so that the manager can easily give the tree any shape he wishes; but the space of sites always retains a tree structure.

2.3.3 Graphics editing

If a user wants to write a program that draws pictures on the screen, his task will be greatly simplified if he uses the graphics editor. In this mode he can draw a picture and enter text on the screen by moving a cursor (Fig. a).

```
MODE GRAPHICS, DRAWING          SITE GORRENGO   MODULE TONI7
```

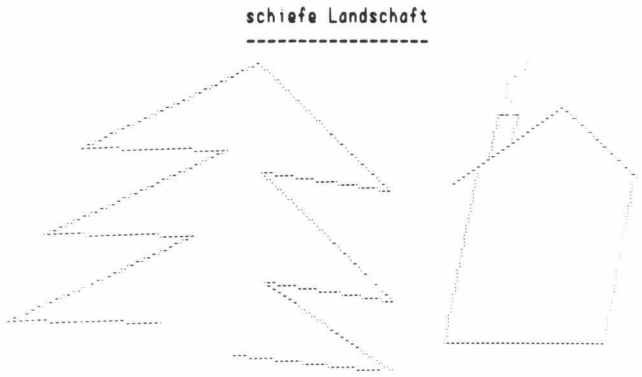


Fig. a) Picture created with the graphics editor

An internal representation of this picture is saved in a graphics module, so the picture can be displayed and further edited at any time. The picture can also automatically be translated into a part of a PASCAL′ program that draws this same picture when executed (Fig. b). The statements thus generated can be inserted into a text module and combined with another program.

```
MODE: PROGRAMS   SITE: GORRENGO   MODULE: TONI1 XS
BEGIN
  DRAW(204,79); DRAW(202,80); DRAW(202,82);
  DRAW(203,84); DRAW(206,86); DRAW(210,88);
  DRAW(211,90);
  DRAW(66,16,14,18,79,41,27,43,92,66,40,68,105,91,155,53,
       105,59,155,21,105,27,155,0,92,6);
  DRAW(190,58,175,9,239,9,254,58);
  DRAW(255,57,225,77,180,55);
  DRAW(195,63,199,75,207,75,204,68);
  WRITEAT(5,24,'schiefe Landschaft');
  WRITEAT(6,24,'------------------');
END.

PROGRAMS
```

Fig. b) Statements generated from picture

2.4 Using a Program

Using a program in the XS-0 system means executing intermediate code (contained in a "debug module") of a user program compiled with an option that makes available a rich set of run-time diagnostic facilities. In order to demonstrate the full capability of mode USE, let us assume a not yet tested user program is to be executed. The name of the corresponding code module is DEBUG XS. The user has selected this module from his private files, and, in order to see what options he has, he presses HELP (Fig. a).

```
MODE USE        SITE: WEIBEL     MODULE: DEBUG XS
The current mode is USE.

USE-COMMANDS:
-------------

X (=EXPLORE):   Return to EXPLORE.

R (=RUN):       Restart the program running before.

P (=PROCEED):   Continue at the break point if the program
                  was interrupted by pressing the HALT-key.
V (=VIVISECT):  Enquire the actual values of the variables of
                  your program.
S (=SPECIFY):   Specify other modes of execution (slow motion,
                  step mode, tracing of variables)
HELP:           To help you.

HELP                              TYPE USE COMMAND
```

Fig. a) Active commands of mode USE

He presses 'R' to run his program. The screen now changes and becomes the output screen for his program. As he watches the drawing produced by his program he notices immediately that the lines are somewhat out of place, as if some counters in his program had incorrect values. He presses the HALT key. This suspends execution of his program, and the screen changes. The program output screen is saved, so that execution can be resumed later with precisely the same state of program variables and output screen. The user sees the "program monitor screen" (Fig. b) which informs him that his program was stopped in a procedure called 'GRAPHIK' at statement 37.

```
MODE: USE        SITE: WEIBEL     MODULE: DEBUG XS

Halt at   37 because of HALTKEY
   GRAPHIK  CALLED AT  505



No step mode
No address traps

No traced variables

RUN                               TYPE USE COMMAND
```

Fig. b) Program monitor screen after interruption

By inspecting the listing module of this program the user would be able to identify statement 37. However, he knows from the current state of the output screen (which he can inspect at any time by flipping back and forth between the two screens) where his program was interrupted, and he wants to know the current values of his two counters A and B. By pressing 'V' he enters mode "VIVISECTION", where he can ask for the value of any variable currently active (Fig. c).

```
MODE VIVISECT   SITE: WEIBEL     MODULE: DEBUG XS
-----------------------------------------------------------

A       =       44 IN GRAPHIK
B       =       66 IN GRAPHIK
ZZZZZ     There is no such variable !
Which variable ? --------



No traced variables



-----------------------------------------------------------
VIVISECT
```

Fig. c) Inspecting variables in mode vivisection

Our user would like to see how his counters A, B, and another variable, CH, change in this portion of his program. He enters mode "SPECIFY", where he is given a wide variety of options about how his program is to be executed (Fig. d).

```
MODE SPECIFY    SITE: WEIBEL     MODULE: DEBUG XS
-----------------------------------------------------------

        0:  Return to mode USE

        1:  Specify step mode
        2:  Specify address traps
        3:  Specify variables to be traced

        4:  Clear step mode
        5:  Clear single address traps
        6:  Clear single traced variables
        7:  Delete all address traps
        8:  Delete all traced variables

Step mode:  Halt after how many steps ? 1___

-----------------------------------------------------------
```

Fig. d) Setting step size in mode specify

Since he wants to watch the execution of his program closely from this point on, he specifies that each statement is to be executed only when he presses 'P' for 'proceed' (if he wants to execute his program a little faster, he could specify that for each press of 'P', n statements are executed). He also specifies that variables A, B and CH are to be traced.

Now he switches to the program output screen, and watches the output as statement after statement is executed at the press of a key. As soon as one of the traced variables changes value, the program monitor screen appears automatically, and the current values of all the traced variables are displayed (Fig. e).

```
MODE: USE       SITE: WEIBEL     MODULE: DEBUG XS
-----------------------------------------------------------

Halt at   52 because of HALTKEY
   GRAPHIK  CALLED AT  505



Step mode:  Halt after    1 steps
No address traps

Traced variables:
A       =       44 IN GRAPHIK     B       =       66 IN GRAPHIK
CH      =        1 IN MAIN PROG

-----------------------------------------------------------
                                        TYPE USE COMMAND
```

Fig. e) Notification that one of the traced variables has changed value

Thus by always having available at the press of a key either a view of the program's output screen or a view of the program monitor screen, which gives access to the entire internal state of the program, the user is greatly helped in debugging a program.

2.5 Using the Pascal Information System

The Pascal Information System PAS serves as an on-line manual for the programming language PASCAL′ and as an index to the PASCAL lessons of the XS-0-system. PAS is a program in the public libarary and therefore is accessible to any user. Upon entering PAS (Fig. a), the user can elect to see an introduction on how to use the system. This introduction presents the following information:

The system PAS contains a vocabulary of about 300 phrases which have to do with PASCAL′, and several classifications of these phrases.

For each phrase PAS provides information which consists of some or all of the following parts:

- Syntax diagram (if the phrase is a nonterminal symbol of the grammar of PASCAL′).
- Explicit remark
- Reference to a classification
- References to lessons
- References to other phrases of the vocabulary.

PAS is designed to have 4 different sites in which different functions are performed:

- a site "Q" where you can ask a question by entering a phrase,
- a site "A" where the answer is given, that is, information that belongs to the specified phrase,
- a site "V" where a section of the alphabetically ordered vocabulary is shown, and
- a site "C" where a classification is presented.

```
                                                PAS        INTRO
-------------------------------------------------------------------

                    ====================
                        P A S C A L
                    I N F O R M A T I O N
                        S Y S T E M
                    ====================

If you need an introduction on how to use this program,
press I !

Otherwise press one of the keys Q, V, C for getting
to the sites "Q", "V", "C" respectively

-------------------------------------------------------------------
Press Q to enter a phrase;          I for introduction;
press V to look at the vocabulary.  C for classification
```

Fig. a) Title page with instructions

```
procedure                                       PAS        ANSWERS
-------------------------------------------------------------------

There exists a classification of special symbols
If you want to see it, press C

Reference to lessons
      1 6       KONZEPT
      1 6 1     PROZEDUR

See also      1  procedure declaration
              2  procedure call
              3  standard procedure
              4  recursive procedure
              5  procedure heading
              6  procedure identifier
              7  formal parameter section
-------------------------------------------------------------------
Press Q to enter a phrase, a digit for one of the phrases above,
press V to look at the vocabulary   C for classification
```

Fig. b) Information about "procedure"

```
procedure                                       PAS        CLASS
-------------------------------------------------------------------

Special symbols
====================
Operator symbols      +  -  *  /  =  <>  <  <=  >=  >  =

Delimiter symbols       ,  ,  :  '      (  )  [  ]  (*  *)

Word-delimiters (Reserved words)
      AND         DO          GOTO        OR          TO
      ARRAY       DOWNTO      IF          PROCEDURE   TYPE
      BEGIN       ELSE        LABEL       PROGRAM     UNTIL
      CASE        END         MOD         RECORD      VAR
      CONST       FOR         NOT         REPEAT      WHILE
      DIV         FUNCTION    OF          THEN

-------------------------------------------------------------------
Press Q to enter a phrase,
press → or ← to see another classification;  R to return
```

Fig. c) Classification of "special symbols"

Assume that a student wants to inquire about procedures in general, and about the syntax of procedure declarations in particular. Therefore he presses 'Q' (Fig. a) and types the word "procedure". The system answers the information shown in Fig. b).

Since the student wonders how that classification of special symbols looks, he presses 'C' (Fig. c). He realizes that the classification is not in his area of interest, and he returns to the phrase "procedure" by pressing 'R' (Fig. b).

The student observes that there are references to several phrases which contain the word "procedure", such as "procedure declaration", "procedure call", "standard procedure" and so on. Among these, he is most interested in "procedure declaration", but let us assume that he first wants to see all phrases of the vocabulary which begin with the word "procedure". By pressing 'V' he obtains Figure d), then by pressing NEXT he comes to Figure e).

```
procedure                                       PAS        VOCABULARY
-------------------------------------------------------------------

    1  plus
    2  pointer
    3  pred
    4  predicate
    5  procedure
    6  procedure and function declaration part
    7  procedure as parameter
    8  procedure call
    9  procedure declaration
    0  procedure heading

-------------------------------------------------------------------
Press Q to enter a phrase; a digit for one of the phrases above,
press → or ← to see another vocabulary section;  R to return
```

Fig. d) Section of the vocabulary containing "procedure"

```
procedure                                       PAS        VOCABULARY
-------------------------------------------------------------------

    1  procedure identifier
    2  procedure or function declaration
    3  procedure statement
    4  program
    5  program heading
    6  program parameter
    7  programming language
    8  rdraw
    9  read
    0  readspot

-------------------------------------------------------------------
Press Q to enter a phrase; a digit for one of the phrases above,
press → or ← to see another vocabulary section;  R to return
```

Fig. e) Next section of the vocabulary

By pressing 'R' the student returns to the phrase "procedure" (Fig. b), and then by pressing '1' he gets to the phrase "procedure declaration" (Fig. f).

```
procedure declaration                          PAS        ANSWERS
-------------------------------------------------------------------

        ----+---1procedure heading---+---2block---+----

Reference to lessons :
      1.2.4.1  DEKLARAT
      1.2.4.6  PROZDEKL
      1.6.1    PROZEDUR

See also :    3  procedure call
              4  procedure statement
              5  parameter
              6  declaration part

-------------------------------------------------------------------
Press Q to enter a phrase; a digit for one of the phrases above;
press V to look at the vocabulary.
```

Fig. f) Information about "procedure declaration"

This display shows that a procedure declaration consists of a procedure heading and a block. The student postpones looking at "block" until later, and proceeds to "procedure heading" by pressing '1' (Fig. g). For further details he presses '2' (Fig. h).

```
procedure heading                              PAS        ANSWERS
-------------------------------------------------------------------

        ----+---PROCEDURE---+---1identifier---+---2parameter list---+---;----

Reference to lessons :
      1.2.4.6  PROZDEKL
      1.6.1    PROZEDUR

See also :    3  procedure and function declaration part
              4  parameter

-------------------------------------------------------------------
Press Q to enter a phrase; a digit for one of the phrases above;
press V to look at the vocabulary.
```

Fig. g) Information about "procedure heading"

Continued from pg. 28

Another great feature of the 1802 is not the 1802 itself, but one of its support chips, the 1861. The 1861 works right along side the 1802. since they were made for each other. It is a graphics chip, which displays a portion of memory as a series of light or dark spots on a video monitor or modified TV screen. The graphics are bit controlled, each bit of each byte corresponding to a position on the screen. Because of this high density you can display up to 64x128 resolution pictures using the 1861, and it only takes up 1K of ram. All three of the single board computers have it built in. In case you are interested, they are the ELF II, the Super ELF and the VIP, made by Netronics, Quest Electronics, and RCA respectively. If you are interested in an 1802-based computer, they are the ones to look at. If you are a homebrewer, you might want to get the Studio II video game from Radio Shack. It uses an 1802, and if you want to modify it to be used as a computer, it is a good deal at only $60.

```
parameter list                                 PAS        ANSWERS
-------------------------------------------------------------------

        ----+---(---+---1formal parameter section---+---)---+----
            |       |                               |       |
            |       +-------------;-----------------+       |
            +---------------------------------------------- +

Reference to lessons :
      1.6    KONZEPT
      1.6.3  PARAMET

See also :    2  procedure and function declaration part
              3  parameter

-------------------------------------------------------------------
Press Q to enter a phrase; a digit for one of the phrases above,
press V to look at the vocabulary.
```

Fig. h) Information about "parameter list"

The phrase "parameter list" refers to a lesson called "paramet". The student decides not to pursue the complete syntactic specification of "formal parameter section" in this on-line manual, but instead to learn about parameters by entering this lesson.

2.6 Comment

These photographs and verbal descriptions cannot convey the full flavor of a dialog with a highly interactive system, which provides instantaneous response to most key presses, and (unlike the linear sequence imposed by paper) lets the user branch out in any of a number of directions at any moment. We hope, however, that these dialogs give an impression of the variety of services provided by XS-0, and of the ease of using them.

the end

CONCLUSION

The 1802 is a very interesting processor. If you like to experiment, it may be just what you were looking for. Perhaps you don't own a computer—the Cosmac computers on the market are very well priced (you can get the ELF II or Super ELF for around $100 in kit form, which is not bad). In general, I think you will hear even more about the 1802 as more and more computer hobbyists and manufacturers discover it. It has quite a future, so don't overlook it.

Karl Mosgofian is 16 years old and attends San Francisco's Washington High School. He has been a computer hacker for two years, has worked in a computer store and in a science museum. He hopes to someday become a professional writer. This article is a good start. – SMR

SELECTING BUSINESS SOFTWARE FOR MICROCOMPUTERS

BY GREG B. SCOTT
Micro Applications Group
7300 Caldus Ave.
Van Nuys, CA 91405

This commentary presents the information necessary to successfully select and purchase business software designed for microcomputer systems.

INTRODUCTION

By 1980, over 125,000 microcomputer systems will have been sold to small businesses.The success or failure of each of these installations will be greatly dependent on how well the selected software addresses the business problems of the buyer. Hence the software becomes a major portion of the decision process when selecting a computer system.

But how does one "shop around" for software? And what guidelines can a prospective buyer follow to ensure that his specific requirements will actually be satisfied?

This article presents the answers to these questions by examining what types of software are available, the sources for software, and an acquisition procedure that should be followed to insure a "good match" between the software and the business.

TYPES OF SOFTWARE

The two basic types of software available are "canned" and "custom."

Canned software, also called "packages," consists of generalized programs that are pre-written and debugged and are designed to perform one or more general business functions such as accounts receivable, accounts payable, general ledger, payroll, or inventory control.

The advantages of software packages are their lower cost and their pre-written, debugged, and operational status.

The disadvantage is that as a generalized approach to a general problem they may not directly meet a user's specific needs.

Custom software consists of programs designed specifically to meet one user's particular requirements.

This provides the advantages that any specific need can be directly addressed (within the capability of the hardware), and that the software will conform to the business instead of the business conforming to the software.

One disadvantage is that custom software is more likely to have residual bugs after installation. Another disadvantage is the cost; since development costs can only be distributed to one user, the price of a custom software system is somewhat higher than a comparable packaged system.

Of the two types of software, packages are the most widely used means of obtaining cost effective computerized solutions to general business problems. If a package meets the user's basic requirements but is weak in some non-critical area, it may be justifiable to modify the package to correct its weaknesses. However, if no package addresses a particular application or need, custom software may be the only alternative.

SOURCES FOR SOFTWARE

There are four primary sources of micro-software: in-house development, contract programming, hardware manufacturers, and software houses. (1, 2)

In-house development is a source of custom software and consists of hiring a full time programmer/analyst. He then analyzes the business needs, designs the system, writes and debugs the programs, and installs the system.

This provides the advantage that the businessman retains control of the entire process which helps to insure the complete system will more closely fit his needs.

But a competent programmer/analyst is sometimes difficult to locate, expensive when found, and represents an ongoing expenditure. For these reasons few small businesses use this approach.

Contract programming is a common source for custom software and consists of contracting with a programmer/analyst or a consulting firm to custom design and implement a system specifically for that business.

This provides most of the advantages of a staff programmer/analyst without the ongoing overhead. Forfeited is the security of having a knowledgeable person on the staff ready to correct any problems that may arise.

Hardware manufacturers are becoming more aware that it is really the software that sells hardware, and are accordingly beginning to offer canned business software as an integral part of their systems, frequently at no additional cost.

This provides the advantage that the hardware and software are completely compatible.

But most manufacturers cannot afford to offer extensive support for their packages. If any modifications are desired the buyer would probably have to contract with an independent consulting firm.

Software houses are becoming an important source for software packages. They now offer a wide range of applications available for most of the popular hardware. Some offer contracting services as well.

The primary advantage of this source is the cost. Since the vendor can distribute his development costs over many customers, the costs to each becomes very reasonable.

But, as with any packaged system, careful analysis must be performed to determine if a package provides a practical and cost effective solution to the problem.

Of the four sources, software houses and hardware manufacturers provide the greatest opportunity for cost savings. Contract programming provides an excellent source of custom software if no packages address the requirements.

ACQUISITION PROCEDURE

Selecting software is a complex task requiring detailed analysis of various interrelated criteria. It should be approached in a systematic and logical manner to insure the selected product will provide long-term economy and satisfaction.

Following is a seven step acquisition procedure that when followed will aid the prospective buyer in making an informed and confident decision.(1)

Determine exact business needs. Before beginning the actual selection process an analysis of the business needs must be made to determine what problem areas are to be addressed and the feasibility of automating each of these areas.

Any area of the business that is chronically over budget, late in meeting schedules, or operating without effective management control is a candidate for computerization. If it is an iterative process that can be defined by relatively simple algorithms, and if the necessary data is available, then it should be possible to provide a computerized solution to the problem. (3)

The "classic" computer applications are general ledger, accounts receivable, accounts payable, payroll, and inventory control. Other common applications include word processing, budgeting, cost analysis, scheduling, and materials requirements planning. The list of possible applications is limitless.

Once the problem areas have been identified, the basic functions to be performed by the package must be clearly defined. It is best to do this in terms of outputs to be produced (reports and displays), inputs to be provided (available data), the volume of data to be processed, and the calculations to be performed. This information will be used later in the selection process.

Next, the selected problem areas should be prioritized according to immediacy of need and cost effectiveness. Each area should be approached individually in order of decreasing priority. This will provide the most rapid return on investment and avoid the chaos of attempting to install more than one application at a time.

Obtain documentation. Now that the goals the software is to meet have been established, the next objective is to acquire information about any packages or custom software services available that may meet these needs.

An excellent source for establishing initial contact with the availability of software is in appropriate trade publications and microcomputer magazines. Both will contain advertisements describing the software's purpose and provide the address where further information may be obtained. In addition the microcomputer magazines regularly publish informative product descriptions and reviews.

Computer trade shows present another means of introduction to available software. It is possible to see demonstrations of systems in operation and attend informative lectures oriented to the small businessman. This can provide a valuable education for the uninitiated.

Independent consultants can be utilized to gather information on applicable packages and also aid in the decision process, if desired.

If after initial contact a product appears to address the needs, send for further documentation. Request that the vendor provide all available documentation including a user's guide, system description, samples of reports and displays, and pricing information. This will provide the information necessary to perform the decision process.

Make intial selection. The first step in the selection process is to perform an initial culling from the list of available products.

The selection of three to five products should be based on each package's ability to meet the immediate and future needs, its ability to be installed in phases, and its approximate cost effectiveness.

This is usually a simple process as few products will be available that specifically address the buyer's requirements and fall within the desired price range.

Contact vendors. (2) Detailed analysis begins with contacting the vendors of each of the packages remaining and arranging for a personal meeting, if possible, and a demonstration. This will present the opportunity to ask any specific questions that may have arisen and to see the system in operation.

Some vendors do not have the staff to provide personal meetings. In this case a telephone conversation will have to be sufficient, but this probably also indicates he will be unable to provide on-site support if his package is the final choice.

In meeting with the vendor be sure he is fully aware of the software needs. Present to him the list of requirements developed in the first step and discuss it with him in detail.

Determine what modifications would be necessary, if the vendor would perform them, and at what cost and time requirement.

Review the pricing structure with the vendor and be sure to include all indirect expenses such as modification, installation, conversion, staff training, maintenance, and documentation.

Also ask the vendor how many systems have been installed (not just sold), if any are installed locally, and if a customer reference list is available. Any further documentation desired should be requested at this time.

Usually a vendor is happy to provide any information desired. If he is not, this could indicate a weakness in his product or support.

All interested parties should be involved in the vendor presentation including the accountant, upper management, and the intended operator. This will provide a good cross section of viewpoints and concerns.

View package demonstration. There are two types of demonstrations: the vendor may demonstrate the software on his own hardware, or he may make the demonstration at a user site.

Both methods provide the opportunity to determine the ease of operation, the readability and content of reports and displays, and the approximate time required to perform each function.

A demonstration on the vendor's hardware will probably provide the greatest flexibility as data can be modified without concern to its later validity. This will allow file updates, month-end and year-end reports, and other data dependent functions to be demonstrated.

A demonstration at a user site can provide some insight into the real-life operation of a system that may not be apparent otherwise.

As in the vendor's presentation, all interested parties should be included so that all viewpoints are covered.

Interview current users. Contact several users from each of the customer reference lists provided by the vendors. Ask each user if he is satisfied with the package and if he has any specific likes or dislikes.

Also ask each user specific questions about package limitations or errors that have surfaced, if any modifications have been made and if so what and why, if the vendor has provided adequate assistance and support, how difficult and time consuming the installation and conversion effort was, and what he would do differently if he had it to do over.

This information should provide valuable insight into each package's strengths and weaknesses.

Make final decision. Using all of the information acquired in the above steps, a decision must be made as to which product will perform the necessary functions and satisfy the evaluation criteria at the lowest overall cost.

The criteria that should be considered are listed below in the form of questions that should be answered in the course of the evaluation. (1,4)

1) Does the basic function of the package meet the determined needs? If it does not address the basic needs the required modifications would negate any cost or other benefit the package may have originally presented.

2) Will the package run on the intended hardware configuration? Is a version available for the targeted make and model of computer? Is it compatible with the associated operating system? How much memory is required? How much diskette storage is required for the given data volume? Does the package require specific peripherals, such as a printer, display, or diskette drives?

If a package requires additional of different hardware than that planned it may still represent a viable alternative if the benefits accrued are sufficient.

3) How well do the detailed capabilities of the system match the requirements? Are the input and output formats and content consistent with the needs? Are the reports and reporting periods adequate? Is the retained data sufficient to comply with reporting and legal requirements? What special forms are required and can the formats be modified to match user specifications? What control procedures, audit trails, and error checks are performed? What provisions are made for data integrity in case of mechanical failure or other disaster? What provisions are made for reruns?

No package will exactly meet all requirements, so it must be determined if it is a close enough fit to live with or if it can be economically adapted to the needs. In no case should a major business function be adapted to the software.

4) Is the performance adequate for the intended data volume? Can the package handle the projected volume in a reasonable period of time? How much manual intervention is required to perform each function?

A system that runs a major portion of each day would leave little time for reruns, period-ending reports, special functions, or future expansion.

5) How flexible is the package? Can its inputs, output, and processing capabilities accommodate changing business requirements? Can the system be expanded and/or modified easily to satisfy present and future needs? Is it capable of handling any expected data volume increases?

An intimate knowledge of the business is mandatory to determine if a package is capable of keeping pace with growing needs.

6) Is the package delivered in source language form? This will allow package modifications by other than the original vendor and is highly desirable.

7) How difficult will the package be to install? What changes are required to existing procedures and forms? How many people will be affected and what training will they need? How much data will have to be entered to provide a timely data base? How much computer and personnel time will be required to enter the initial data? Can a parallel operation be performed to insure the system is performing correctly? What manual operation is possible in the event of a disaster? Can the vendor furnish assistance in the installation and conversion process? At what cost?

These are important questions as the indirect expenses incurred during installation can quickly exceed the original cost of the software.

8) Will the system be easy to use? Is it designed for straight forward operation? Does it include easy to understand documentation for operating procedures? Are the input forms and the instructions for preparing them clear and efficient? Are the reports clear, comprehensive, and self-evident? Will everyone affected by the package find it satisfies his needs with a minmum of effort on his part?

If a package is difficult to use and understand it is probably doomed to failure regardless of the other benefits it presents.

9) Is adequate documentation provided? Is a comprehensive user's guide provided that includes basic operating instructions, specific operating instructions for each function? Are input preparation and report and utilization instructions provided? Is system documentation provided that includes system and program flow charts, program function descriptions, file layouts, and program source listings?

Documentation is vital for proper system function and ease of modification, and the quantity and quality of the documentation available with a package is a good indicator to its overall quality.

10) What support will the vendor provide? Will he adapt the package to meet the user's specification? Is operator training provided? Is installation assistance available? Will any errors discovered after installation be corrected? For how long? Will improved versions of the package be provided periodically? Is continuing maintenance available? What is the cost and availability of each of these services?

Most micro-software vendors cannot afford to include much support in the base price of the system and still remain price competitive. Thus the above services will range from non-existent to very expensive.

11) What is the operational status of the package? When, where, by whom, and for what purpose was it originally developed? When was the first installation completed for a user other than the developer? How many companies are currently using the package?

Another good indicator for the quality of a package is the number of satisfied users.

12) What is the total cost of acquiring, installing, and using the package? What is the direct cost of the package? What is the cost of any required hardware? What is the cost of installing the package, including modification, training, data conversion, and check out? What is the ongoing operating cost including personnel, paper, forms, diskettes, and maintenance?

The indirect costs can vary widely from package to package because of differences in the package design and the vendor's support policies.

13) What financial arrangements are offered? Is the package available for outright purchase, month-to-month lease, year-to-year lease, or perpetual lease? What are the specific terms and conditions for each plan? Are there any extra cost options? Are there any constraints on the use or modification of the package? What additional charges, if any, are imposed for support, training and maintenance?

A full understanding of each of the available plans and of the user's economic objectives is necessary for the wise selection of the pricing agreement.

When all the information available on each package has been gathered, analyzed, and compared according to the above criteria, a decision can be made as to which package will provide the best long-term economy and satisfaction.

CONTRACT NEGOTIATION

Having made the decision as to which package best satisfies the requirements, it becomes time to draw up a sound contract or modify the vendor's standard one. Avoid signing the standard contract as is because the protection and support needed later may be forfeited.

Specific terms and conditions that should be included in the contract, when applicable, follow: (1,2,4)

1) Product definition should include package requirements and performance specifications and what items will be delivered and in what form. Components should include source code, listings, and documentation, specifying the nature and the number of copies included for each.
2) Vendor support provided, including installation, training, and maintenance, should be specified in detail, together with the associated costs, if any. The means of support should also be explicitly defined (i.e., on site, telephone, mail, etc.).
3) Package modifications agreed upon should be concisely described as to which programs and documentation are affected and how.
4) Financial agreement should indicate the terms (outright purchase, month to month lease, etc.), pricing arrangement, and the associated rights.
5) Sign-off procedure shoud include contingencies upon sucessful delivery, installation, and operation of the product. Percentage of payment at each agreed-upon milestone should be based on the importance of the corresponding phase of acceptance.
6) Warranty should include the time period during which the vendor will correct any problems encountered during installation and use of the product. It should also specify the method of solution and response time.
7) Non-performance escape clause permits the buyer to withdraw from the agreement without penalty should the package fail to perform as specified.
8) Non-performance penalty clause specifies the manner in which the vendor could be penalized if the product has been misrepresented, or is delivered late or incomplete.
9) Package update arrangements, if applicable, should specify at what frequency, content, and format, they are to occur, and at what cost, if any.
10) Objectionable restrictions on the modification or use of the package should be removed from the contract.
11) Vendor proprietary rights should be respected including non-disclosure, resale, or other non-distributive clauses.

If the above guidelines are observed, the buyer will be in a better position to receive excellent support and reduce expenditures. The buyer's attorney should be involved in all contract negotiations with the vendor.

INSTALLATION

The next step is to install and check out the package. This stage requires careful and detailed planning and an organized and systematic approach. But that will be left for another to explain.

SUMMARY

Software selection for small business applications can be approached in a manner that will help insure the buyer receives maximum value for his investment. This procedure can be summarized as follows:

1) Analyze the business needs.
2) Gather information on available software.
3) Make initial selection.
4) Meet with vendors.
5) View demonstrations.
6) Interview current users.
7) Make the final decision according to the evaluation criteria.
8) Negotiate a sound and favorable contract.

The user who is willing to follow the above guidelines will most likely be one of the successful microcomputer installations which selected the correct software system as a cost effective automated approach to his business requirements.

REFERENCES

1) "How to Buy Software Packages," *Datapro 70*, Datapro Research Corporation, 1805 Underwood Blvd, Delran, NJ 08075
2) "Software -- Make or Buy?", ibid.
3) "All About Small Business Computers," ibid.
4) Wm. F. Kluckas and J.M. D'Allegro, "Evaluating Software Packages," *ICP INTERFACE Mini-Small Business Systems*, International Computer Programs, Inc., 9000 Keystone Crossing, Indianapolis, IN 46420

Reprinted from Volume 1, No. 1 of Business Computing Newsletter.
Free subscriptions available at P.O. Box 55056, Valencia, CA 91355.

THE END

What's a Cosmac?

BY KARL MOSGOFIAN
1362 16th Avenue
San Francisco, CA 94122

INTRODUCTION

Recently, computers based on the Cosmac 1802 microprocessor chip have become very popular. Many magazines are printing articles on the Cosmac computers and the 1802. But despite the rather sudden discovery of the 1802 by the microcomputer world, it is not new, and there have been computers based on the 1802 available for several years. So why has it taken so long for people to notice the 1802? And why is it all of a sudden so popular? Well, the answer to both questions is that it is different. Things that are different are thought to be either better or worse than the normal. In the case of the 1802, there is a little bit of both. But since it is such a different processor, it deserves some looking into.

For an uncommon (until recently), unusual chip, the 1802 has quite a following. People who like it, really love it. And those who don't like it are usually bothered by the fact that it is so different to program, when one is used to an 8080/Z80 type processor. It is manufactured by two companies, RCA (who developed it) and Hughes. There are five newsletters and at least that many 1802 clubs. One of the newsletters averages 50 pages—quite a lot to be put out by people who aren't paid for it. The 1802 is used in three highly selling single board computers and a bus oriented system in this country. It is used in the TELMAC-1800 series of computers, the best selling system in Scandinavia (600 delivered in four months). It is also being used in the development of the OSCOM-2000, for which Fortran and full Basic will be written. There really is an abundance of hardware and software available, if you know where to look. Multi-voice music boards, color graphics, cheap memory and full interface boards are all being sold, and for very reasonable prices. In the software department, there are lots of programs in newsletters, as well as Tiny Basic and several nice programs that individual companies sell. For instance, Netronics, makers of the ELF II computer, sell an assembler, disassembler and text editor for their system. And Quest Electronics, makers of the Super ELF, sell booklets of software and publish a newsletter.

So wha't's so great about the 1802? And if it's so great, why was it ever overlooked? Well, besides its oddness, which scares many away from it, the 1802 has not yet been used in real "personal" computer, along the lines of the TRS-80 or Apple. So you don't hear as much about it. But the single board computers that it is used in are reaching a new market, that of hobbyists who want to get started for very little money.

PROGRAMMING THE 1802

The closest thing the 1802 compares to in programming is the 6502 (at least in 8 bit processors—for all I know, there may be a bigger cpu which is very much like the 1802). If you are used to 8080/Z80 programming, it will take a bit getting used to. For instance, on the Z80, when you input a byte it comes from an input port to the accumulator. When you output a byte, it goes from the accumulator to an output port. The Z80 is, in a sense, register oriented. The 1802 on the other hand, is memory oriented. When you input a byte, it goes from an input port to the memory location specified by one of the 16 bit registers. So when you want to input a byte, you must set register X to equal the register which contains the memory address that you want the inputed byte to go to. Sound confusing? You get used to it after a while, and it makes the transfer of data very easy. By the same token, when you want to output a byte, the process works in reverse. Register X contains the register which contains the memory location whose contents you want to output. Perhaps an example will make it clear.

```
SEX    5    ;Register X=5
INP    1    ;input from port # 1
For this example we assume register # 5 contains 1FFF
```

Now the input byte goes to memory location 1FFF. So you see that this operates much differently than the Z80. Another oddity of 1802 programming is the multiple and movable program counters. All in all, it adds up to a processor that takes some getting used to. But there are programmers that swear by the 1802. There are others that swear at it.

HARDWARE FEATURES

Many of the best features of the 1802 are in its architecture and support chips. The 1802 has some built-in lines which are very helpful to experimentalists and make it possible to sell or build very cheap interfaces. First are the three external lines, which can be accessed in a single byte instruction. So it is very easy to accept inputs from various devices. And there is the "Q" line, which is a bidirectional software controlled line. In a single byte instruction you can turn it on or read it, so it is, in effect, like a built-in serial interface. That's why you can build such a cheap but reliable and good cassette or serial interface. And hooking up a speaker is nothing. So the external and Q lines make the 1802 a good chip to experiment with, in controlling the outside world, for instance.

Continued on pg. 23

EXOS®—A Software Development Tool Kit for the 6500 Microprocessor Family

BY DON COLBURN
EG&G Washington Analytical Services Center, Inc.
2150 Fields Road
Rockville, Maryland 20850

The fastest and easiest way to complete any task is with the right tools. This statement is as true with software as it is with hardware. The person that assembles a CPU card with a 60 watt soldering iron, 1/8" solder, and a pair of lineman's pliers is in the same boat as the one who hand assembles an 8K assembly language program. Although it is possible to assemble a CPU card with limited tools or to hand assemble large assembly language programs, few people who are seriously developing software can afford the time and resources required to do the job without the right tools.

This article describes a software development tool kit which provides the "right tools" required to efficiently and effectively both generate and modify 6500 assembly language programs. The level of capability is equal to that of some minicomputer operating systems.

When I bought my JOLT® 6502 based system a couple of years back, I quickly realized that I would need something more powerful than the TIM® monitor if I seriously intended to do more than flash LED's. I had two alternatives. I could write the required software, assemble it, load it, and debug it by hand, or I could purchase someone else's existing software. Faced with the chicken and the egg problem, and feeling somewhat strongly about what such a package should contain, I decided to first buy an existing assembler and editor package: with these basic tools I would develop my own expanded package. The assembler had to be easy to use and compatible with the standard MOS technology mnemonics and the editor had to have the ability to create and save source files which could be directly input to the assembler. The task of finding these basic tools proved more difficult than I first anticipated.

While at a computer show one weekend, I came across a booth offering 6500 software for sale. Their sign advertised a 4K operating system, a 2K single pass Assembler, a 1K Editor, and a Trace/Disassembler package. Skeptically, I approached the counter. Two hours later, I left the counter carrying eight EPROMS and four comprehensive manuals. Not only had I found the assembler and editor I had been looking for, but I had found the software development tools I had intended to write. The name of the package is EXOS®.

EXOS stands for "Extended Operating System." This is because it contains a base-level operating system with 17 standard and 5 extended commands. The extended commands pass control to an optionally resident Assembler, Editor, Disassembler and Trace program, or Disk Operating System. EXOS is compatible with all 6500 systems based on the TIM monitor. Other monitors such as Apple and homebrew versions may be interfaced, because both the program cold and warm entry points and system I/O vectors are located in the first 27 bytes of each program (Table 1) and initialized into RAM upon cold start. This allows the user to change the I/O calls according to System Requirements.

EXOS® is a Trademark of RCS Associates, NY.
JOLT® is a Trademark of Microcomputer Associates, CA
TIM® is a Trademark of MOS Technology, PA.

Table 1

EXTENDED OPERATING SYSTEM "EXOS" FAMILY VECTORS

1. The following vectors are used as convention in all "EXOS" family software packages and programs.

   a. This feature should allow "EXOS" software portability on any MOS Technology 65XX based system.

   b. Vectors are common if used in that particular package. Where a vector is not used, either $EA/$FF will be found.

   c. All "EXOS" programs will operate under the "TIM" or "DEMON" operating systems without modification. Operation under the "KIM" OS is attained thru change of the address vectors to be described.

   d. Vectors

HEX PROGRAM ADDRESS	LABEL	OPCODE	LABEL	COMMENTS
XX00*	ENTER	JMP	ENTRY	COLD START initialized regs.
XX03	RENTER	JMP	RENTRY	WARM START re-enters program counter & regs.
XX06*		SPECIAL (See note)		Varies w/EXOS program
XX09	PR1CHR	JMP	PRCHR	Print 1 character to input (i.e. printer, CRT) device from accumulator
XX0C	RD1CHR	JMP	RDCHR	Read 1 character from input (i.e., keyboard, tape, disc) device to accumulator
XX0F	PRCRLF	JMP	PREOL	Print CR & LF w/delay in $E3
XX12	PR2SPC	JSR	PR1SPC	Uses PR1SPC
XX15	PR1SPC	JMP	PRSPC	Print 1 space
XX18	PR1BYT	JMP	PRBYTE	Print 1 hex byte
XX1B***		BREAK	JMP	BRKTST Test for dynamic escape thru use of "break" hey.
XX1E**		SPECIAL		
XX21	3 byte vectors that depend upon particular "EXOS" program			

Table 1 cont'd

NOTES:

* All programs use the "ENTER" vector, however the presence of other vectors depend upon the program itself.

** Varies w/program

*** Routine depends upon I/O used w/system some "EXOS" programs have the "BREAK" imbedded in object code, however use of an external break routine is still allowed. Imbedded routines are identified w/the particular program and locations are given to allow the user to alter those routines if desired.

Standard EXOS commands may be divided into four categories; operational, functional diagnostic, and extended. The operational commands provide the user with the capability to ENTER, LOAD, and DUMP data to and from memory and to DISPLAY, FIND, MOVE, calculate branches and addresses (MATH) and EXECUTE that data. The Functional commands allow the user to modify the carriage return delay in all systems, and the baud rate (up to 4800 baud) in TIM systems, under keyboard/software control. The diagnostic commands allow the user to nondestructively TEST or COMPARE any specified blocks of memory on a bit-by-bit basis. Extended commands allow direct access to the optionally resident Assembler, Editor, and Debug firmware. Table 2 contains a brief description of each of the 17 standard EXOS commands.

TABLE 2

The Table contains the name and a brief description of each of the 17 Standard EXOS commands. The short form of the command is capitalized and underlined.

NAME	DESCRIPTION
COMPare	Compares any size block of data to another. All differences are listed along with the Hex Address.
DISPlay	Displays memory contents in hex and ASCII. The display is formatted into 16 lines of 16 bytes to assist in address identification. A header is output every 16 lines (256 byte block) so that a header is always visible. ASCII equivalents are presented along side the display.
DUMP	Dumps the specified range of memory to the I/O port in MOST* format.
ENTEr	Allows entry of hex data into memory in the display format. Pointer is auto-incremented and formatted. Editing is permitted, and ASCII equivalents are displayed similar to the display command.
EXECute	Allows user to jump to any specified address and begin execution.
FILL	Fills a specified range of memory with a specified byte of data. If memory does not accept data, an error diagnostic is printed.
FINd	Lists occurrences of specified hex data or an ASCII string within any specified range.
LOAD	Loads a MOST* format tape from either the serial I/O line or the high-speed reader port. An offset may be applied to the tape specified addresses, and a file number may be searched for.
MATH	Converts decimal to hex, hex to decimal, calculates relative branches, and absolute differences. Answers are displayed in hex and signed decimal.
MOVE	Copies specified range of memory to target address.
NULL	Inserts the requested number of nulls after each carriage return/line feed sequence.
SPEED	When used in systems using the TIM monitor, this command allows keyboard selection of different baud rates under program control. Available baud rates are 110, 300, 1200, and 2400 baud. (4800 baud with 2 MHz systems).
TAPE/IRG	Modifies the print 1 character routine to output an inter-record gap (IRG) between text lines dumped by the editor. This allows any tape cassette to input source to the assembler at speeds up to 2400 baud without any handshake. The IRG command allows the user to specify an inter-record gap timing value different from the "TAPE" default of 2.
TEST	Non-destructively tests each RAM location over a specified range. If an error is found, a diagnostic message specifying the exact word and bit in question is printed on the console.
USER/UTILity	These commands execute a pre-defined user program pointed at by a Z-page vecotr. The user may alter these vectors as desired.

* MOST format is the tape dump format developed by MOS technology for 6500 based systems.

The EXOS Assembler (EXASMB) is a very unique piece of software. First and foremost, it is a single pass assembler. This means that forward references are resolved when the referenced label is defined. If never defined, it is so flagged and cross-referenced in the symbol table.

Second, the assembler provides full tape I/O handshake control with both ASCII control characters and physical relay drivers.

Third, when the Assembler is used with EXOS and the EXOS editor (it is available stand alone), it provides audio or digital cassette input capability up to 4800 Baud, without the requirement for incremental start-stop control. This allows complete independence from the mass storage device. No other 6500 assembler that I am aware of provides this capability. All other require an ASR teletype, an incremented digital cassette or use of inefficient memory to memory techniques.

Fourth, a set of MOS Technology compatible pseudo operations is avialable, and when used with the optional DOS, one may designate multiple disk source files inside a single source file.

Fifth, high and low order address bytes may be designated as immediate operands.

Sixth, the symbol table uses an extremely efficient label packing algorithm which allows two distinct 8 character labels which share the same first six characters to require a total of only 11 bytes in the symbol table. The pseudo-op .END on the EXOS command XREF allow the symbol table to be dumped repeatedly. All labels appear in alpha sorted order when listed.

Finally, and most remarkably, the entire object image of the Assembler requires only 2K. This occupies 1/3 the space of some 6500 assemblers, while providing more capability in a single 2716 or two 2708's.

The EXOS Editor (EXEDIT) was written with the same 'attitude' as the assembler. It provides full line editing capabilities, as well as the ability, when used with EXOS, to output assembler compatible source code to any cassette interface. The line editing capabilities allow one to print any number of lines, to insert a line, or delete a line. A unique feature allows one to automatically generate line numbers when source code is being input, and turn them off when the source is being dumped. Another is a selectable automatic tab feature which advances a variable number of spaces (default is 8) following the label to align all operation fields. Again it is quite remarkable that the total EXEDIT software resides in only 1K of memory (1-2708 EPROM).

The EXOS Disassemble/Trace package (EXTRA) is one of the most powerful software debugging tools available. The disassembler disassembles code from the start to the end of a user defined range. If the user interrupts the program with the BRK key, he is given the option of continuing with the disassembly or exiting back to EXOS. The Trace program requests the user to load the register with the conditions he wants to trace from (includes P.C.). It then disassembles the code at that location, displays the hex code and mnemonic, and awaits user input. If the user strikes a '/', the op code is executed, the resultant registers are displayed and the next line of code is disassembled. If the user strikes a '+' and a hex address, the processor executes the code in real time until the P.C. equals the given address. At this point, the line is disassembled, and the program waits for user input. If the user strikes an 'X', tracing from the current location is terminated and the user is reprompted for new PC and register contents. Upon receipt of a ':' the software allows the user to modify the current register contents prior to execution of the current line of disassembled code, allowing modification of program flow as desired. Any time the "ESC" key is struck, control is passed back to EXOS. This tool allows tracing through RAM *or* ROM resident programs in both binary *and* decimal modes (the 6502 has a unique and easy-to-use decimal operating mode). The object image for both the Trace and Disassembler combined is under 1K bytes (1-2708 EPROM).

A recent addition to the EXOS family is a firmware interface to the PerSci 277 Dual full size floppy disc and intelligent controller. This capability provides EXOS with an extremely powerful Disc Operating System (EXDOS). The interface resides in 2K bytes and adapts the entire EXOS system without modification for Disc I/O, including the Assembler and the Editor. This is typical of all EXOS software in that each higher level of software that is added requires no changes to previously purchased software. This true upward compatibility is provided for through manipulation of the Z-page systemI/O vectors so that new capabilities are a super set of the previous level.

I have been using EXOS for more than a year now, and I have yet to come across a software bug. This issue contains two source and object code listings of software developed using EXOS. They include a standalone version of the EXOS Display command and a standalone interface to the Per Sci 1070 controller based on the Interface Age article of August 1977.

EXOS products are available from:

CGRS Microtech
P.O. Box 368
Southampton, PA 18966

RCS Associates
P.O. Box 160
Miller Place, N.Y. 11764

EXOS software is delivered on prime 450ns 2708 or 2716 EPROMS and each piece of software is accompanied by a comprehensive user manual. Both of the above sources offer the manuals for $5.00 each or $15.00 for the entire set (cost is applied towards purchase price). EXOS is now available on AIM/KIM/VIM compatible cassettes for $30 each from RCS.

HOW TO MAKE YOUR COMPUTER TALK

THE SONG OF THE CPU

BY CHARLES M. PHELAN
1817 N. Edgewood Terrace
Fort Worth, Texas 76103

© 1979, Charles M. Phelan

A simple and very inexpensive method for producing voice output from a microcomputer was described by Phil Mork in "*Vocal Memory Dump*," *Dr. Dobb's Journal*, Vol. 3, Issue 8. Voice signals are processed by first differentiating the input waveform and then sampling the sign of the derivative at a rate determined by software. This information (one bit per sample) is then stored in memory. These bits can then be output at the same rate through an integrator, producing a fair reproduction of the original waveform. The fidelity improves to some degree with higher sampling rates.

I have performed some experiments with this technique, and have discovered a pitfall or two in its application.

The input processing is not sensitive to amplitude and is very sensitive to relatively low level noise contained in the voice signal. My first efforts, using voice recorded and played back on a Radio Shack CTR-21 which I use for digital work, produced vaguely intelligible voice along with sounds similar to a buzz saw cutting a bucket of ball bearings. Some testing showed this latter sound to originate from noise such as tape hiss, amplifier noise, and background noise during recording, most of which could not be eliminated using the CTR-21.

Fortunately, I was able to get a cassette recorded by a professional announcer using high quality equipment. This removed all but the playback noise from the CTR-21 and gave understandable output. As the noise level was still annoying, I introduced a small amount of hysteresis in the processing circuit. This made a dramatic improvement in the noise with a small loss of intelligence in the voice.

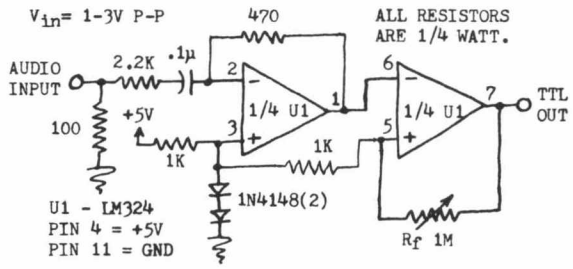
Since nearly all voice information is contained in a band of frequencies between 250-2500 Hz, a new recording with strong rolloff of both high and low frequencies gave further improvement with some loss of naturalness to the voice. A good communications type voice filter should give even better results.

I have not attempted experiments with the output circuit yet, but one approach is dictated by information theory. Since the higher harmonics can beat with the sampling frequency to produce sum-difference overtones, frequencies in the waveform to be digitized should be limited to less than half the sampling rate. Thus the somwhat "buzzy" quality of the speech output might be reduced. Further, with an audio cutoff around 2.5 Khz, good quality output might be obtained with a 6 Khz sampling rate (I've been using 8Khz with the R-C filter). Incidentally, caution should be exercised when putting the output through a high quality audio system since harmonics are present which could fry a tweeter in short order.

Reprinted from *The Printed Circuit*, March, 1979.

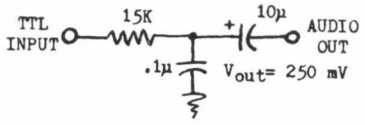
The input circuit shown is more complicated than that in *Dr. Dobb's*, but is still rather inexpensive and easy to build. The op amp is a LM324 but many others will work. Better performance could be obtained with a higher open-loop response and slow rate amp. The 3900 or other Norton amplifiers will not work in this circuit without biasing modifications.

The first stage differentiates the input and introduces some low frequency rolloff with the 2.2K input resistor. The second stage functions as a comparator and introduces a small amount of hysteresis which can be adjusted by Rf. Input voltage should be 1-3 volts p-p. The adjustment of Rf is sensitive to the input level and must be varied to a compromise between noise and distortion. Those with high quality audio equipment can set Rf to its maximum value or remove it altogether (open circuit).



Voice Input Circuit

The output circuit is essentially unmodified. A blocking capacitor has been added to remove DC bias from the signal and the input resistance increased to prevent overloading my cheap amplifier (250 mV rms max input). Some component value juggling might prove beneficial here.



Voice Output Circuit

Current required by the input circuit is less than 10 mA so power may be obtained in most cases from the I/O board (in my case a P.T. Co. 3P+S). If standard TTL is used for the input bit, damage to the chip could occur if a separate power supply is used with nonsimultaneous turn-on and turn-off.

Memory usage is the main drawback with this technique. Some applications requiring limited vocabulary can keep the digitized speech resident in RAM, but some form of disk storage would be necessary for a larger number of words. Numerous applications come to mind such as memory dumps, interactive prompts, terminals for the blind, and of course games (since the voices are recognizable, words could be "borrowed" from TV broadcasts for a really farout Star Trek). Further information was published in *Data-Boy™ Speech Processor*, Mike Gabrielson, *Dr. Dobb's Journal*, Vol.3, Issue 9.

A Look at Computer Music

BY CURTIS ROADS, EDITOR
Computer Music Journal
P.O. Box E
Menlo Park, Califronia 94025

Curtis Roads is the Editor of Computer Music Journal. *He is also a composer. Mr. Roads has been active in the fields of music, computers and computer music for many years.*

Preface

A slightly longer version of this paper was written at the prompting of the Pascal Special Interest Group of DECUS, for a "Pascal Applications Seminar" at the DECUS Symposium in San Francisco. The presentation was accompanied by a demonstration tape which was compromised of both historical examples of computer music as well as contemporary examples. The talk was geared to a technically sophisticated audience who presumably knew little about computer music. The subjects of Pascal and computer music were intertwined throughout the original paper. This version focuses mainly on just computer music itself, and in particular sound synthesis and composition with computers. Due to the brief and introductory nature of the paper, some important developments and names have been omitted, for which I apologize. Although the early history of electronic music is extremely well-documented, a good comprehensive history of computer music has yet to be written.

A BRIEF CONVERSATION WITH CURTIS ROADS

DDJ: Curtis, tell me about *Computer Music Journal.*

Roads: *Computer Music Journal* is the only journal in its field. We're in our third year of publication and we're presently distributed to universities, research institutes and individuals in 27 countries. We feature both beginning and advanced articles in the field.

DDJ: What function do you see *Computer Music Journal* fulfilling?

Roads: I see *Computer Music Journal* as a service. We're here to relay information about new developments, to teach, and as a forum for all of those interested in applications of computers to music.

DDJ: Just who are those people?

Roads: First of all, composers, music teachers and music theorists. Secondly, we have a very broad technical audience consisting of designers, programmers, and researchers. Third, other interested individuals ranging from hobbyists to musicians who have not yet entered the computer field.

DDJ: That sounds much like the *Dr. Dobb's* audience. Which leads me to ask you if there is anything going on in the way of microprocessors and computer music?

Roads: Very much so. Some of the most intense activity is coming from that domain. People are using high speed microprocessors for sound synthesis. Still others are using them to control traditional electronic music synthesizers or instruments like organs and pianos. Another development is that of the microcomputer band in which each member of the band is represented by a microprocessor in a network.

DDJ: A little bit more on that one.

Roads: Well, there's now a League of Automatic Music Composers in the Berkeley area who perform every week.

DDJ: You mean they're actually playing microcomputers like instruments?

Roads: Yes, yes! (Roads points to a poster on the wall which announces a concert by a microcomputer network band.)

DDJ: Since you *are* the only computer music journal, Curtis, you must publish articles which are not only state-of-the-art but controversial.

Roads: Yes indeed, if only because one person's music is another person's noise. For example, for some people it is presently important to simulate natural instruments, while another group feels that research into new sounds must take precedence. Then there are the dedicated professionals versus the dedicated amateurs. And, finally, there is a group who feels that any encroach-

The Early History of Computer Music

Although the history of computer music is intimately tied to the history of other forms of electronic music, I'm going to treat it separately, for purposes of brevity.

The sound generating programs developed by Max Mathews and his associates at Bell Laboratories in the 50's are the earliest known computer music attempts. While various experimental sound synthesis systems were set up at the University of Illinois Argonne National Laboratories and MIT in the early 60's, it was not until the publication of Max Mathew's book *The Technology of Computer Music* by MIT Press in 1969 that wider interest and activity in computer music began to take off. Concurrent to this software advance, the introduction of the minicomputer at about the same time was another influential factor.

Parellel to these developments in sound synthesis, a number of experiments using computers to generate scores were being carried out by Lejaren Hiller at the University of Illinois. Central to these experiments was the understanding that music is an algorithmic process. Hiller used Markov-chain techniques to generate harmonic and contrapuntal progressions, as well as atonal aleatoric music, and serial music.

Another early experimenter with computer music composition was Herbert Brun, also at the University of Illinois, who developed personal alogrithms for composing, Simultaneously, experimenters in France, the Soviet Union, and the Netherlands were developing score-generating programs and languages for conputer-aided composition.

Regardless of these early experiments with sounds and scores, it has only been in the past decade that a rapid advance in computer music has been possible, due primarily to the development of sophisticated computer music software. Clearly, computer music is still in its infancy. The problems in this domain are very complicated and are intrinsically multidisciplinary. While some problems are "hard" problems in the sense that their solution is simply a matter of engineering, still another set of important problems are "soft" in that issues of musical aesthetics and philosophy are involved.

An Introduction to Computer Music

First of all, how does a computer make music? There are two principal ways of using a computer to make music: first, one can use the computer as a very general instrument for producing sound; second, one can use the computer as a composing assistant for generating scores.

ment upon the present composer's role will lead to detrimental results, while another group feels that the composer's role may be transformed through computer-aided composition.

DDJ: What advantages does a computer have for a composer?

Roads: Presently, there's a great deal of development being done in generating new sounds and in interesting transformations of known sounds. Beyond this a composer may use a computer as a composing aid, since the information processing capabilities of the computer are as useful to a composer as they are to any other user. Going still further, some composers are interested in modeling compositional processes and in researching various compositional problems.

DDJ: So music can be produced autonomously by a computer?

Roads: Yes.

DDJ: Curtis, in your article you refer to the two main applications for computer music: sound synthesis and score generation. What else is going on along these lines?

Roads: There are all kinds of applications. Many people are working on music printing, computer-aided analysis of music, computer-based acoustical research and, the one which will probably have the widest impact, digital recording.

DDJ: Can you describe why it is so superior to ordinary recording?

Roads: In the computer, the sound signal is treated as just another form of information. Using digital techniques, we can achieve higher fidelity and more precise treatment of the sound signal. For instance, with every bit of precision that we use to represent the sound, we gain 6 decibels of fidelity. In theory, therefore, fidelity is just a matter of the word length which one uses to represent one sample of music. As to precision, computer editing techniques are limited only by the sampling rate which means that one can describe a sound down to around 1/50,000 of a second.

DDJ: Digital records are about $15 right now. Do you see them getting cheaper, or, rather, will the recording process be getting less expensive in the future?

Roads: I cannot predict what the marketing strategies of recording companies will be in the future; I only know that the cost of digital recording as a process will become much cheaper. The most important factor in the cost of digital recording at present is the cost of mass storage. Presently, digital recordings are made on large disks or high-priced tape machines. In the future, inexpensive chips will probably bring down the cost of mass storage greatly.

DDJ: Your article covers the history of computer music, but what do you see happening in the future?

Roads: I hope that computers will better enable us to understand music and that they will help us to organize it in as yet unexplored ways. Hopefully, the increasing intelligence of the computer will transform our role in interacting with music and with each other when we are making music.

First, let me describe how a computer generates sound. To start with, speaker systems are driven by voltages – +10 volts makes a speaker cone move forward, while – 10 volts makes a speaker cone move backward. By varying the voltages sent to a speaker one can produce vibrations in the air that we call sound. Now computers can very quickly generate bit patterns which we interpret as numbers. Then by sending these numbers (which may vary as a sine wave varies, for instance) out to a device called a digital-to-analog converter at audio rates, we can generate variations in voltage that drive speakers back and forth to produce sound. Sampling theory tells us that in order to generate sounds of n Hertz, we must send out digital numbers (or samples) to the digital -to-analog converter at a rate of $2n$ Hertz. Since we can hear sounds up to around 16 KHz in frequency, it is a good idea to generate at least 32,000 samples (or numbers) for each second of sound.

Thus computer sound synthesis of this type is a sizable number-crunching problem. Recently, however, a number of strictly digital sound synthesisers have appeared. These devices are typically microprogrammable very high speed machines of technology like Schottkly TTL or ECL with lots of parallel pipelining. Their only purpose is to generate numbers for sound synthesis, and they are generally controlled by a host computer which can be a microcomputer, a mini such as PDP-11, or a large-scale machine. Presently, the most advanced device of this type is the Systems Concepts digital synthesizer (or "Samson Box") designed by Peter Samson here in San Francisco. The computer music group at Stanford uses a DEC-10 variant to control their Samson Box. The Boston Symphony is also buying one which will be controlled by DEC's compact 2020 mainframe computer.

Yet another approach to the digital synthesiser has been taken by Hal Alles at Bell Laboratories in his LSI-11 controlled digital synthesizer. While most of us are accustomed to thinking in terms of two, four, or sixteen channels of sound with analog technology, both Samson's and Alles's devices can generate 256 "voices" of sound in real-time. Of course, these machines are on the extreme high-end of computer synthesis devices at the present time. Both of these devices are one or two of a kind, and while the magnitude of sound produced by them is on the order of the symphonic, their prices are beyond $100,000.

Apart from these supermachines, many people are now involved with projects in which high-quality sound may be generated using lower-cost 1 to 16 channel digital synthesizers. Still others are using microcomputers to control traditional analog systems like the Moog synthesizer. I have seen intelligent microprocessor-based music systems in performance which cost as little as $1000. A word of warning is in order though, as some of the microcomputer magazines have had articles and ads for so-called "computer music" software or hardware which amount to nothing more than tapping the interrupt line off of an S-100 bus in order to get square-wave-like sounds. This fixed-waveform type sound is of very low quality and what is worse, inflexible, making it impossible to get beyond an annoying buzz or beep in terms of timbre. Most people today are agreed that satisfactory digital music requires from 12 to 16 bits of precision, and that an ability to produce time-varying waveforms is essential.

Computers and Composition

As I mentioned earlier, one of the most interesting uses of computers in music is as a composing aid. This aid can go all the way from handling real-time input from physical devices such as keyboards or digitizers, all the way to modeling the composing process in software. At intermediate stages between these two poles, the computer may perform necessary calculations for the composer, or fill-in a shorthand compositional specification provided by the composer. This filling-in process can take place on various levels, from the generation of sounds to the generation of structural descriptions of music.

One of the great problems in computer music is that of the representation of music software. Music is an intrinsically multi-leveled language. On the one hand, it can be seen on a surface level as a collection of sounds of various kinds; on the other hand, there are always various substrata of architectural form which constitute what one might call the macrostructure of a particular piece of music. The macrostructure or background structure of a composition consists of the interrelations amongst the surface of foreground elements, that is, amongst the sounds.

For example, phrases, progressions, transformations, and repetitions of various kinds are all components of the macro or background structure of a composition. In a more general sense, most temporal processes are generated by some macrostructure, so this characterization goes far beyond music.

One way to think of this representation problem is that you have two poles; on the one hand you have a collection of notes, and on the other hand you have a whole score. How do you represent all the middle-level structures inbetween?

One particularly apt representation is a hierarchial tree organization. Using this tree structure, composers can be aided in organizing complicated musical structure chunk-by-chunk, on whatever level they wish to work. What is important about this approach is that it allows any chunk to be treated in exactly the same manner as a single note, regardless of the size of the chunk. Once again, this kind of representation is of general utility in fields outside music as well. To return to a musical example, a score is a series of musical events. A musical event may be a section of a composition, a phrase, or a single note. Any of these entities can be treated as a chunk. If we want to repeat a phrase for instance, we need keep only one *master copy* of that sub-tree and at each instance of it in the score we store only the subtree identifier and the transformations to be effected at that particular instance. This saves space and has the important implication that any changes in the master copy are reflected in every instance of it in the score. Thus. if we transposed a figure up an octave, every copy of the figure in sub-scores below it would be similarly affected by just one action. This is the approach that has been taken by a group at the University of Toronto. Their model has been implemented via an elegant system of linked-lists, and runs on a minicomputer.

So a linked-list construction is one way of representing a possibly complicated tree structure. Yet another way of representing tree structures is by means of a table-driven system. The syntactic structure of a formal language like Algol, for instance, can be represented in a list of production rules

called a production table. A BNF representation typically specifies the content of these tables, which may be constructed by a compiler-compiler. The phrase structure of music may be so defined. That is, a composition may be seen as a collection of movements, which may be rewritten into large-scale phrases, which maybe again rewritten into still smaller phrases, which may in turn be rewritten into music notes or other sound objects. As it turns out, simple context-free tables are actually not powerful enough to describe certain context-sensitive features of some music, and a two-level grammar with some special procedures must be used in conjunction with the parse tables, but I needn't go into details here. This table-driven kind of music specification system has been a project of mine over the past year. To very briefly explain this project – a composer specifies the syntactic structure of a *set* of compositions by means of a metalanguage. A compiler-compiler for these composing grammars constructs the necessary tables. The composer then specifies a set of compositional expressions in a composing language. By referencing the composing grammar, these compositional expressions (which describe the structure of a composition at a very high level) are then compiled down to a sequence of terminals. Another program maps these terminal tokens into a lexicon of sound objects.

Using such a system, composers can experiment with any number of unified compositional structures which all derive from a single composing grammar.

```
compared; if <string> is < the last record compared, return -1;
if > the last record, return +1.

< sub > :  How can we address two-dimensional arrays in tiny-c,
which permits arrays of one dimension only?  We dimension the
array to (maxkey * maxcolumn) - 1, then address ARRAY(X, Y) as
ARRAY(sub (X, Y)).

< traverse > :  Traverse and print the tree.  All arguments are
pointers.  Since C calls by value, a called function can change
values in the calling function only if they're globals or if
the values passed are pointers to those that must be changed.
<traverse> is recursive and must pass its arguments in both
directions.

< write > :  Write <tree> to a file, along with the parameters
<n> and <root> and the table of pointers in <avail>.  This is
a wasteful function, since it writes out all of <tree> when we
need only so as far as tree(sub(highestpointer, maxcolumn)).  But
there's no way to find the highest pointer without making the
demonstration muddier than it is.

< yourchoice > :  Get a character from the terminal; if it's
upper case, return it as l.c.
D.
```

Continued from pg. 41

Continued from pg. 44

exactly the same as both a manual experiment and the true probabilities. When I tried the fourflush hand for 10,000 cycles, I got the following results:

Bust	Probability = 23/47 = .4895	Experiment = .5014
Pair	Probability = 12/47 = .2553	Experiment = .2450
Straight	Probability = 3/47 = .0638	Experiment = .0618
Flush	Probability = 8/47 = .1702	Experiment = .1679
Straight Flush	Probability = 1/47 = .0213	Experiment = .0239

The estimates obtained this way are off by only about 1% at worst. If I ran more trials, the estimates would improve.

When Should You Simulate?

Basically, you should think of doing a *probabilistic* or *Monte Carlo* simulation whenever the direct analysis of a problem is about to get lost in a forest of cases and calculations. But for the simulation to be successful, you must meet certain conditions. First, the real world situation needs to have a model accurate in all its salient features. In the poker model, a bit vector 52 long is a good model of a deck of cards and random selection of elements of the vector is a good model of the drawing process from a shuffled deck. Notice that we do not need to have a model of the shuffling process itself and to construct one would be to bring extra work upon ourselves.

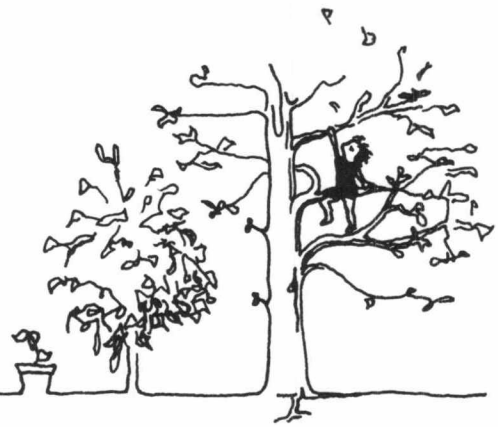
Second, it must be fairly easy to evaluate the results of an experiment within the program. For my poker program, a 50-line PASCAL procedure suffices to decide the value of a hand. If deciding the results of an experiment takes forever and a day, you are not going to be able to run many experiments. However, the validity of the method depends on large numbers, so you must do many experiments to be successful. Finally, you must have a random number generator you can trust not to bias the experimental outcomes. This is exactly the same problem as guaranteeing that your shuffling technique is fair. If you don't know how to do this, read up on the subject in Chapter 3, Volume II of Knuth's *Art of Programming*. By the way, I suspect a bias in my random number generator and I am using the fourflush problem as a backwards check on the generator's fairness.

Summary

Although I have illustrated Monte Carlo simulation with a game program, it is commonly used for much more serious business. Many scientific problems much too hard to analyze consume hundreds of hours a month on the nation's super computers, drawing cards and shooting dice to come to a solution. Uses closer to home exist as well. I have a high-school aged friend who wrote a small physics simulation in BASIC, ran it for several months on a micro, and has now published his results in the prestigious *Physics Review Letters*. While your use of simulation will probably not give you such instant acclaim, I can guarantee that it will give you a lot of enjoyment.

Growing Pruning & Climbing

BINARY TREES with tiny-c



BY LES HANCOCK
305 Lexington Avenue
New York, NY 10016

© 1979 Leslie Hancock

Alexander Graham Bell invented the tin ear, so maybe it's not surprising that Bell Labs' best software should have names like UNIX and C. The UNIX operating system, which really has nothing to do with keeping a harem, has become fairly well known, but the C language it's written in is just beginning to be heard from. The purpose of this article is to show you what C looks like and how much it can do, even in a tiny avatar.

C's popularity has probably suffered from the fact that there's nothing racy or showy about it. If programming languages were horses, C would be a hard-working quarter horse and not a show horse like PASCAL or a race horse like APL. C is docile, regular, predictable, sometimes all but invisible—certainly nothing you'd buy a ticket to watch. In a word, C is *pragmatic*—but remember that pragmatism can be a rigorous philosophy.

C was written in 1972 by Dennis M. Ritchie, and has the coherence that often comes from single authorship (think of APL, PASCAL, TRAC). Ritchie's goal was to provide high-level controls for low-level programming—in other words, to get as many jobs as possible done with the least possible fuss. He was successful. True, C doesn't quite have the generality of an assembler, and it doesn't quite have the formal elegance of PASCAL or APL; but it can do systems work without killing the programmer and applications work without killing the machine. In a sense it's the logical successor to FORTRAN. FORTRAN was created to free applications programmers from assembly language; C extends that freedom to systems people.

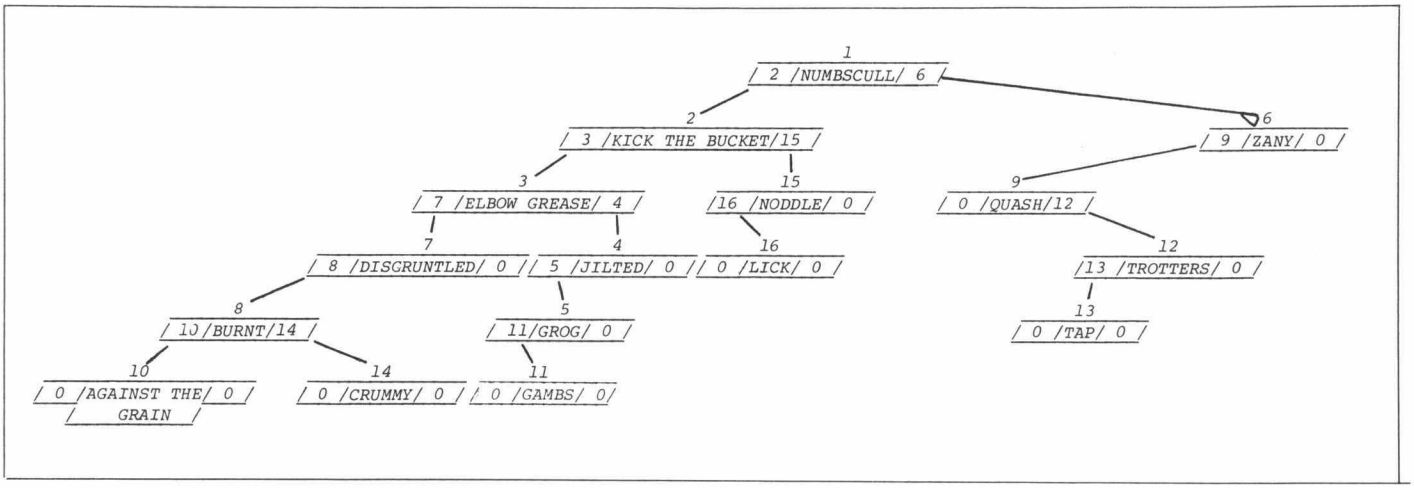
C was first installed on a PDP-11, then on a Honeywell 6070, then on a 370. Other versions are in the works, but at present no compiler is available for any microprocessor. What we do have is "tiny-c," created by Tom Gibson and Scott Guthery and sold (with lovely documentation) by tiny-c associates, P.O. Box 269, Holmdel, NJ 07733. Their version is an interpreter that fits into just over 4K bytes of 8080 code. It lacks the conveniences of big C, but the essentials are there. In fact, tiny-c sometimes expresses algorithms more clearly than its big brother, since it forbids the shorthand and outright sleight-of-hand that you can get away with in the full implementation. Its only real sin is that it's not a proper subset of the language. There are differences in syntax which make it impossible to put tiny-c code through a C compiler.

Be that as it may, I've begun using tiny-c to code up examples of programming algorithms from Knuth[1], with the idea of eventually putting them into standard C, then putting that into a book—a sort of commonplace book for programmers. Each example would be a runnable, useful program. Knuth provides his own examples, of course, but they're written in a hypothetical assembly language and seldom actually *do* anything. He defends this approach on the grounds that "algebraic languages such as ALGOL or FORTRAN" are useful for numerical work but not for coroutines, combinatorial searching, recursion and such.[2] Knuth's discussion of this point might be run as an advertisement for C, which is perfectly at home with those non-numerical topics. Here again, C spans the gap between systems and applications. It doesn't stretch an algorithm to fit its own shape (at least not as grossly as ALGOL or FORTRAN would), nor does it blur the outlines with dozens of jumps and fetches (as any assembler must).

A *Dr. Dobb's* article by the ubiquitous Mike Gabrielson[3] gave me the clue for my first tiny-c demo, which grows, prunes and climbs binary trees. The code, listed and run on the following pages, is the real point of this article. (Since we're dealing with the C language here and not with data structures as such, readers new to the idea of binary trees are referred to Mike Gabrielson, or to Knuth).

Look at the sample run. The user builds a tree by entering strings of text—in this case, words and definitions from Grose's *Classical Dictionary of the Vulgar Tongue.*[4] The program assigns each string a key taken from a stack of available keys. Routine #4 lists the strings in alphanumeric order by traversing the binary tree. Routine #3 finds the string that matches one given by the user, and since this works for partial strings you can type in a word and get back its definition. Routine #2 finds and deletes matching strings, returning their keys to the stack. When routine #1 is used to put more strings into the tree, the new entries pick up the abandoned keys.

What's actually in the tree? Each string goes into a record of fixed length, and each record contains two extra bytes to be used as pointers. The left byte is the key of a record whose string is alphanumerically below the current string, and the right byte the key of a record whose string is higher (if there are no such records the pointers remain at zero). After the initial entries shown in the example, the tree looks like this:



Every C program is a collection of functions, and the function descriptions provided in the docfile should tell most readers enough to let them follow the code with no trouble. Functions may take arguments and return results. Variables declared in a function are local to it; variables declared outside a function are global and therefore available to any function in the program. Functions are invoked by name. In general, arguments are passed by value. When an argument is an array, the value passed is a pointer to the zeroeth element of the array (C indexes from zero).

Pointers deserve more explanation, especially since tiny-c's pointer syntax is quite different from standard C's. If we declare this array:

int vec(n)

then *vec* (the array's name without a subscript) is a pointer variable whose value is the address of *vec(0)*. The address of *vec(1)* is given by the expression *vec + 1*; the address of *vec(n)* is *vec + n*. Declaring an array of one element, like *int x(0)*, effectively gives us a pointer *x* to the single value *x(0)*.

Why use pointers at all? Since C calls by value, a called routine can't pass altered arguments back to the calling routine. This is fine—it keeps subroutines from holding arguments hostage the way they do in FORTRAN, PASCAL, PL/1, etc. Still, it does seem to limit the amount of work a function can do, since no function may return more than one result. Pointers are a way around this limitation. Give a function the pointer to a value and it can change that value by storing something else in its place. The process is so explicit that accidental side effects are unlikely.

And in tiny-c the process is so obscure that an illustration is in order. Consider these two functions:

```
callingsfunction [      /* main function, no arguments
   int x(0)             /* declare integer array of 1 element
   x(0) = 1             /* assign that element a value of 1
   calledfunction(x)    /* x, pointer to x(0), used as argument
   pn x(0)              /* print current value of x(0)
]                       /* close function
/*
calledfunction int arg(0) [     /* arg = int array of 1 element
   arg(0) = 2
]
```

callingfunction declares an integer array *x(0)* with one element, and assigns that element the value 1. Then it calls *calledfunction*. As we see from the header of *calledfunction*, it expects one argument (indicated by the dummy *arg(0)*), and it expects that argument to be an array. But arrays are identified by the address of their zeroeth element, so whatever value *calledfunction* receives in its argument will be taken as the address of *arg(0)*. This means that *x* in the calling function and *arg* in the called function have the same value and point to the same address. When *calledfunction* executes the assignments *arg(0) = 2* it puts the value 2 in the place where *callingfunction* put the value 1. The argument itself (which is a pointer to *x(0)*) hasn't been changed, but the array's contents have been, as we learn when *callingfunction* prints *x(0)*:

```
>.callingsfunction
2
```

A few words about syntax and you should be ready to read code. Every function has a header, which consists of its name and any arguments. Data types of the arguments must be declared, as must the data types of the local and global variables. The two permissible data types in tiny-c are integer (int) and character (char).

Blanks are used as separators; otherwise they're ignored. Statements are separated by carriage returns or by semicolons. Statements may be grouped inside square brackets, whose placement is a matter of style so long as they nest properly. Comments begin with /* and end with a carriage return.

To execute a function at the console, type a period, then the function's name. This example begins with *demo*, which eventually calls every function in the code.

# # #

References

1. Donald E. Knuth, *The Art of Computer Programming*, 2nd edition. Addison-Wesley, 1973.
2. Ibid., vol. 1, x-xi.
3. Mike Gabrielson, "Binary Tree Manipulation on the 8080," *Dr. Dobb's* #30 (Nov.-Dec. 1978), 15-17.
4. Captain Francis Grose, *A Classical Dictionary of the Vulgar Tongue*, 3rd edition. London, 1796.

tiny-c LISTING

```
>,p 999

/* * * * * * * * * * * * * *
/*  BINARY TREE DEMO
/*  Copyright (c) 2/23/1979
/*     Les Hancock
/*
/*  { Knuth 2.3.1, 6.2.2 }
/*
int maxptr, maxcolumn, n, root
int left, right, text, null
int base(1), avail(16 - 1)
char tree ((16 * 64) - 1)
/*
compare char string1(0), string2(0)
   int n [
   int counter
   while (n - counter) [
      if (string1(counter) < string2(counter)) return -1
      if (string1(counter) > string2(counter)) return +1
      counter = counter + 1
   ]
   return null
]
/*
delete [
   int ptr(0), k, preptr(0), preres(0), res, newptr, column
   char string(maxcolumn - 3)
   pl '---> Deletion <---'
   while (n) [   /* else tree empty, all ptrs used
      pl ''
      k = getstring (string)
      if (k == null) return
      res = sniff (string, ptr, k, preptr, preres)
      if (res) ps 'Not found.'
      else [
         newptr = rub (ptr(0))
         column = maxcolumn + 1
         while (column = column - 1) [
            tree(sub(ptr(0), column)) = null
         ]
         if (preptr(0)) [  /* this wasn't the root record
            tree(sub(preptr(0),((preres(0)+3)/2))) = newptr
         ]
         else root = newptr  /* reset root pointer
         push ptr(0)   /* decrements <n>
         ps 'Record'
         pn ptr(0)
         ps ' deleted.'
      ]
   ]
   pl ''
   pl 'Table empty.'
   pl ''
   initialize
]
/*
demo [
   int choice
   initialize
   pl ''
   pl 'DEMO: BINARY TREE'
   pl ''
   pl 'Want quick description? '
   if (yourchoice() == 'y') printmenu
   while (choice = getchoice()) [
      if (choice == 1) insert
      else if (choice == 2) delete
      else if (choice == 3) search
      else if (choice == 4) list
      else if (choice == 5) write
      else if (choice == 6) read
   ]
]
/*
format int number [
   int spaces
   pn number
   if (number < 10) spaces = 8
   else if (number < 100) spaces = 7
   else if (number < 1000) spaces = 6
   while (spaces = spaces - 1) ps ' '
]
/*
getchoice [
   pl 'Routine: '
   return gn
]
/*
getname char fname(13) [
   int k
   k = gs (fname)
   if (k > 14) [
      pl 'Name too long.'
      return 1
   ]
   return null
]
/*
getstring char string(0) [
   int k
   pl ''
   ps 'Enter string: '
   k = gs (string)
   while (k > (maxcolumn - 2)) [
      pl 'String exceeds max length of '
      pn maxcolumn - 2
      ps '; re-enter.'
      pl ''
      k = gs (string)
   ]
   return k
]
/*
initialize [
   null = 0
   root = 1
   left = 1
   right = 2
   text = 3
   maxptr = 16  /* 0 <= maxptr <= 127
   maxcolumn = 64
   /* 2 bytes in each element of <base>,
   /* maxptr*2 in <avail>
   avail = base + (2 * 2)
   tree = base + (2 * 2) + (2 * maxptr)
   n = maxptr + 1
   while (n = n - 1) avail(n - 1) = n
]
/*
insert [
   int k, ptr(0), res, dummy(0), newptr
   char string(maxcolumn - 3)
   pl '---> Insertion <---'
   while (n < maxptr) [   /* else no more ptrs available
      pl ''
      k = getstring (string)
      if (k == null) return
      res = sniff (string, ptr, k, dummy, dummy)
      if (res == 0) ps 'Record exists.'
      else [
         newptr = pop   /* increments global <n>
         /* If first record, skip pointer assignment.
         if (newptr != root) [
            tree(sub (ptr(0), ((res + 3) / 2))) = newptr
         ]
         move (string, (tree + sub (newptr, text)))
         ps 'Key ='
         pn newptr
      ]
   ]
   pl ''
   pl 'Table full.'
   pl ''
]
/*
list [
   int index(0), ptr(0), stack(maxptr - 1)
   pl ''
   if (n) [      /* else nothing to list
      pl 'KEY        TEXT'
      pl ''
      index(0) = -1
      ptr(0) = root
      traverse index, ptr, stack
      pl ''
   ]
   else [
      ps 'Nothing to list.'
      pl ''
   ]
]
/*
pop [
   int nextptr
   nextptr = avail(n)
   n = n + 1
   return nextptr
]
/*
printmenu [
   pl '---> Menu of Routines <---'
   pl ''
   pl '0) Halt'
   pl '1) Insert new entries'
   pl '2) Delete entries'
   pl '3) Search for given string'
   pl '4) List tree in inorder'
   pl '5) Write tree to disk'
   pl '6) Read tree from disk'
   pl ''
   pl 'Routines 1, 2 and 3 prompt for strings,'
   pl 'which musn't exceed'
   pn maxcolumn - 2
   ps ' characters.'
   pl ''
   pl 'Routine 1 enters new strings into the tree,'
   pl 'refusing any that are already there.'
   pl ''
   pl 'Routine 2 deletes records which begin with'
   pl 'the string entered.'
   pl ''
   pl 'Routine 3 searches the tree for a match to'
   pl 'the text you enter."
   pl ''
   pl 'To quit any of those routines, do a carriage'
   pl 'return without entering any text.'
   pl ''
   pl 'Caveats.  (1) No two records may be the same.'
   pl 'This means that if you enter one with the text'
   pl ''ABCDEFG' you can't enter another with text 'ABC'.'
   pl '(2) When you specify a string for searching'
   pl '(in routines 2 and 3), the record found will be'
```

```
    pl "the first one entered which begins with the given"
    pl "string.  If you entered 'ABCDEFG' before 'ABCD'"
    pl "and you ask routine 2 to delete 'ABC', the record"
    pl "deleted is 'ABCDEFG'."
    pl ""
    pl "At present the tree's contents cannot exceed"
    pn maxptr
    pl "records."
    pl ""
/*
push int oldptr [
    n = n - 1
    avail(n) = oldptr
    return n
]
/*
read [
    pl "---> Read <---"
    pl ""
    char fname(13)
    pl "Enter name of input file: "
    if (getname (fname)) return
    readfile (fname, base, tree + sub (maxptr, maxcolumn),1)
    n = base(0)
    root = base(1)
]
/*
rub int ptr [
    int r, s, t
    t = ptr
    if (tree(sub(t, right)) == null) return tree(sub(t, left))
    if (tree(sub(t, left)) == null) return tree(sub(t, right))
    r = tree(sub(t, right))
    if (tree(sub(r, left)) == null) [
        tree(sub(r, left)) = tree(sub(t, left))
        return r
    ]
    s = tree(sub(r, left))
    while (tree(sub(s, left))) [
        r = s
        s = tree(sub(r,left))
    ]
    tree(sub(s, left)) = tree(sub(t, left))
    tree(sub(r, left)) = tree(sub(s, right))
    tree(sub(s, right)) = tree(sub(t, right))
    return s
]
/*
search [
    int res, k, ptr(0), dummy(0)
    char string(maxcolumn - 3)  /* <string> indexed from 0
    pl "---> Search <---"
    while (1) [
        pl ""
        k = getstring (string)
        if (k == null) return
        res = sniff (string, ptr, k, dummy, dummy)
        if (res) ps "Not found."
        else [
            ps "Found at "
            pn ptr(0)
            ps ": "
            ps tree + sub (ptr(0), text)
        ]
    ]
]
/*
sniff char string(0)
    int ptr(0), k, preptr(0), preres(0) [
    int temp, res
    preptr(0) = null
    preres(0) = null
    ptr(0) = root
    while (1) [
        res = compare (string, (tree + sub (ptr(0), text)), k)
        if (res == null) return res
        temp = tree(sub (ptr(0), ((res + 3) / 2)))
        if (temp == null) return res
        preptr(0) = ptr(0)
        preres(0) = res
        ptr(0) = temp
    ]
]
/*
sub int row, col [
    return (col - 1) + (maxcolumn * (row - 1))
]
/*
traverse int index(0), ptr(0), stack(maxptr - 1) [
    index(0) = index(0) + 1  /* bump stack
    stack(index(0)) = ptr(0)  /* save pointer
    if(tree (sub (ptr(0), left))) [  /* if left subtree exists
        ptr(0) = tree(sub (ptr(0), left))
        traverse index, ptr, stack
    ]
    pl ""   /* print record being visited
    format ptr(0)
    ps tree + sub (ptr(0), text)
    if (tree (sub (ptr(0), right))) [   /* if right subtree
        ptr(0) = tree(sub (ptr(0), right))
        traverse index, ptr, stack
    ]
    /* If stack is empty, we're done.
    if (index(0) == 0) return
    index(0) = index(0) - 1    /* pop <stack>
    ptr(0) = stack(index(0))
]
/*
```

```
write [
    char fname (13)
    pl "--> Write <---"
    pl ""
    pl "Enter name of output file: "
    if (getname (fname)) return
    base(0) = n
    base(1) = root
    writefile (fname, base, tree + sub (maxptr, maxcolumn), 1)
]
/*
yourchoice [
    char c
    if ((c = gc()) <= 'Z') c = c + 'a' - 'A'
    return c
]
>
```

SAMPLE RUN

```
D,tinyc    call out tiny-c
 ***  TINY-C    VERSION 1.0  ***
COPYRIGHT 1977, T A GIBSON
>,r tree.ccc    read file containing source code
 7812
 0 336 7812 4188
>,demo      execute main function


DEMO: BINARY TREE

Want quick description? yes

---> Menu of Routines <---

0) Halt
1) Insert new entries
2) Delete entries
3) Search for given string
4) List tree in inorder
5) Write tree to disk
6) Read tree from disk

Routines 1, 2 and 3 prompt for strings,
which mustn't exceed 62 characters.

Routine 1 enters new strings into the tree,
refusing any that are already there.

Routine 2 deletes records which begin with
the string entered.

Routine 3 searches the tree for a match to
the text you enter.

To quit any of those routines, do a carriage
return without entering any text.

Caveats.  (1) No two records may be the same.
This means that if you enter one with the text
'ABCDEFG' you can't enter another with text 'ABC'.
(2) When you specify a string for searching
(in routines 2 and 3), the record found will be
the first one entered which begins with the given
string.  If you entered 'ABCDEFG' before 'ABCD'
and you ask routine 2 to delete 'ABC', the record
deleted is 'ABCDEFG'.

At present the tree's contents cannot exceed 16
records.

Routine: 1

---> Insertion <---

Enter string: NUMBSCULL.  Stupid fellow.
Key = 1

Enter string: KICK THE BUCKET.  To die.
Key = 2

Enter string: ELBOW GREASE.  Labour.
Key = 3

Enter string: JILTED.  Rejected by a woman.
Key = 4

Enter string: GROG.  Rum and water.
Key = 5

Enter string: ZANY.  The jester, or merry Andrew.
Key = 6

Enter string: DISGRUNTLED.  Offended, disoblised.
Key = 7

Enter string: BURNT.  Poxed or clapped.
Key = 8

Enter string: QUASH.  To suppress, annul, or overthrow.
Key = 9
```

```
Enter string: AGAINST THE GRAIN.  Unwilling.
Key = 10

Enter string: GAMBS.  Thin, ill-shaped legs.
Key = 11

Enter string: TROTTERS.  Feet.
Key = 12

Enter string: TAP.  A gentle blow.
Key = 13

Enter string: CRUMMY.  Fat, fleshy.
Key = 14

Enter string: NODDLE.  The head.
Key = 15

Enter string: LICK.  To beat.
Key = 16

Table full.

Routine: 4

KEY       TEXT

 10       AGAINST THE GRAIN.  Unwilling.
 8        BURNT.  Poxed or clapped.
 14       CRUMMY.  Fat, fleshy.
 7        DISGRUNTLED.  Offended, disoblised.
 3        ELBOW GREASE.  Labour.
 11       GAMBS.  Thin, ill-shaped legs.
 5        GROG.  Rum and water.
 4        JILTED.  Rejected by a woman.
 2        KICK THE BUCKET.  To die.
 16       LICK.  To beat.
 15       NODDLE.  The head.
 1        NUMBSCULL.  Stupid fellow.
 9        QUASH.  To suppress, annul, or overthrow.
 13       TAP.  A gentle blow.
 12       TROTTERS.  Feet.
 6        ZANY.  The jester, or merry Andrew.

Routine: 3

---> Search <---

Enter string: GROG
Found at  5: GROG.  Rum and water.

Enter string: KICK
Found at  2: KICK THE BUCKET.  To die.

Enter string: XXXXXX
Not found.

Enter string: (c/r)

Routine: 2

---> Deletion <---

Enter string: TROTTER
Record 12 deleted.

Enter string: NUMBSCULL
Record 1 deleted.

Enter string: (c/r)

Routine: 1

---> Insertion <---

Enter string: TAT.  Tit for tat; an equivalent.
Key = 1

Enter string: (c/r)

Routine: 4

KEY       TEXT

 10       AGAINST THE GRAIN.  Unwilling.
 8        BURNT.  Poxed or clapped.
 14       CRUMMY.  Fat, fleshy.
 7        DISGRUNTLED.  Offended, disoblised.
 3        ELBOW GREASE.  Labour.
 11       GAMBS.  Thin, ill-shaped legs.
 5        GROG.  Rum and water.
 4        JILTED.  Rejected by a woman.
 2        KICK THE BUCKET.  To die.
 16       LICK.  To beat.
 15       NODDLE.  The head.
 9        QUASH.  To suppress, annul, or overthrow.
 13       TAP.  A gentle blow.
 1        TAT.  Tit for tat; an equivalent.
 6        ZANY.  The jester, or merry Andrew.

Routine: 0

>,X    exit PPS (tiny-c's editor/operating system)

DONE
>>>,X    exit tiny-c

D.
```

DOCFILE LISTING

```
D.type treedoc.ccc


   DESCRIPTION OF BINARY TREE DEMO FUNCTIONS

< compare > :  Compare the first <n> characters of <string1> with
the first <n> characters of <string2>.  If <string1> < <string2>,
return -1.  If <string1> > <string2>, return +1.  If
<string1> == <string2>, return 0.

< delete > :  Delete records from the tree.  Note that <ptr(0)>,
<preptr(0)> and <preres(0)> are passed to <sniff> and to <rub> as
pointers.  The reason for this is that C calls by value.  A called
function can't change values local to the calling function, but
it can change the contents of the location in memory where a value
is stored, which comes to the same thing.

< demo > :  Main function.

< format > :  Formatting function for use with <traverse>.

< gc > :  Library function.  Gets string from terminal, returns
first character.

< getchoice > :  Prompt for and return user's choice of routines.

< getname > :  Get file name for <write> and <read>.

< getstring > :  Get string from terminal and return number of
characters in it.  If the user enters only a carriage return,
the number of characters is zero.

( globals ) :  Described under the rubric <initialize>.  Unlike
big C, tiny-c can't dynamically allocate space for global arrays.
If it could, we would declare avail(maxptr - 1) and
tree((maxptr * maxcolumn) - 1).

< gs > :  Library function.  Gets string of characters from the
terminal.  Returns number of characters read.

< initialize > :  Set up globals.  <root> is pointer to the
current root of the tree.  <left> and <right> are the column
numbers of left and right pointers in each record.  <text> is
the number of the column in which text begins.  <maxptr> is the
value of the highest allowable pointer, <maxcolumn> that of the
highest column.  Because left and right pointers are stored as
signed integers in a single byte, <maxkey>'s range of values is
1 - 256.  In the code as written the limits are 1 - 127, since
we don't offset negative values.  Note that although C uses
origin 0, we're indexing pointers and columns from 1.  <avail>
is a stack whose stack pointer is the global <n>.  <n> points
to the next available pointer number in <avail>, which is
initialized to the values 1, 2, 3, ... , maxptr.  <tree> is of
course the tree itself.  The arrays <avail> and <tree> are made
contiguous to the array <base> so that all parameters and records
may be saved in a single file write and restored by a single
read.

< insert >:  Insert records into the tree.  The local variable
<dummy> is used to fill out the list of arguments in the call to
<sniff>.  Where <sniff> finds the target string in the tree, <insert>
prints 'Record exists.'  When the string is not found, it's in-
serted in the table.  Insertion is done as follows.  If the
record to be inserted is not the first (root) record, find the
(left or right) pointer in the tree which should point to it.
That pointer's value is changed from 0 to the next value available
in the stack <avail>.  The stack pointer <n> is bumped.  The
string is moved into the text area of the new record, and the
new pointer (or key) is printed at the terminal.

< list > :  List records in inorder.  All arguments are pointers.
See <traverse> for details.

< move > :  Library function.  Takes two arguments, both pointers.
Moves bytes from location given by first argument to location
given by second, terminating at a null.

< pl > :  Library function.  Prints string on a new line.

< pn > :  Library function.  Prints an integer at the terminal.

< pop > :  Pop next available pointer value from <avail> when a
new record is inserted.

< ps > :  Library function.  Prints string until halted by
null byte.

< printmenu > :  Optional help function.

< push > :  Push released pointer value onto <avail> when a
record is deleted.

< read > :  Restore saved tree from a file.  Note that only the
parameters <n> and <root> are restored.  The tree must conform
to the <maxptr> and <maxcolumn> values already set by
<initialize>.

< rub > :  Remove from the tree the record indicated by <ptr>
and return a pointer (<newptr>) to the record which replaces it
in the tree structure.  <delete> will reset the (left or right)
pointer that pointed to the record rubbed.

< search > :  Find records specified by initial strings.  The
local variable <dummy> is used to fill out the list of arguments
in the call to <sniff>.

< sniff > :  Sniff through the tree looking for a record whose
initial <k> characters match the <k> characters of <string>.
If a match is found, return zero; <ptr(0)> points to the matching
record, <preptr> points to the last record compared, and <preres>
is -1 or +1 depending on whether the string is < or > the last
record compared.  If no match, <ptr(0)> points to the last record
```

Continued on page 36

PROGRAMMING PASTIMES AND PLEASURES

BY CHARLES WETHERELL

© 1979, Charles Wetherell

OBSERVATION VERSUS ANALYSIS: TWO PROBLEM SOLVING METHODS

Analysis

In my first column (January), I proposed a little draw poker machine as an interesting programming project. Although this subject no doubt led to my evil reputation for frivolity, there was a deeper, hidden motive for the column. Readers who persevered to the end noticed that the real problem was not to write a poker game program. Rather, it was to answer a question in economic strategy—to wit, what are all the possible choice points in this version of poker and what is the optimal choice of alternatives for each situation?

Such observant readers would also have noticed that I claimed there were two different ways to solve the problem. The first way, which I anticipate will leap to mind, is to do an exhaustive *analysis*. To illustrate, assume we are interested in the expected outcome of a draw when we hold the hand

♥2 ♥3 ♥4 ♥5 ♦9.

Then if the choice of strategy is to discard the diamond nine, hoping for a flush or a straight, the possible outcomes are:

♥6	straight flush	Probability = 1/47
♣ ♦ ♠ 6's[	straight	Probability = 3/47
♥7-A	flush	Probability = 8/47
] Any 2-5	pair	Probability = 12/47
] All others	bust	Probability = 23/47

Remember that only 47 cards are available for the draw since five cards are used for the initial hand. This analysis is quite straightforward and the results are perfectly accurate. Now that the probabilities are known, the table of payoffs can be consulted to compute the expected value of drawing one card to this fourflush. A similar analysis can be carried out for the fourflush combinations where an outside straight or no straight is possible.

But the analytic approach breaks down when confronted with a hand like

♦2 ♣6 ♥5 ♣J ♠9.

There are so many possible ways to hold for the draw, and so many possible outcomes, that even to list them all is a huge task; probability calculations would be even worse. And although this is typical bust hand picked up on the deal, the exact distribution of suits and ranks significantly alters the probabilities of flushes and straights after the discard. So the analysis will have to be repeated for every possible bust pattern. Certainly a computer program could be a help in such analysis, but I wouldn't want to have to write such a program.

Simulation

The second solution technique, *simulation*, is much less likely to be familiar. Mathematically trained programmers are particularly apt not to try simulation if an analytic solution appears at all possible. I can easily illustrate simulation with our fourflush hand example. Take a deck of cards, remove the heart deuce through five and the diamond nine, shuffle the deck thoroughly, pull off the top card, and use it as your draw. Now replace the drawn card in the deck, shuffle again, draw again, tally again. Repeat this process, keeping track of the number of repetitions. After several thousand trails you will have a fair estimate of the probabilities of each of the types of payoff hands.

Of course, thousands of shuffles will take you a fair time, not to mention the problem of guaranteeing randomness in your shuffling. The computer, however, can relieve you of your chore if you can develop a *model* of a deck of cards and if you can operate that model with a program. And now we have reached the point of this column.

A good model for a deck of cards is just a bit vector numbered from one to 52. Mark the deuce through five of hearts and the nine of diamonds as in use, possible by turning on the corresponding bit. Use a random number generator to pick a number between 1 and 52—that is the card you have drawn (throw it back and try again if it is in use). Add the card to the fourflush and evaluate the resulting hand. Assuming your random number generator is in good shape, a few thousand cycles shouldn't take long and your results should be almost

Continued on pg. 36

further thoughts on ackermann's function

BY LLOYD RICE
Computalker Consultants
1730 21st Street
Santa Monica, California 90404

Like author Gabrielson (*DDJ* #31), I also encountered Ackermann's Function years ago, and also gave my 5 minutes penance to A(2,3) and beyond. Like so many others, I worked out the table as far as somewhere in the 4th row and then set it aside for other diversions. For the sake of completeness and reference, I will repeat the recursive definition here:

$$A(0,n) = n+1$$
$$A(m, 0) = A(m-1, 1)$$
$$A(m, n) = A(m-1, A(m, n-1))$$

Gabrielson's presentation of a bit of software and the dangling mystery of the final question in his article did for me what the acquisition of an 8080 system had not done, namely implanting the desire to explore that beckoning 5th row. Once again, I took pencil in hand and proceeded to explore the multiple recursions expressed here. A(4, 1) is not so bad. A(4, 2) is the killer.

Inasmuch as I am not entirely possessed to view the world recursively, one of my first urges was to try to express this most famous of recursive functions in a direct form, that is, something that could be computed directly from a non-recursive algebraic equation. With this goal in mind, I set out to discover such a general algebraic statement of the Ackermann Function of m and n.

As stated by Gabrielson, the first few rows come easy. The first row is, by definition, A(0, n) = n+1. With just the slightest amount of inspection, one sees that the second row, A(1, n) is given by n+2. Nothing complicated yet. And again, after a small amount of shuffling of terms, we discover the equation for the third row as A(2, n) = 2n+3. Great! Carrying on with determination and nerves of steel we plunge into the fourth row. At first there seems to be a stumbling block; the products pile up into an apparently never-ending series. I first expressed this as a summation of decreasing powers of 2, when suddenly, like the proverbial bolt out of the blue, the whole thing collapsed down into the elegant form shown in table 1.

$$A(0, n) = n+1$$
$$A(1, n) = n+2$$
$$A(2, n) = 2n+3$$
$$A(3,n) = 2^{n+3} - 3$$

Table 1. Ackermann's Function for m = 0, 1, 2, 3

Now with hopes buoyed by this optimistic sign, we decide to continue on and tackle the fatal fifth row. Working as before, we express a general entry in that row in terms of its predecessor:

$$A(4, n) = A(3, A(4, n-1))$$

From our table of row equations previously worked out, we have

$$A(3, n) = 2^{n+3} - 3,$$

from which

$$A(4, n) = 2^{A(4, n-1)+3} - 3$$

For the case where n = 0, we know this reduces to

$$A(4, 0) = A(3, 1) = 2^4 - 3 = 13$$

For n = 1, we have the rather larger result

$$A(4, 1) = 2^{13+3} - 3 = 2^{16} - 3 = 65533$$

However, on A(4, 2), as we carry on this process, we arrive at the formidable result

$$A(4, 2) = 2^{65536} - 3$$

This is indeed just 2 counts less than the largest integer value that could be stored in a word 8K bytes long! Truly an astronomical number. Undaunted by mere numerical magnitude, we push on to see what lies in store ahead. At first, A(4, 3) looks hopeless of further simplification than to write

$$2^{(2^{65536})} - 3$$

Now we come to the point of this entire matter. There seems to be an intriguing parallel between these results in the

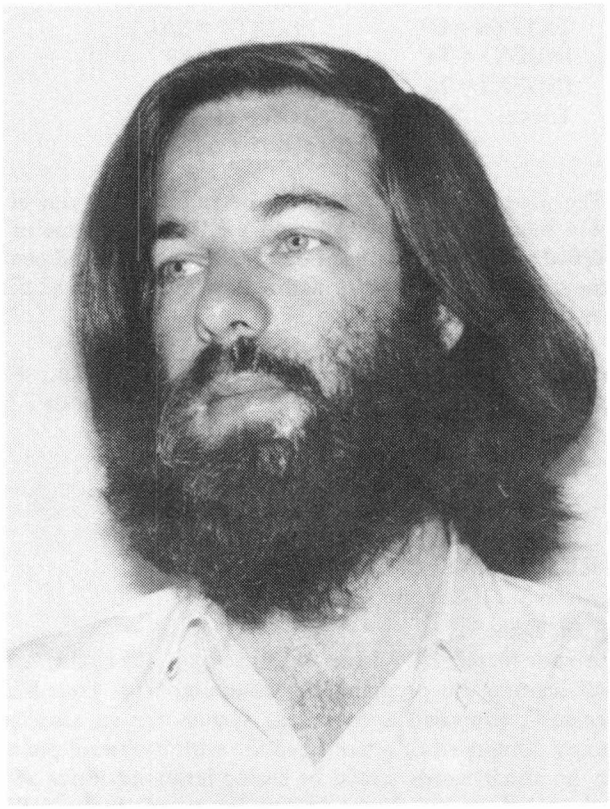
fifth row and the elegant 4th row expression, $2^{n+3} - 3$. Looking more carefully, we see that $A(4, 0) = 2^{(2^2)} - 3$, with a total of three 2's in the expression. Likewise,

$$A(4, 1) = 2^{\left(2^{(2^2)}\right)} - 3,$$

for a total of four 2's. Aha!! It would seem that A(4, n) would involve this sort of stacking of exponents (the parentheses are important) such that there would be a total of n+3 2's in the expression. Curious and interesting!?!?

At this point, let us take a slight digression into the quirks of computational symbolism. Is it indeed pure chance that the operation expressed in BASIC by the up-arrow "↑" is stated in FORTRAN as a doubling of the multiplication operator, that is, as "**"? What 2**n really means, of course, is that a collection of n 2's are to be combined by repeated application of the "*" operation.

$$2**3 \equiv 2*2*2$$



Lloyd Rice has been interested in electronics since he was an active amateur radio operator (K9DSH) in high school 25 years ago. He has studied languages and linguistics for 20 years, and has worked with computers for nearly as long. These interests led to several moments of indecision about a college career, which resulted in a degree in linguistics from UCLA. After 10 years as a programmer in the Phonetics Laboratory at UCLA, he co-founded and is now senior partner of Computalker Consultants, a Santa Monica, California based manufacturer of computer speech hardware and software.

Let us apply this same logic to the exponentiation operator "↑". We will define 2↑↑4 to be equal to

$$2^{\left(2^{(2^2)}\right)},$$

that is, four 2's combined by repeated application of "↑". Since the order of evaluation is important (exponentiation is neither associative nor commutative), we will assume parentheses as shown, that is, evaluation will proceed from top down. In general, we will define

$$x{\uparrow\uparrow}n \equiv x^{\left(x^{\left(x \ldots {}^{(x^x)}\right)}\right)}$$

where there are n x's. Using this representation, we can now conveniently state the expressions for the 4th and 5th rows of Ackermann's Function as

$$A(3, n) = 2{\uparrow}(n+3) - 3$$
$$A(4, n) = 2{\uparrow\uparrow}(n+3) - 3$$

Incidentally, I do not take full credit for this extension of the exponentiation operator. I seem to recall a similar discussion by Martin Gardner in his column in *Scientific American* several years ago, although I could not locate the reference.

If we now proceed to apply this same strategy to the next row, we have

$$A(5, n) = A(4, A(5, n-1))$$
$$= 2{\uparrow\uparrow}(A(5, n-1)+3) - 3$$

By definition,

$$A(5, 0) = A(4, 1)$$
$$= 2{\uparrow\uparrow}4 - 3$$

Therefore, we can write A(5, 1) as follows:

$$A(5, 1) = 2{\uparrow\uparrow}(2{\uparrow\uparrow}4 - 3+3) - 3$$
$$= 2{\uparrow\uparrow}(2{\uparrow\uparrow}(2{\uparrow\uparrow}2)) - 3$$

Similarly,

$$A(5, 2) = 2{\uparrow\uparrow}(2{\uparrow\uparrow}(2{\uparrow\uparrow}(2{\uparrow\uparrow}2)) - 3+3) - 3$$
$$= 2{\uparrow\uparrow}(2{\uparrow\uparrow}(2{\uparrow\uparrow}(2{\uparrow\uparrow}2))) - 3$$

It seems clear enough by now that what we really want is a further extension of the nomenclature, specifically . . .

$x{\uparrow\uparrow\uparrow}n = x{\uparrow\uparrow}(x{\uparrow\uparrow}(\ldots (x{\uparrow\uparrow}x)))$ where there are n x's.

The sixth row of Ackermann's Function can then be written as

$$A(5, n) = 2{\uparrow\uparrow\uparrow}(n+3) - 3$$

I leave you, Dear Reader, with the suspicion, although I have no proof, that Ackermann's Function for m>2 can be generally stated as

$A(m, n) = 2{\uparrow\uparrow\uparrow} \ldots {\uparrow}(n+3) - 3$ where there are m−2 "↑"s.

Letters

FOLLOWUP ON LISP INTERPRETER

Dear Suzanne:

This letter is a brief set of followup notes on my LISP interpreter (November/December 1978) which answer a number of questions I have received since then. I still encourage people who have troubles to contact me for suggestions, but I hope that note will answer most problems.

There are a few source errors which have been reported. These are:

LINE	NOW	THERE	SHOULD BE
260	ANI	A,3	ANI 3
333	JZ	NOTLAB	JZ NOTLAM
557	XRA	A	(deleted)

While stated in the article, I reiterate that the I/O routines MUST save and restore all registers (except input must change the PSW). Note that many systems (such as CPM) restore none of the registers except the stack pointer, and will require short interface routines. The I/O routines also assume the high order bit is zero. Lower case is recognized by the interpreter but the built-in function names are in upper case. The two equates for MEM and FREBLK on lines 3 and 4 control the area of memory used for allocating list structures. MEM is the address of the lower bound of the block of free memory. FREBLK is the count of the number of 256 byte pages of memory available. The values in the listing define a 16K byte area from 700H to 4700H. Both the interpreter and this free area may be moved around but must be kept below 8000H. It is possible to use several separate free areas in memory. Anyone interested in doing this may write me.

Several people have also asked me for simple sample inputs for debugging. Here are a few:

CAR (((A.B.) . C)) b
which gives >> (A.B)
and CDR (((A.B.).C)) b
which gives >> C
where b denotes a blank.

Users should note that there is an extra set of parentheses around the list of arguments to a function. Thus in the examples here, the one argument actually passed to CAR and CDR is ((A.B.) . C).

Sincerely,
Darrel J. Van Buer

1522 Brockton Ave., Apt. 5
Los Angeles CA 90025

KIM RENUMBER

Dear *Dr. Dobb's*:

The PET BASIC Renumber program (*DDJ* #31, January, 1979) is easily modified to renumber KIM-1 Microsoft BASIC programs with the following changes:

VARTAB = 7A	CHRGET = OOCO
FACTO = BQ	CHRGOT = OOC6
LINNUM = 19	FINI = 23EE
TXTTAB = 78	LINGET = 28B8
TXTPTR = C7	FLOATC = 3AC1
INDEX1 = 74	FOUT = 3C69
INDEX2 = 76	
Tokens: GOTO = 88	RUN = 89
GOSUB = 8C	THEN = A1

The program will also need to be relocated to run in the KIM-1 memory space. I will send a KIM format tape of the modified program organized at 0200 HEX to interested persons if they will send a cassette and return postage, or $2.50 to cover duplication cost.

Sincerely,
Fred J. Monsour

3301 S.W. Archer Rd., #26A
Gainesville, Florida 32608

IN RESPONSE TO KLAUS HOLTZ

Dear *Dr. Dobb's*:

In your March 1979 issue an article by Klaus Holtz, "Build a Self-learning No-programming Computer With Your Microprocessor", proposed a data base system for an associative memory comprised of a tree structure with backward pointing links in which words would be stored letter-by-letter with a novel technique to share storage for words with identical initial letter sequences. It was claimed in the article that this structure would result in a decrease in storage requirements as against other methods of allocating memory.

The mathematics of the situation do not bear this out. In the scheme each letter in memory is represented by a one byte character representation and a two byte pointer; the saving is realized in that concurrent initial letter sequences in words need only be stored once. The saving then, depends on the extent to which such identical combinations of letters appear in English. A count made on twenty random columns

in a desk dictionary shows that the concurrency is about 40-50%. Since the average number of letters in an English word is eight, to which a one byte space terminator is added by the algorithm, an average word would be stored as 50% of nine letters, about 4 letters per word. Even the favorable estimate of 4 letters shows that a 16 bit pointer is inadequate. Sixteen bits can address 64K letters or 16 thousand words. A desk dictionary contains 50,000 words which would need an 18 bit address. For the type of application contemplated, a 24 bit address should be used to include about four million words.

Each letter, then, should be represented by an 8 bit character and a 24 bit address, a total of 4 bytes. A word needs four letters on the average for a total of 16 bytes.

The simplest alternative storage technique would be to store each word in a fixed 10 byte block with longer words accommodated in the following block with a continuation character in the last byte of the first block. Since no pointer is required in a linear allocation, each word needs only ten bytes for a saving of 37.5% space.

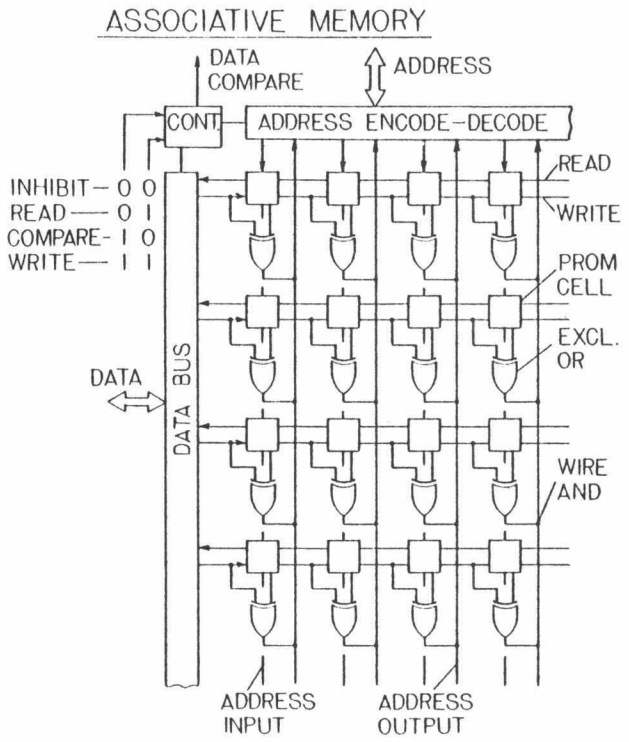
Since the system is meant to be implemented in slow mass memory, the space requirements are not really critical. Search time is. The data base is meant to be searched against input words for a match. In the Holtz algorithm each letter must be searched for in a sequential search. On the average we must search half of the remaining memory to find each of the next letters. For the average eight letter word we must search 1/2+1/4+1/8+1/16+1/32+1/64+1/128+1/256 of the tape for each word and this is over 99% of the memory. The alternative would need to search only 50% of the memory in a linear search. Of course a hashing algorithm could be used reducing the number of probes to about five, even with the simplest collision handler.

Sincerely,
Gordon Wilk

6707 Springpark Ave.
Los Angeles CA 90056

MR. HOLTZ REPLIES

The mathematics of Mr. Wilk is essentially correct, except that in my estimate the number of different letters for each word is more like 3. With a slightly different algorithm the space character could be avoided. Even though the English dictionary may contain 50,000 words, each one of us uses only a few hundred or at most a few thousand words in everyday life. The size of the pointer in this algorithm is at least one count smaller than the address in which it is stored. The address may therefore be used to scale the pointer size or, in other words, if the address is small and contains a lot of leading zeroes, so does the pointer. These leading zeroes need not be stored. The pointer may therefore be initially small and grow later with the system. I frankly believe that a 64K memory will be sufficient for the average vocabulary. If I am wrong, the memory can be expanded later. We have to raise our sight and look to the future of truly self-learning computers. Storing a vocabulary can be only the very first step. This algorithm allows the later use of an associative memory, shown below. This memory is not yet available, but as a large scale chip it could become very cheap and it would solve all speed or access problems. We should not block future progress for a few K of memory and avoid dead end solutions. Let's build self-learning computers and let us solve the problems together.



ADDING CLK TO SWTPC

Dear Suzanne:

Many game programs are (or are made) more enjoyable if quick responses are required. The program "Oregon Trail" by Dan Rawitsch in the May/June issue of *Creative Computing* is greatly enhanced by such a feature.

Unfortunately, the computerware BASIC interpreter (PROM version) I use with my SWTPC 6800/2 computer does not have the CLK (clock) timing command. I wrote a subroutine, (see Listing 1) to satisfy this need. Listing 2 incorporates the routine in a typing skill test which has been fun for us to use. Modification to embed this routine in "Oregon Trail" was straightforward.

In essence, the user command is used to transfer program control to this machine-language routine. The routine counts the number of times the computer tests for user response before an input key is depressed. The program in Listing 2 generates a randomly chosen group of letters, the number of which has been preselected by the user, and compares user responses with the random characters for accuracy. If the characters have been accurately matched, the user is told how long (in arbitrary units) he took to respond.

The program in Listing 1 is written as a subroutine and is called by line 110 in Listing 2 with the B=user (0) command. The user-pressed key causes a character to be stored in $0002. Line 110 pulls the character and concatenates it into A$. The two poke commands in line 20 set up the starting address for reference by the user command. Line 10 reads the machine language routine into the desired section of memory for its later use. Line 90 initializes the counter to zero for each test. Line 80 randomly selects the characters to be matched by the user. Line 120 recognizes asterisks as the end-of-test signal.

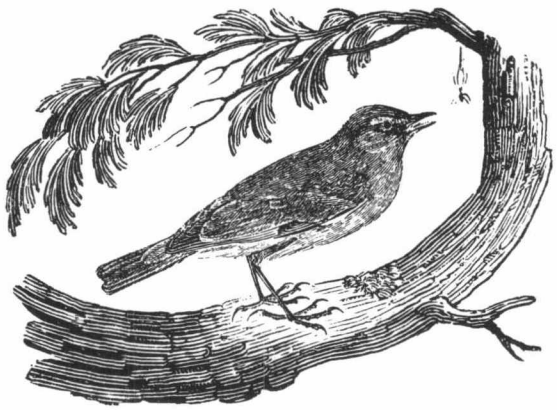
Lines 130 and 140 test for accuracy and excess response time. Line 150 captures the counter value and formats it for printing. Lines 190 through 210 are the decimal equivalent of the machine language routine in Listing 1.

Sincerely,
Chesley H. Looney

3406 Notre Dame Street
Hyattsville, MD 20783

LISTING 1

```
A048     37              PSHB
A049     BD E1 D3        JSR    SAVGET
A04C     86 15           LDAA   #$15
A04E     A7 00           STAA   X
A050     5F              CLRB
A051     A6 00     ACIAIN LDAA  X
A053     5A              DECB
A054     27 0C           BEQ    ADD
A056     47        IN2   ASRA
A057     24 F8           BCC    ACIAIN
A059     A6 01           LDAA
A05B     97 02           STAA   ATEMP
A05D     33              PULB
A05E     FE A0 10        LDX    XTEMP
A061     39              RTS
A062     7C 00 01  ADD   INC    COUNT
A065     7D 00 01        TST    COUNT
A068     26 EC           BNE    IN2
A06A     7C 00 00        INC    COUNT-1
A06D     20 E7           BRA    IN2

S113A04837BDE1D38615A7005FA6005A270C47241D
S113A058F8A601970233FEA010397C00017D0001A7
S10AA06826EC7C000020E758
```



```
LISTING 2
BASIC
•
0010 FOR W=1TO39:READ V:POKE(410+W,V):NEXT W
0020 LINE= 0:POKE(40,160):POKE(41,72)
0030 PRINT "SPILL TEST - TYPE REQUESTED CHARACTERS AS FAST AS YOU"
0040 PRINT "CAN.  YOUR SCORE WILL BE SMALLER THE FASTER YOU TYPE."
0050 PRINT "EACH WRONG ENTRY WILL INCREASE YOUR SCORE.":PRINT
0055 INPUT "HOW MANY CHARACTERS DO YOU WISH IN EACH GROUP",N:PRINT
0060 PRINT "TO END TEST, TYPE "CHR$(34)"ASTERISK"CHR$(34)".":SKIP 3
0070 INPUT "TYPE  RETURN  TO START";A$:SKIP 2
0080 LET B$="":FOR X=1TON:A=65+INT(RND(.25+1)):B$=B$+CHR$(A):NEXT X
0090 POKE 0,0:POKE 1,0
0100 PRINT "TYPE - "B$" "
0110 LET A$="":FOR X=1TON:B=USER(0):A$=A$+CHR$(PEEK(-1)):NEXT X:PRINT " "
0120 IF PEEK(2)=42 SKIP 3:GOTO 170
0130 IF A$<>B$ PRINT "WRONG!":F=F+100:GOSUB 1:GOTO 80
0140 IF PEEK(0)<3 PRINT "TOUCH TOO LONG!":E=E+1:F=F+75:GOTO 80
0150 LET C=PEEK(0)+.5*PEEK(1):PRINT C;E=E+1:F=F+C:D=INT(F/E)
0160 GOTO 80
0170 PRINT "RIGHT/WRONG = "E1"/"E1"***  AVERAGE SCORE = "D
0180 END
0190 DATA 55,189,225,211,134,21,167,0,95,166,0,90,2
0200 DATA 12,71,36,248,166,1,151,2,51,254,140,17,57
0210 DATA 124,0,1,125,0,1,39,,6,124,0,0,,,231
BASIC
•
•
```

SEARCH

Dear Doc:

Quite often in my software wanderings, I find it necessary to locate a byte or two in the middle of a program. If the program happens to be a large one, such as a BASIC interpreter, this search can be nerve-racking and tiring.

I like as much uniformity as possible in my software. However, such is not the case with commercial programs. For instance, one might use Backspace to back up and erase a character and another might use Delete. Still another uses Control "A."

If you can find the two byte CPI"X" mnemonic (FE X), all you have to do is change X from whatever it is now to whatever you want it to be. Using the example above: suppose some program uses Delete for erase and you want it to respond to Backspace instead.

First determine that the program doesn't use Backspace for something else already, then load it into its normal operating location. Modify "SEARCH" to reflect the addresses of the block of memory you want to search and make BYTE2 EQU to 07FH (Delete). You'll also want to change SCREEN and BOOT to fit your system.

All that remains is to load "SEARCH" and Execute it. Almost instantly the location of CPI 07FH (FE 7F) will appear on the screen. Using your front panel switches or your monitor, examine and change the 7F (Delete) to 08 Backspace) and you're almost done. If the address on the screen is the same as your Finish address, then CPI 7F couldn't be found.

Save the changed program back on tape or disk and you have a corrected copy that willl respond to the new Erase code. Don't forget to save "SEARCH" also or you'll have to enter it by hand again the next time you need it. You can also use this routine to hunt for any two bytes by making BYTE1 and BYTE2 equate to them.

If you are not loading this program into an assembler for modification, then you will have to make the necessary changes to the object code. For instance, if you want to run "SEARCH" on a SOL, you'd replace "CD03F8" with "CD19C0" each time that the CALL SCREEN mnemonic appears in this program. The byte that you're searching for goes into location B71B.

Sincerely,
Rod Hallen

P.O. Box 73
Tombstone, AZ 85638

```
;*****************************************************
;*                                                   *
;*      'SEARCH'    (C) COPYRIGHT 1979               *
;*  BY ROD HALLEN  P.O. BOX 23  TOMBSTONE, AZ 85638  *
;* A PROGRAM TO SEARCH A BLOCK OF MEMORY FOR A 2 BYTE *
;*  'FE' INSTRUCTION AND RETURN THE ADDRESS WHERE IT  *
;* IS LOCATED TO THE IMSAI VIO SCREEN                *
;*                                                   *
;*****************************************************

;* COMMENTS

;* ORG CAN BE SET TO ANY VACANT LOCATION WHICH HAS AT
;* LEAST 2FH BYTES OF MEMORY AVAILABLE
;* START AND FINISH ARE THE ADDRESSES OF THE BLOCK
;* YOU WANT TO SEARCH
;* SCREEN IS THE ADDRESS OF YOUR VIDEO DRIVER.  IN THE
;* 'DOS THIS WOULD BE F803H.
;* BYTE1 AND BYTE2 ARE THE 'FE ??' THAT WE ARE
;* LOOKING FOR.  ANY TWO BYTES CAN BE PUT HERE.
;* BOOT IS THE LOCATION THAT YOU WANT TO JUMP TO
;* AFTER THE SEARCH IS FINISHED

;* SYSTEM EQUATES *

B700                    ORG     0B700H  ;"SEARCH" LOCATION AND EXECUTE
0100 =          START   EQU     100H    ;BEGINNING OF SEARCH
4FFF =          FINISH  EQU     4FFFH   ;END OF SEARCH
F803 =          SCREEN  EQU     0F803H  ;VIO SCREEN DRIVER ADDRESS
00FE =          BYTE1   EQU     0FEH    ;CPI ASCII CODE
0040 =          BYTE2   EQU     40H     ;THE CPI CHARACTER WE WANT
0000 =          BOOT    EQU     0       ;MONITOR ENTRY POINT

;* B000 ASSEMBLY LANGUAGE PROGRAM

B700 210001             LXI     H,START ;LOAD BEGINNING OF BLOCK
B703 11FF4F             LXI     D,FINISH;LOAD END OF BLOCK

B706 7E         LOOP    MOV     A,M     ;GET A BYTE
B707 FEFE               CPI     BYTE1   ;IS IT "FE" ?
B709 23                 INX     H       ;BUMP BYTE POINTER
B70A F5                 PUSH    PSW     ;SAVE BYTE AND FLAG
B70B 7A                 MOV     A,D     ;HIGH ORDER END BYTE
B70C BC                 CMP     H       ;COMPARE WITH PRESENT
B70D C215B7             JNZ     LOOP1   ;GET ON WITH IT
B710 7B                 MOV     A,E     ;LOW ORDER END BYTE
B711 BD                 CMP     L       ;COMPARE WITH PRESENT
B712 CA22B7             JZ      FAIL    ;DIDN'T FIND IT

B715 F1         LOOP1   POP     PSW     ;RESTORE BYTE AND FLAG
B716 C206B7             JNZ     LOOP    ;IF NOT "FE" GET ANOTHER BYTE
B719 7E                 MOV     A,M     ;GET BYTE FOLLOWING "FE"
B71A FE40               CPI     BYTE2   ;IS IT THE ONE WE WANT?
B71C 23                 INX     H       ;BUMP BYTE POINTER
B71D C206B7             JNZ     LOOP    ;IF NOT OURS, GET ANOTHER BYTE

B720 2B         ADDR    DCX     H       ;BACK UP POINTER
B721 2B                 DCX     H       ;AND ONCE MORE

B722 EB         FAIL    XCHG            ;MOVE POINTER TO D-E
B723 C329B7             JMP     ADDR2   ;SKIP ADDR1 FIRST TIME

B726 53         ADDR1   MOV     D,E     ;SET UP SECOND ADDRESS BYTE
B727 0E01               MVI     C,1     ;COUNT ADDRESS BYTES

B729 7A         ADDR2   MOV     A,D     ;GET ADDRESS BYTE
B72A E6F0               ANI     0F0H    ;CLEAR LOWEST 4 BITS
B72C 0F                 RRC             ;MOVE 4 MOST SIGNIFICANT
B72D 0F                 RRC             ;  ADDRESS BITS DOWN TO
B72E 0F                 RRC             ;  4 LEAST SIGNIFICANT
B72F 0F                 RRC             ;  ADDRESS BITS
B730 CD4AB7             CALL    PRINT   ;PUT ASCII CHAR ON SCREEN
B733 7A                 MOV     A,D     ;GET ADDRESS BYTE AGAIN
B734 E60F               ANI     0FH     ;CLEAR MOST SIG 4 BITS
B736 CD4AB7             CALL    PRINT   ;PUT ASCII CHAR ON SCREEN
B739 0D                 DCR     C       ;SECOND BYTE?
B73A C226B7             JNZ     ADDR1   ;IF NOT GO GET IT'

B73D 3E0D       EXIT    MVI     A,0DH   ;LOAD CARRIAGE RETURN
B73F CD03F8             CALL    SCREEN  ;AND PUT TO SCREEN
B742 3E0A               MVI     A,0AH   ;LOAD LINE FEED
B744 CD03F8             CALL    SCREEN  ; AND PUT TO SCREEN
B747 C30000             JMP     BOOT    ;JUMP TO MONITOR

B74A 215BB7     PRINT   LXI     H,TABLE ;POINT TO TABLE
B74D BE         PRINT1  CMP     M       ;NUMBERS COMPARE?
B74E 23                 INX     H       ;BUMP TABLE POINTER
B74F C257B7             JNZ     PRINT2  ;IF NOT, SKIP SCREEN
B752 7E                 MOV     A,M     ;IF SO, GET ASCII EQUIV.
B753 CD03F8             CALL    SCREEN  ;PUT ON SCREEN
B756 C9                 RET             ;RETURN FOR ANOTHER
B757 23         PRINT2  INX     H       ;BUMP TABLE POINTER 2ND TIME
B758 C34DB7             JMP     PRINT1  ;RETURN FOR ANOTHER

;* TABLE CONVERTS FROM A 4 BIT HEX NUMERAL TO ITS
;* ASCII EQUIVALENT FOR OUTPUT TO THE SCREEN

B75B 00300131   TABLE   DB      0,30H,1,31H
B75F 02320333           DB      2,32H,3,33H
B763 04340535           DB      4,34H,5,35H
B767 06360737           DB      6,36H,7,37H
B76B 08380939           DB      8,38H,9,39H
B76F 0A410B42           DB      0AH,41H,0BH,42H
B773 0C430D44           DB      0CH,43H,0DH,44H
B777 0E450F46           DB      0EH,45H,0FH,46H

B77B                    END
```

RSL II's MEMORY MAP

Dear Sirs,

I was looking through my Radio Shack L II's memory map the other day, and poking and peeking around when I ran across some interesting information that might be of interest to your readers.

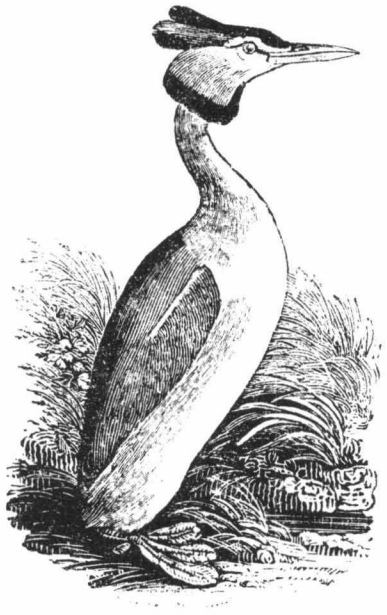
Every key on the keyboard is available for direct access by the user. That means EVERY key, including the four arrows and clear key. By typing in the program below, programmers can get some ideas on how to put this to use. Probably the first and foremost use would be in games, where, instead of using 'inkey$' and 'U' or 'D' for up and down, he may directly use the arrows. The interactive clear key could be used the same way Level I uses it, by clearing the screen while the program is executing if there is a need to do so.

```
10 cls
20 if peek(14400)= 8  then print "up   ";
30 if peek(14400)= 16 then print "down  ";
40 if peek(14400)= 32 then print "left  ";
50 if peek(14400)= 64 then print "right ";
60 goto 10
```

After running the above, start depressing the four arrow keys and see how smooth the operation is to use.

Thank you,
Greg Perry

6104 E. 48th St.
Tulsa, Okla. 74135



DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 37 | August 1979 | Volume 4, Issue 7

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

On Computer Crime Bill S240 4
Jon K. Taber

Diskette Copying on Intel Single Density DOS 28
Bill Collis

Hardware Relative Addressing for the 8080 30
John Beetem

A Floating Point Subroutine Package for the 1802 17
Paul Wasserman

Context Switch for the 6800 27
M. Mickelsen and N. Worth

Those All-Important Extras 20
Don Colburn

UCSD PASCAL to CP/M File Transfer Program 12
Ronald G. Parsons

And—as always—much, much more ...

On Computer Crime Bill S240

BY JON K. TABER
609 Meadow Ave.
Santa Clara, Calif 95051

INTRODUCTION

I wish to bring to the attention of interested readers information about Senate Bill S240, now pending in Congress. This bill attempts to define and outlaw "computer" crimes. It is sponsored by Senatore Ribicoff (D-Conn.) The bill was formerly known as S1766 (see *DDJ* #28.)

Senate Bill S240 consists of a preamble and three sections. The preamble claims that computer crime is a growing, serious problem difficult to prosecute under existing law, thus necessitating the bill. Section (a) attempts to outlaw any use of a computer for fraud; section (b) outlaws "intentional, unauthorized" use, access, or alteration of a computer, computer programs or data. Section (c) consists of definitions of terms. The bill covers government computers, computers for private entities with government contracts, computers used in banking and finance, and computers used by "any entity . . . affecting interstate commerce." Penalty for violation of section (a) is 15 years and/or fine of 2½ times the amount stolen; for section (b) 15 years or $50,000 fine, or both.

The bill is dangerously vague and broad; it has potential for serious abuse; and it is largely unnecessary. It should be opposed.

COMPUTER CRIME: FACTS AND MYTHS

The rationale presented by the bill's preamble is not established. There is no evidence that computer-based fraud and embezzlement are increasing. In fact, the loss from computer crime is insignificant. I say this being well aware of sensational accounts of computer crimes in the mass media. Simply put, mass media accounts are grossly exaggerated or just myths. For example, newspapers widely reported the Security Pacific Bank theft in Los Angeles as a "computer" crime; in fact, it was not. The Pennsylvania RR boxcar thefts were reported as a "computer" crime; again, in fact, the thefts were not. The Equity Funding fraud, one of the largest ever, was also reported as a "computer" crime; this is very much disputed.

From information supplied to the Government Accounting Office by many government investigative agencies, the GAO found 69 cases of computer crime in the entire Federal Government. In reality there were only 66 cases; the Air Force had erroneously identified three cases as computer crimes which did not involve computers in any way. Nine cases involved no dollar loss; they were incidents such as privacy invasion. The total loss was $2,161,413. The average loss was $44,000, and the median, or typical, loss was $6749.

In a study conducted by Stanford Research Institute, and based largely on newspaper articles, the dollar loss for 1975, the latest year for which figures are available, was given as $1.45 million. This includes the private sector as well as local and national government. The total accumulated loss for the past 15 years was given as $280 million (as of 1978). The average was $450,000 per case. Estimated annual losses, made by SRI and others, vary wildly; $100 million, $300 million, $160 million by 1985. *In contrast, the estimated loss due to white collar crime in 1974 was $40 billion.*

These figures are questionable; there is something especially wrong with the average of $450,000, more than ten times the GAO average. The difference is probably due to the fact that it is based on amounts quoted in newspaper articles. Also, the actual loss for 1975 of $1.45 million conflicts with the accumulative loss of $280 million, which is 200 times greater —an inexplicable spread in only fifteen years. It implies that either computer crime is rapidly vanishing, or 1975–1978 were incredible years for computer crime, or 1975 was incredibly low. But even if these figures are blindly accepted, the fact remains that, compared to white collar crime losses, computer crime losses are insignificant.

Furthermore, the incidence rate is insignificant. In 1975 there were 381 known incidents of computer "abuse" worldwide since the advent of computers. Of these cases, 77 are verified, 218 cases have been assigned varying "levels of confidence" (i.e., credibility weights as actually having occurred), and 86 are unverified. Some are suspected to be fictitious.

This is an important point; many well known cases of computer crime which have become part of our folklore have never occurred; they are totally mythical, even if computer professionals widely believe and cite them.

Even verification has dangers; Parker reports that two cases which were verified actually turned out to be fictitious. In other words, that figure of 381 "abuses," as small as it is, could well be much smaller, and must be treated with circumspection. But even accepting this figure tentatively, we are led to an "abuse" incidence rate of one case per year per 2000 computers. This is an insignificant rate.

These 381 cases are not all crimes—they include a large number of questionable uses of computers and odd incidents. Some don't really involve computers at all; for example, the stealing of check forms for forgery from the computer printer area. Of these cases, 145 involved fraud or theft (including Equity Funding; 144 otherwise). A surprising 66 involved physical assaults on the computer, including four cases of the computer being shot, and one of a woman in France who beat up her CRT terminal with high-heeled shoes.

Many cases are not crimes but are, perhaps, unethical. A good example is that of the instructor who used the school computer to print 50 copies of campaign material in a school election. Many others are student shenanigans with school computers; most of the time these do not involve criminal motives.

Because of loose definitions and somewhat arbitrary classifications ("stealing" a password, for example, is classed as theft) it is difficult to determine the number of real crimes out of the 381 "abuse" cases; it is about 210. They are false entry of records, fraud and embezzlement, theft, including theft of computer programs and theft of records, vandalism and sabotage. These are crimes already adequately covered by existing state and federal laws. There are over 40 applicable laws at the federal level alone. S240 is simply unnecessary.

There are other "estimates" that should be mentioned. One is that only one fifth of detected "computer" crimes are reported to the authorities from fear of embarrassment. There is no evidence to support this contention. Furthermore, federal regulations require financial institutions to report all crimes. It is unlikely that they don't, unless there is a massive breakdown in enforcement of banking regulations. If there is, a law like S240 will hardly cure the problem. Another estimate is that only 1/100 of computer crimes is ever discovered. There is not one shred of evidence to support that figure.

It may be argued that the use of computers for fraud creates a unique sort of crime requiring its own criminal law. I doubt this; the use of the computer in fraud is equivalent to the use of the office adding machine. There is an ancient principle which holds that the law is concerned with serious matters, not trivia. The appropriate sanction for cases such as the instructor who misused the school computer is reprimand from his employer, or even dismissal, depending on the gravity of the case, but certainly not 15 years in prison! The system programmer who, without authorization, plays tic-tac-toe on a computer should be beneath the notice of the law.

It is also argued that computer crimes are difficult to prosecute. To the contrary, convictions have been easy to obtain, sometimes where the prosecution was an unwarranted intrusion of federal power into areas of state sovereignty. In the Kelly and Palmer case in Philadelphia, Kelly and Palmer had used their employer's computer without permission in their private music business. The federal prosecutor tried them for mail fraud for advertising their music. A conviction was obtained but the result of appeal is unknown to me. This is a clear case of unwarranted federal intrusion into a state matter.

In the Jones case, the U.S. Attorney complained at the hearings that the judge would not allow a charge of forgery, which he attributed to the legal complexities caused by computers. The judge's ruling, he said, implied that the computer committed the forgery; thus Jones could not be charged with the crime. More likely, however, the legal complexities were caused by the fact that the forgery occurred in Canada where the U.S. prosecutor does not have jurisdiction. It would seem that the U.S. Attorney blamed the computer for his own bungling of the indictment. Jones was convicted under the correct charges allowed by the court. In every known case in which real crime occurred, prosecutors have been able to secure convictions under existing law.

PRISON PROGRAMMING

The Department of Justice, and other proponents, contend that computer crime is easy to commit and difficult to detect. For inexplicable reasons they seem to regard programmers with suspicion and some hostility. The FBI is afraid of "computer freaks," and a *Time* magazine article on computer crime, whose obvious source was the Department of Justice, concluded with "Ideally, the first step in securing a system would be to shoot the programmer." This hostility is impossible to understand; studies of computer crime, as flawed as they are, agree that they are rarely perpetrated by the programmer (it is usually the data entry clerk or the managers).

Apparently, this fear and hostility doesn't apply if the programmers are armed robbers, murderers, and forgerers. The Justice Department's Bureau of Prisons runs a small but burgeoning data processing service employing convicts in at least six Federal prisons. Customers include the Departments of Defense, Agriculture, the Internal Revenue Service, the Bureau of Prisons, and the National Endowment for the Humanities. The Department of Agriculture has even located its new computer center in Kansas City, Missouri, just to be close to the convict programmers incarcerated in Leavenworth. Gross earnings in fiscal year 1976 were just shy of one million dollars, and must be well over that today. Federal Prison Industries Inc. claimed net profits of about 15 percent for contract data processing services. Data entry services are provided for the Navy's supply system by female offenders in Alderson, W. Va. and Terminal Island, California. Serious COBOL programming is provided by Leavenworth inmates for the Agriculture Department's Stabilization and Conservation Service. These programs form part of the general ledger and accounting programs of the Department of Agriculture, and affect the disbursement of funds. The same Leavenworth convicts have reportedly written unspecified programs for the IRS. Indeed, there were rumors at one time, apparently unfounded, that the convicts learned enough about the IRS computerized tax return system so that they were filing fraudulent returns which escaped detection by the IRS computers.

I stress, however, that these rumors appear unfounded. Furthermore, future prison business can only grow if enough prisoners are found who are willing and able to program. The GSA, no doubt at the urging of the Department of Justice, promulgated Federal Procurement Regulation 1-5.402 on December 2, 1974. This regulation requires all Federal agencies to give priority to the Federal Prison Industries, Inc. over private industry for all data entry and programming services. The agencies are required to pay the going commercial rates with perhaps an incentive deduction.

There is something clearly insane here. On the one hand, the Department of Justice asks for a broad, dangerous law because it says computer crime is so easy to commit and so

difficult to detect. On the other hand, the Department of Justice sees nothing wrong with prisoners convicted of serious crimes programming accounting applications, and indeed the Department even attempts to increase such activities through federal regulation. The insanity is called schizophrenia.

There is nothing wrong with prisoners writing programs. Computer crime is *not* easy to commit: the Bureau of Prisons is living proof. It might be the one good job training program in the entire prison system. Qualified convicts learn a useful trade, unlike the usual prison job training, and are hired upon release into well-paid jobs. As one would expect, recidivism is extremely low.

The point to be made is that the Department of Justice itself gives the lie to contentions in favor of the Ribicoff bill.

I would suggest that FPR 1-5.402 be rescinded to avoid the possibility of abuse. By the admission of the authorities, many prison programmers are worked over 14 hours a day. Growing business may tempt prison staff to coerce unwilling or unsuited prisoners into programming. Also, it may be wise to redirect programming away from high priority work to low priority work that normally would not get done due to lack of assignable personnel. This would remove any temptation that might exist on the part of government agencies to exploit what amounts to slave labor while still keeping a worthy program going.

OUTLAWS COMMON PRACTICES

Section (b) of the bill is too broad. It doesn't attempt to distinguish felonious from misdemeanor misuses of computers. Nor does it distinguish ethically questionable uses. I will address the "unauthorized" use, which is neither felony nor misdemeanor.

"Unauthorized" use of a computer is widespread among programmers. Programmers on occasion use computers to play games like tic-tac-toe or Star Wars. They draw tabby cat calendars and pinup girls. They have discovered certain combinations of nonsense characters on the 1403 line printer which generate musical tones. I know a programmer who played "She'll Be Comin' Round the Mountain" on the printer, using judiciously-timed page ejects to create a drum beat accompaniment.

Programmer ingenuity is amazing. Computers are used to balance checkbooks, chart stocks, and figure mortgage tables. Programmers write "unauthorized" programs that have little earthly use, like the knight's tour of the chessboard, or a base-256 multiplier. Sometimes, an "unauthorized" program proves useful: traveling the programmers' grapevine, it becomes adopted at computer centers all over the world. This play and incidental personal use is without pecuniary motives, but all of it is a serious crime under section (b).

There may be some who would argue that such play should be forbidden, that the computer is not a toy and such games steal time and resources from their rightful owners. This argument has merit: I am not trying to condone unethical behavior with the excuse that everybody does it. But Senate Bill S240 is a radical "solution" to this problem – if it is a problem. Imprisonment for 15 years is out of proportion to the offense. The bill constitutes improper intrusion into an area where employer sanction is adequate and proper.

Also, employer attitudes vary widely. Some flatly forbid non-business uses of their computers, even policing machine usage to enforce the ban. Some forbid it in theory but allow it in practice. Others allow it as a sort of fringe benefit. For many companies, the question has never occured. Thus, an act committed on one computer would be perfectly legal but if committed on another computer it would result in a programmer's imprisonment. This bill cannot be equitably enforced.

In any case, great care should be exercised in forbidding such uses. Many "games" provide insight into computer programming and are of professional benefit. Eliza is a game which is a classic in artificial intelligence research. I suspect that in many areas of science, mathematics, and computer science it will be impossible to distinguish between "unauthorized" use and research. I suspect that researchers will find this bill intolerable.

The bill does not address the problem of who may authorize use of the computer. Presumably, the authorizer is the employer, which leads to the absurdity that the Equity Fund fraud would not violate section (b) because the company officers authorized the use of the computer in perpetrating the fraud. The question of who may authorize what is not an idle one. Commonly, "employers" are "users" and do not own either the hardware or the software; these are rented, and the owner does not relinquish rights of ownership by renting them to the user. Yet it is common practice for the user to alter rented code (and even hardware) without authorization from the rightful owner. In fact, the owners generally prefer the user not modify software because of the maintenance problems it creates. Thus, section (b), which makes unauthorized alteration of computer programs a felony, will force wholesale renegotiation of contracts. There is no good reason to change this common industry-wide practice.

MAKES FELONIES OF ABSURDITIES

The definition of "computer" is too loose; "computer means an electronic device which performs logical, arithmetic, and memory functions by the manipulations of electronic or magnetic impulses.." This definition would include, and is apparently meant to include, pocket calculators and even some digital watches. The fact is that microprocessors enjoy wide application today in all sorts of gadgets. Thus, under this bill, a secretary who uses the office word processor to type a personal letter, and the clerk who plays the "shell oil" calculator joke on his office calculator, are felons who may be imprisoned for 15 years. I doubt very much that this definitional problem can ever be overcome. We in the industry cannot agree on a definition of computer for technical purposes, let alone legal.

MAY BE ABUSED

I most strongly protest that this bill has dangerous potential for abuse. First, the bill is a serious threat to privacy. Second, there is an obvious danger of jailing programmers (and all other computer users) for bagatelles.

The reader must realize that record keeping is the most important use of computers today from a social point of view. Most record keeping is computerized, and virtually all records soon will be. This bill, because it is so broad, gives the FBI legal access to most records, under their investigative powers. They have never had this access before. This point was made by Senator Biden at the hearings on this bill:

"I know that there is a good deal of criticism and concern about abuse of power (by the FBI, the CIA, and our security industry)... We are going to be turning to these agencies and saying, 'We are going to broaden your jurisdiction now. We are going to allow you legally to get into a number of data banks that you did not have access to before'.... Your legislation is very broad. As I read it, just about any computer in America will be accessible for the first time to investigation by a major Federal law enforcement agency."

Precisely. The point is undeniable, and Senator Percy, to whom Biden was speaking, did not deny it. Senator Percy allowed that S240 was not a panacea, and when Biden pressed the point, Percy expressed pious hopes that a privacy bill, "vitally needed", (but not enacted yet) would help prevent abuse. Nor did Senator Ribicoff deny it, nor the Justice Department which had requested the bill in the first place, nor the FBI spokesman.

I sincerely hope the Congress will reject the bill on this point alone.

The second potential abuse is the arbitrary jailing of programmers for bagatelles. As has been pointed out, "unauthorized" use of computers is widespread. The bill will not change this fact. Most will be unaware of the law; they do not feel that their personal use of computers is wrong, so long as it is not for material gain. Even if some become aware of the law, they will simply disbelieve that it applies to their "unauthorized" use of computers.

Generally speaking, computer professionals are technically oriented and do not know or care how laws and sausages are made. The result will be that most programmers (and many other computer users) will be unprosecuted felons, vulnerable to abuse. This is too much discretionary power to give law enforcement. While some discretion is necessary for effective law enforcement, modern American history does not make happy reading in the enforcement of broad laws. The FBI *has* abused its powers; I cite its abuses of the Mann Act and the Dyer Act. It *has* acted illegally in searches and seizures; why legalize it? Prosecutors have, out of sheer muddle-headedness, not to mention ambition, jailed people who should never have been charged.

But, of course, will this specific law be abused? The Department of Justice said it would prosecute only cases in which the Federal Government has a compelling interest. This is cold comfort. It should be noted that nowhere does either the FBI or the Department of Justice expicitly promise not to prosecute the programmer who plays tic-tac-toe or draws a calendar, even under direct questioning on the point:

BIDEN: Let us level with the public. Let us acknowledge to them, by implication at least, that we are not going to prosecute that particular person...

FINNEY (Department of Justice): The Snoopy (calendar) was our case.

BIDEN: Yes. Acknowledge that we are not going to prosecute Snoopy and do not leave the possibility of abuse.

Finney responds then for several pages, mentioning the good sense of the FBI and the Department of Justice, and claiming that trust is needed, but he does not give the acknowledgement asked for.

Will computer users be jailed for bagatelles? Computer crime is, on the whole, record keeping crimes. But it is a new, glamorous idea, computer crime, currently sensationalized by the media. Dick Tracy is fighting computer crime these days. *Penthouse* is planning an article on it. Computer crime suffers great publicity creating the impression of a widespread problem, and generates pressure for the prosecution of computer criminals. However, there are scarcely any. The prosecutor, trained for and assigned to computer crime, will have to be content with the games-playing programmer if he is ambitious for his career.

The reader may think it unlikely that a judge would permit prosecution of the tic-tac-toe player. Unfortunately, the reader is mistaken; the judge will have little choice. Senator Ribicoff, on reintroducing the bill as S240, said in the *Congressional Record* that this type of "playing around" is the same type of activity that enables computer crime. He cited the specific example of a Department of Agriculture employee at the Washington Computer Center who permitted his children to play computer games on the WWC computer. There is scarcely a programmer in the country who hasn't done the same. But the same employee had also used the WWC computer for his own outside consulting business. The drift of Senator Ribicoff's contention is that the main purpose of the bill is not to jail programmers for "playing around"; but to change the language of the bill to accommodate "playing around" would seriously weaken the bill. Therefore, if a programmer falls afoul of the language, too bad. He shouldn't be "playing around" anyway.

This is more than a Senator's mistake. Trial judges, when attempting to apply a new law in a doubtful case, consult the legislative record to divine the legislator's intent. Unless there are constitutional grounds, the judge is required to respect that intent, and I remind the reader that the Constitution does not forbid bad laws, only certain bad laws. Senator Ribicoff was clearly instructing trial judges as to his intent. Thus if this bill becomes law, the judge must permit the trial, which will be limited to the factual determination of whether or not the programmer intentionally played tic-tac-toe without authorization. We hope the programmer will not receive a severe sentence, but the judge may be limited. Senate Bill S1437, the successor to SB1, may become law soon. This bill limits the judge's sentencing discretion to a formula of plus or minus 25 percent of the nominal sentence. The programmer must receive a minimum sentence of 11 years and may receive as much as 19.

For some time now, the FBI has been training prosecutors and law enforcement personnel on computer crime. The FBI conducts both a one week course and a four week course at Quantico, VA. The one week course has 500 graduates. Mr. Henehan of the FBI said in testimony that "there is a reluctance on the part of both the prosecutors and the investigators to get into these cases (that is, computer crime). We find that, through training, they are much more anxious to accept a case." It seems that this course generates enthusiasm for prosecuting computer crimes. Also, the FBI thinks it will need 200 more special agents, 45 more accountant technicians, and 10 auditor-computer specialists. What this means is that a lot of people are being geared up to prosecuting and investigating computer crimes, a glamorous new area.

Thus, I think the chances for abuse are enormous. Our only protection so far is the FBI's and the Justice Dept's promises that we can trust them.

CONCLUSION

Senate Bill S240 is an ill-formed and dangerous law that must be rejected. The bill is fundamentally flawed. The fundamental problem is that it defines an abstraction, "computer crime," as a crime rather than specific acts. Compare "computer crime" with "filing cabinet crime" to make this flaw apparent. Computer crime (or filing cabinet crime) beclouds specific criminal acts, and non-criminal acts, with a trope drawn from the instrument of the acts. It is true that one may commit murder with a filing cabinet by dropping it from sufficient height on a victim, and one may tamper with records using a computer to perpetrate a fraud. Nevertheless, the crime is murder, not unauthorized use of a filing cabinet, and fraud, not unauthorized access of a computer.

Criminal law, as I understand it, is pragmatic; its concern is concrete acts that have active, stark verbs to describe them: steal, murder, defraud, vandalize. There is a corresponding stark noun to name the doer of the active verb; thief, murderer, defalcator, vandal. Let us not now create out of latinate English the adverbial crime, "access without authorization," of which the sense at least is passively voiced. It lacks a clear noun to name the doer.

REFERENCES

Senate Committee on Government Operations. *Staff Study of Computer Security in Federal Programs.* Committee Print. 95th Congress, 1st Session, Feb. 1977.

Senate Committee on Government Operations. *Problems Associated with Computer Technology in Federal Programs and Private Industry. Computer Abuses.* Committee Print. 94th Congress, 2nd Session, June 1976.

U. S. Senate. *Hearings before the Subcommittee on Criminal Laws and Procedures of the Committee on the Judiciary. Federal Computer Systems Protection Act.* (S1766). 95th Congress, 2nd Session, June 21 and 22, 1978.

Parker, Donn B.; Nycum, Susan; Oura, Steven. *Computer Abuse.* Stanford Research Institute Report (NTIS #PB 231-320/AS). Nov 1973.

Parker, Donn B. *Computer Abuse Assessment.* Stanford Research Institute Report, NSF Grant MCS 76-09183. Dec 1975.

Parker, Donn B. *Computer Abuse Perpetrators and Vulnerabilities of Computer Systems.* Stanford Research Institute Report, NSF Grant MCS 76-09183. Dec 1975.

Parker, Donn B. *Crime by Computer.* Charles Scribner's Sons. New York. 1976.

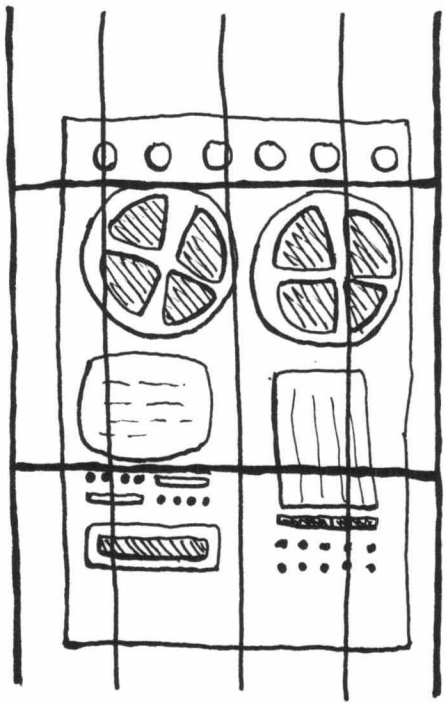
Loeffler, R. M. *Report of the Trustee of Equity Funding Corp of America.* Oct 31, 1974.

"Introduction of S240 by Mr. Ribicoff", *Congressional Record.* Vol 125, No. 7. Jan 25, 1979. pp S709-S726.

U. S. Chamber of Commerce. *A Handbook on White Collar Crime.* 1974.

Federal Prison Industries, Inc. *Annual Report,* FY 1976.

"Computer Capers", *Time,* Aug 8, 1977; p 53.



S 240

96th CONGRESS
1ST SESSION

To amend title 18, United States Code, to make a crime the use, for fraudulent or other illegal purposes, of any computer owned or operated by the United States, certain financial institutions, and entities affecting interstate commerce.

IN THE SENATE OF THE UNITED STATES
JANUARY 25 (legislative day, JANUARY 15), 1979

Mr. RIBICOFF (for himself, Mr. PERCY, Mr. KENNEDY, Mr. INOUYE, Mr. JACKSON, Mr. MATSUNAGA, Mr. MOYNIHAN, Mr. WILLIAMS, Mr. ZORINSKY, Mr. DOMENICI, Mr. STEVENS, Mr. CHILES, and Mr. NUNN) introduced the following bill; which was read twice and referred to the Committee on the Judiciary

A BILL

To amend title 18, United States Code, to make a crime the use, for fraudulent or other illegal purposes, of any computer owned or operated by the United States, certain financial institutions, and entities affecting interstate commerce.

Be it enacted by the Senate and House of Representatives of the United States of America in Congress assembled,

That this Act may be cited as the "Federal Computer Systems Protection Act of 1979."

Sec. 2. The Congress finds that–

(1) computer-related crime is a growing problem in the Government and in the private sector;

(2) such crime occurs at great cost to the public since losses for each incident of computer crime tend to be far greater than the losses associated with each incicent of other white collar crime;

(3) the opportunities for computer-related crimes in Federal programs, in financial institutions, and in other entities which operate in interstate commerce through the introduction of fraudulent records into a computer system, unauthorized use of computer facilities, alteration or destruction of computerized information files, and stealing of financial instruments, data, or other assets, are great;

(4) computer-related crime directed at institutions operating in interstate commerce has a direct effect on interstate commerce; and

(5) the prosecution of persons engaged in computer-related crime is difficult under current Federal criminal statutes.

Sec. 3. (a) Chapter 47 of title 18, United States Code, is amended by adding at the end thereof the following new section:

§ 1028. Computer fraud and abuse

"(a) Whoever knowingly and willfully, directly or indirectly accesses, causes to be accessed or attempts to access any computer, computer system, computer network, or any part thereof which, in whole or in part, operates in interstate commerce or is owned by, under contract to, or in conjunction with, any financial institution, the United States Government or any branch, department, or agency thereof, or any entity operating in or affecting interstate commerce, for the purpose of–

"(1) devising or executing any scheme or artifice to defraud, or

"(2) obtaining money, property, or services, for themselves or another, by means of false or fraudulent pretenses, representations or promises, shall be fined a sum not more than two and one-half times the amount of the fraud or theft, or imprisoned not not more than fifteen years, or both.

"(b) Whoever intentionally and without authorization, directly or indirectly accesses, alters, damages, destroys, or attempts to damage or destroy any computer, computer system, or computer network described in subsection (a), or any computer software, program or data contained in such computer, computer system or computer network, shall be fined not more than $50,000 or imprisoned not more than fifteen years, or both.

"(c) For purposes of this section, the term–

"(1) 'access' means to approach, instruct, communicate with, store data in, retrieve data from, or otherwise make use of any resources of, a computer, computer system, or computer network;

"(2) 'computer' means an electronic device which performs logical, arithmetic, and memory functions by the manipulations of electronic or magnetic impulses, and includes all input, output, processing, storage, software, or communication facilities which are connected or related to such a device in a system or network;

"(3) 'computer system' means a set of related, connected or unconnected, computer equipment, devices, and software;

"(4) 'computer network' means the interconnection of communication systems with a computer through remote terminals, or a complex consisting of two or more interconnected computers;

"(5) 'property' includes, but is not limited to, financial instruments, information, including electronically processed or produced data, and computer software and programs in either machine or human readable form, and any other tangible or intangible item of value;

"(6) 'services' includes, but is not limited to, computer time, data processing, and storage functions;

"(7) 'financial instrument' means any check, draft, money order, certificate of deposit, letter of credit, bill of exchange, credit card, or marketable security, or any electronic data processing representation thereof;

"(8) 'computer program' means an instruction or statement or a series of instructions or statements, in a form acceptable to a computer, which permits the functioning of a computer system in a manner designed to provide appropriate products from such computer system;

"(9) 'computer software' means a set of computer programs, procedures, and associated documentation concerned with the operation of a computer system;

"(10) 'financial institution' means –

"(A) a bank with deposits insured by the Federal Deposit Insurance Corporation;

"(B) a member of the Federal Reserve including any Federal Reserve bank;

"(C) an institution with accounts insured by the Federal Savings and Loan Insurance Corporation;

"(D) a credit union with accounts insured by the National Credit Union Administration;

"(E) a member of the Federal home loan bank systems and any home loan bank;

"(F) a member or business insured by the Securities Investor Protection Corporation; and

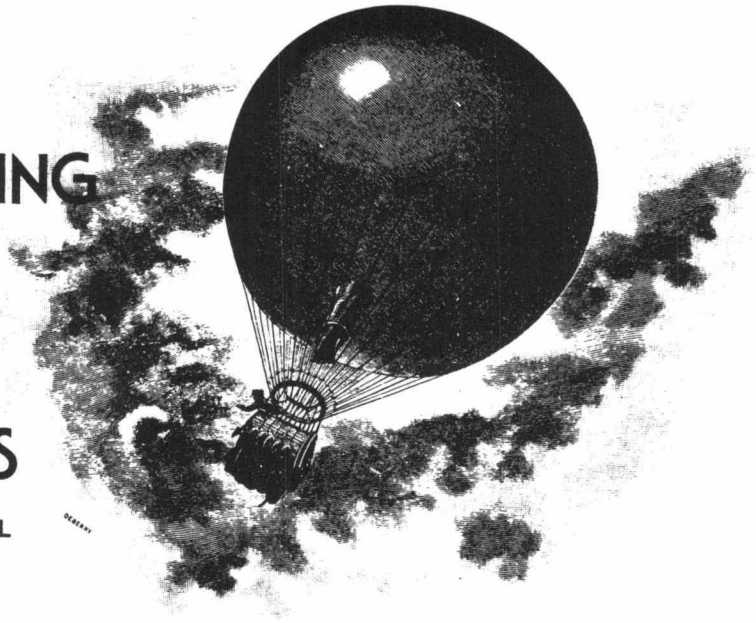
"(G) a broker-dealer registered with the Securities and Exchange Commission pursuant to section 15 of the Securities and Exchange Act of 1934."

(c) The table of sections of chapter 47 of title 18, United States Code, is amended by adding at the end thereof the following:

"1028. Computer fraud and abuse.".

PROGRAMMING PASTIMES AND PLEASURES

BY CHARLES WETHERELL



What's An Ackermann and How Does It Function?

In recent issues of *Dr. Dobb's*, Ackermann's function has been discussed by several authors. None of them have explained why Ackermann's function is important and why we should be interested in it. I am going to remedy that lack by reviewing the history and theory behind the function.

Theoretical Computers

When I want to prove a mathematical theorem about computers and their powers, I will probably use a simplified model of a computer. Partly this is the mathematician's trick of ignoring inconvenient parts of the real world so that a proof is possible; mostly, though, real computers are far too messy to be tractable in proofs. So a simple model eliminates all the complicated, special purpose instructions and concentrates on the essentials. This strategy succeeds because no real power is lost even when almost all instructions are eliminated from most computers; for example, the PDP-8 has only eight instructions to begin with.

Naturally enough, theoreticians have proposed a variety of computer models, among them Turing machines, partial recursive functions, Markov algorithms, RASP's, RAM's, the lambda calculus, type 0 grammars, Post correspondence problems, flowchart schemata, and many more (a computer science graduate student can probably go on for days talking about this list). Some of these models look a lot like ordinary computers; some are almost unrecognizable as computers. But they all share one property summarized by this rough version of Church's Thesis:

> Any *sufficiently* powerful computing mechanism can compute exactly the same functions as any traditional computer and hence all *sufficiently* powerful mechanisms compute exactly the same set of functions.

Church's Thesis does not mean that all mechanisms take the same time or space for the same function or that the programs have to look the same – only that there is *some* way for any mechanism to compute any function your computer can compute. Church's Thesis is very reassuring; if I can prove my model (or computer) "sufficiently" powerful, then it is as good as your giant supercomputer, even if perhaps a trifle slower.

What Is Sufficiently Powerful?

The question arises: "What is sufficiently powerful?" One definition is that a mechanism is sufficiently powerful if it can compute anything an ordinary computer can. This definition is circular and hence unappealing, although not completely frivolous. Taking another view, a computing mechanism is sufficiently powerful if it can

- define the constant zero,
- define the successor (*i.e.*, add one) function,
- compare two values for equality,
- jump on the basis of such a comparison,
- define new functions in terms of old ones using some kind of recursive subroutine mechanism, and
- access potentially infinite storage (*e.g.*, the operator will provide all the floppy disks the system desires).

This definition is accurate enough, but is hard to apply. So mathematicians imagine very simple computer models and test to see how powerful they are. By doing enough of these "thought" experiments, they hope to understand exactly where the line between sufficiently powerful and not-good-enough lies.

The partial recursive functions were the first carefully studied model, developed by Godel perhaps 20 years before digital computers were available. A simpler model, called the primitive recursive functions, was used as one of the first "thought" experiments. An integer function is primitive recursive if it is

a. a constant – say 3, 5, 97, or 0, or
b. an identity function – say f (x)=x or g (x,y,z)=y, or
c. the successor function s (x)=x+1, or
d. built from already known primitive recursive functions by composition – g (x)=s (s(x))=(x+1)+1=x+2 for example, or
e. built from already known primitive recursive functions using the mechanism of primitive recursion – that is, if we know g and h as functions already, then we can define f by

f (0, y, z) = g (y, z)
f (x+1, y, z) = h (x, f (x, y, z), y, z)

Primitive recursion is illustrated for three arguments; f can have any number of arguments. If the first argument is zero, apply the first rule to compute a value; otherwise, apply the second rule. Notice that primitive recursion is a lot like iteration (DO-loop) combined with a recursive subroutine call.

Using these ideas, I can easily define a primitive recursive function PLUS to do addition:

PLUS (0, y) = y (using the identity function)
PLUS (x+1, y) = s (PLUS (x, y)) using the successor function

If you fiddle around with primitive recursive functions enough, you will find that all the functions you ever want to compute – square root, exponential, Bessel functions, sine, cosine, factorial, all of them – are primitive recursive, even though some of the programs may be quite long.* So are primitive recursive functions sufficiently powerful.

Ackermann's Function

NO, primitive recursive functions are not powerful enough, all because of Ackermann's function. To understand this, consider the function a(x,y) which is defined as applying the successor function to y x times, or

$$a(x,y) = y + 1 + 1 + 1 \cdots + 1 \text{ (with x 1's).}$$

Clearly a(x,y) is just x+y. Now consider the function b(x,y) which is defined as applying the function a to y x times, or

$$b(x,y) = y + y + y + y \cdots + y \text{ (with x y's).}$$

If you think about it, you will see that b(x,y) is x • y or multiplication. Similarly, let c(x,y) be b applied to y x times, or

$$c(x,y) = yyyy \cdots y \text{ (with x y's).}$$

Then c(x,y) is y^x. Keep going with d(x,y) and you get

$$d(x,y) = y^{y^{y^{\cdot^{\cdot^{\cdot^{y}}}}}} \text{ (with x y's).}$$

You might try to write out what e(x,y), f(x,y), g(x,y), . . . , are. You will see that the process can go on forever. And now I can define Ackermann's function by

A(0, x, y) = a(x, y) = x + y
A(1, x, y) = b(x, y) = xy
A(2, x, y) = c(x, y) = yx
$A(3, x, y) = d(x, y) = y^{y^{y^{\cdot^{\cdot^{\cdot^{y}}}}}}$ (y's)
A(4, x, y) = e(x, y), and so on.

This definition of Ackermann's function is correct but vague. A more precise definition is

A(0, 0, y) = y
A(1, x+1, y) = A(0, x, y) + 1
A(1, 0, y) = 0
A(n+2, 0, y) = 1
A(n+1, x+1, y) = A(n, A(n+1, x, y), y).

To use the definition, work from the top down to find a match with your arguments and then use the (possibly recursive) rule on the right to compute the value. Now, even though every one of the functions a, b, c, d, e, . . . is primitive recursive, there is *no primitive recursive* program for Ackermann's function A(n,x,y). The basic reason is that the first argument makes A grow too quickly for the primitive recursion mechanism to keep up. Since I can write a computer program for A, primitive recursion is not good enough to emulate all possible computer programs.

The proof that A is not primitive recursive is the only reason for the existence of Ackermann's function; there is no other practical use for A(n,x,y). For computer programmers, though, A can be a lot of fun and a good test. To write a completely correct program for A, you must have a full general recursion feature on your computer or in your programming language; no simpler mechanism will do. You also have to have a lot of computer time, because A grows very fast. For example,

$$A(3,3,3) = A(2,A(3,2,3),3) = 3^{A(3,3,2)} = 3^{(3^3)} = 3^{27} = ???$$

Because the previous discussion gives explicit formulas for A(n,x,y) when n is small, you can write your program to cut off the recursion early when n gets small. Also, you can build a table of all previously computed values and never have to recompute any value. I have programmed Ackermann's function several times using the general definition and displayed all the partial computations on a screen; watching the steps go by is a lot of fun. By the way, you may see slight variants on the definition of Ackermann's function; they will all share this property of rapid growth.

If you are interested in more reading, you might be interested in Minsky's *Computation: Finite and Infinite* or the front part of Rogers' *Theory of Recursive Functions and Effective Computability* (after page 21, the going gets pretty heavy).

[1] Primitive recursive functions work only on integers. But if you think about floating point on your computer, you will realize that you are not computing with *real* real numbers, but with elaborately coded integer approximations to real numbers. Primitive recursive functions can be made to do the same encoding, albeit at great expense.

UCSD PASCAL to CP/M file transfer program

BY RONALD G. PARSONS
9001 Laurel Grove Dr.,
Austin, TX 78758

© 1979, Ronald G. Parsons

Many users of the 8080/Z-80 version of UCSD PASCAL also have the CP/M operating system from Digital Research. The file editing facilities of the screen editor of UCSD PASCAL are far superior to that of ED, the CP/M editor. Release I.5 and later versions of PASCAL also contain a Z-80 macro/conditional assembler. The program in the listing allows files to be transferred from a PASCAL diskette to a CP/M diskette. Thus files can be created, edited and assembled with the PASCAL facilities and then run or otherwise used with CP/M.

Two types of PASCAL files are treated specially. Files with the suffix .TEXT have a two block (1024 byte) header. This header is skipped and the remainder of the .TEXT file is converted to CP/M format, i.e., a line feed follows a carriage return and the indent flag of the PASCAL file causes blanks to be inserted. The last block of the file is filled with control-Z's. Files with the suffix .CODE have a one block (512 byte) header. This header is skipped and the remainder of the file is transferred to the CP/M file (probably a CP/M cOM file). To be properly loaded and executed by CP/M, the .CODE file should be an assembled file with .ORG at 100H. Of course, only the 8080 subset of the Z-80 language should be used on an 8080 version of CP/M. PASCAL filenames not ending in .TEXT or .CODE are transferred without modification.

The program runs under CP/M with the syntax

PASTOCPM <CP/M filename> <PASCAL filename>

The PASCAL filename suffix is tested and the CP/M disk is searched to see if a file already exists with the same name. If so, the user is asked whether to abort the transfer or overwrite the old file.

LISTING

```
                ; Copyright (C) 1979 Ronald G. Parsons
                ;
                ; Transfers a Pascal file to a CP/M file.
                ;
                ; Syntax PASTOCPM  <CP/M filename>  <Pascal filename>
                ;
                ; Transfers the Pascal filename on a Pascal disk in B
                ;    to the CP/M filename.  If the CP/M file already exists,
                ;    you will be asked for permission to overwrite.
                ; If Pascal file is a .TEXT file, then two blocks are skipped
                ;    and indents replaced by spaces.  LFs are added after CR.
                ; If Pascal file is a .CODE file, then one block is skipped.
                ; If neither .CODE or .TEXT, copy is exact.
                ;
                ; Pascal disk in B; CP/M disk in A.
                ;
CBFF =          STACK   EQU     0CBFFH          ;change if necessary
BA00 =          CBIOS   EQU     0BA00H          ;point to CBIOS. Change if nec.
0010 =          DLE     EQU     10H
0000 =          CBOOT   EQU     0               ;LOCATION OF CP/M BOOT VECTOR
001A =          DENTSZ  EQU     1AH             ;DIR ENTRY SIZE IN BYTES
0006 =          DTITLE  EQU     06H             ;OFFSET TO ENTRY TITLE
1000 =          BLKBUF  EQU     1000H           ;TOP OF BLOCK BUFFER
2000 =          DIRTOP  EQU     2000H           ;TOP OF TEMP RAM DISK DIRECTORY
                ;
0100                    ORG     100H
0100 31FFCB             LXI     SP,STACK
                ;
0103 218100             LXI     H,BUFF+1        ;get Pascal file name
0106 11E804             LXI     D,SYSTLE+1
0109 0E00               MVI     C,0
010B CD0903             CALL    SCBK            ;get second parameter
010E CDFF02             CALL    SCBKCH
0111 7E         PFN2    MOV     A,M
0112 FE00               CPI     0
0114 CA1E01             JZ      PFN3
0117 12                 STAX    D
0118 23                 INX     H
0119 13                 INX     D
011A 0C                 INR     C
011B C31101             JMP     PFN2
011E 79         PFN3    MOV     A,C             ;get filename length
011F 32E704             STA     SYSTLE
0122 B7                 ORA     A
0123 CAAE03             JZ      NOFLNM
                ; Check for .TEXT or .CODE filename
0126 11A304             LXI     D,TEXT+6
0129 2B                 DCX     H               ;HL points to end of filename
012A E5                 PUSH    H
012B 0E05               MVI     C,5
012D 1A         TEXTLP  LDAX    D
012E BE                 CMP     M
012F C24501             JNZ     NOTEXT          ;filename does not end in .TEXT
0132 2B                 DCX     H
0133 1B                 DCX     D
0134 0D                 DCR     C
0135 C22D01             JNZ     TEXTLP
0138 21D704             LXI     H,TXTFLG        ;got .TEXT file
013B 3602               MVI     M,2
013D 11D904             LXI     D,TEXT
0140 0E09               MVI     C,WRITECB
```

```
0142 CD0500            CALL    BDOS
                NOTEXT
0145 E1                POP     H               ;point to end of filename
0146 11C004            LXI     D,CODE+6
0149 0E05              MVI     C,5
014B 1A         CODELP LDAX    D
014C BE                CMP     M
014D C26301            JNZ     NOTCODE         ;filename does not end in .CODE
0150 2B                DCX     H
0151 1B                DCX     D
0152 0D                DCR     C
0153 C24B01            JNZ     CODELP
0156 21D704            LXI     H,TXTFLG        ;got .CODE file
0159 3601              MVI     M,1
015B 11BA04            LXI     D,CODE
015E 0E09              MVI     C,WRITECB
0160 CD0500            CALL    BDOS
                NOTCODE
0163 115C00            LXI     D,FCB           ;OPEN   FILE
0166 0E0F              MVI     C,OPENF
0168 CD0500            CALL    BDOS
016B FEFF              CPI     ERNXF           ;NON-EXISTENT FILE?
016D CA8F01            JZ      CREF            ;YES - CREATE IT
0170 114904            LXI     D,PERMSG        ;get permission to delete file
0173 0E09              MVI     C,WRITECB
0175 CD0500            CALL    BDOS
0178 =          RDCHR  EQU     $
0178 0E01              MVI     C,READC
017A CD0500            CALL    BDOS            ;READ CONSOLE
017D FE00              CPI     MODE
017F CA0000            JZ      CBOOT           ;ABORT
0182 FE0D              CPI     CR
0184 C27801            JNZ     RDCHR
0187 115C00            LXI     D,FCB
018A 0E13              MVI     C,DELETEF       ;KILL   FILE
018C CD0500            CALL    BDOS
                ;
018F =          CREF   EQU     $
018F 115C00            LXI     D,FCB
0192 0E16              MVI     C,CREATEF       ;CREATE FILE
0194 CD0500            CALL    BDOS
0197 FEFF              CPI     ERNDR           ;create error?
0199 CACA03            JZ      CERROR
                ;
019C 218000     INIT   LXI     H,BUFF
019F 22E304            SHLD    BOL
01A2 21FF00            LXI     H,BUFF+127
01A5 22E504            SHLD    EOB
01A8 210012            LXI     H,BLKBUF+512
01AB 22DD04            SHLD    BUFADD
                ;
01AE 010020            LXI     B,DIRTOP        ;READ THE DIRECTORY INTO DIRTOP
01B1 CD0C02            CALL    READ$DIR
                ;
01B4 210020            LXI     H,DIRTOP        ;SET THE DIRECTORY ENTRY POINTER
01B7 111A00            LXI     D,DENTSZ        ;  TO FIRST ENTRY AFTER THE VOLUME N
01BA 19                DAD     D
01BB 22D804            SHLD    DENTP
                ;
01BE CD1502            CALL    FIND$FILE       ;THEN FIND THE FILE
                ;
01C1 2AD804            LHLD    DENTP           ;start of directory
01C4 5E                MOV     E,M
01C5 23                INX     H
01C6 56                MOV     D,M
01C7 D5                PUSH    D               ;save first block
01C8 23                INX     H
01C9 5E                MOV     E,M
01CA 23                INX     H
01CB 56                MOV     D,M
01CC EB                XCHG                    ;last block + 1 in HL
01CD 29                DAD     H
01CE 29                DAD     H
01CF 22E104            SHLD    LSTLSN          ;last LSN + 1
01D2 E1                POP     H
01D3 3AD704            LDA     TXTFLG
01D6 3D         SKPBL1 DCR     A               ;skip TXTFLG blocks
01D7 FADE01            JM      SKPBL2          ;  depending on .TEXT or .CODE
01DA 23                INX     H
01DB C3D601            JMP     SKPBL1
                SKPBL2
01DE 29                DAD     H
01DF 29                DAD     H
01E0 22DF04            SHLD    LSN             ;first LSN
                ;
01E3 CDC402     LR80B  CALL    R80B            ;read 80 bytes
01E6 3ADA04            LDA     EOFFLAG
01E9 B7                ORA     A               ;eof?
01EA C2F301            JNZ     FILL1A
01ED CDE502            CALL    WB              ;write buffer
01F0 C3E301            JMP     LR80B
                ;
01F3 3AD704     FILL1A LDA     TXTFLG
01F6 FE02              CPI     2               ;.TEXT?
01F8 C20102            JNZ     FILL2
01FB CDA702            CALL    CT              ;fill buffer with 1AH
01FE CDE502            CALL    WB              ;write buffer
0201 115C00     FILL2  LXI     D,FCB
0204 0E10              MVI     C,CLOSEF
0206 CD0500            CALL    BDOS            ;close file
0209 C30000            JMP     CBOOT           ;boot
                ;
                ;*****************************************************
                ;
                ;   SUBROUTINES
                ;
                ;*****************************************************
                ;
                READ$DIR                        ;READ DIRECTORY'S 4 BLOCKS TO BUFFER
                                                ;BUFFER ADDRESS IS ALREADY IN BC-REG
020C 1E04              MVI     E,4             ;DIR IS 4 BLOCKS LONG
020E 210200            LXI     H,2             ;AND STARTS AT BLOCK #2
0211 CD4602            CALL    SYSRD           ;GET IT
0214 C9                RET
                ;
                ;
                FIND$FILE
0215 0E4D              MVI     C,77            ;STOP AFTER THE 77'TH ENTRY
0217 2AD804            LHLD    DENTP           ;GET STARTING ENTRY
                FI$SCH$LP
021A 110600            LXI     D,DTITLE        ;ADVANCE TO TITLE STRING
021D 19                DAD     D
021E 11E704            LXI     D,SYSTLE        ;SET DE-REG TO COMPARISON STRING
0221 3AE704            LDA     SYSTLE          ;comparison length
```

```
0224 3C             INR     A
0225 47             MOV     B,A
                FI$CMP$LP                       ;START COMPARING
0226 1A             LDAX    D
0227 BE             CMP     M
0228 C23402         JNZ     FI$CONT             ;IT'S NOT THIS ONE
022B 13             INX     D
022C 23             INX     H
022D 05             DCR     B                   ;IS THIS THE END OF THE STRING
022E C22602         JNZ     FI$CMP$LP
0231 C34502         JMP     FI$FOUND            ;FOUND IT
                FI$CONT
0234 2AD804         LHLD    DENTP               ;ON TO THE NEXT ENTRY
0237 111A00         LXI     D,DENTSZ
023A 19             DAD     D
023B 22D804         SHLD    DENTP
023E 0D             DCR     C                   ;IS THERE ANY DIR LEFT?
023F C21A02         JNZ     FI$SCH$LP
                FI$HANG                         ;FILE NOT THERE
0242 C3B403         JMP     NOFILE

                FI$FOUND                        ;GOT IT
0245 C9             RET
                ;
                ;
                SYSRD                           ;READ BLOCKS FROM PASCAL DISKETTE
0246 D5             PUSH    D                   ;SAVE BLOCK COUNT
0247 E5             PUSH    H                   ;AND BLOCK NUMBER
0248 CD5302         CALL    READ$RX             ;BUFFER IS ADVANCED BY 512 BYTES
024B E1             POP     H
024C D1             POP     D
024D 23             INX     H                   ;ADVANCE TO NEXT BLOCK
024E 1D             DCR     E
024F C24602         JNZ     SYSRD               ;SEE IF WE'RE DONE
0252 C9             RET
                ;
                ;
                READ$RX                         ;READ A PASCAL BLOCK
0253 29             DAD     H                   ;THERE ARE 4 SECTORS TO A BLOCK
0254 29             DAD     H                   ;MULT LOGICAL BLOCK BY 4
0255 1E04           MVI     E,4
                RR$LP                           ;THIS GETS CONFUSING
0257 C5             PUSH    B                   ;SET BUFFER ADDRESS
0258 D5             PUSH    D
0259 E5             PUSH    H
025A CD24BA         CALL    SETDMA
025D E1             POP     H                   ;NOW COMPUTE TRACK-SECTOR
025E E5             PUSH    H
025F CDD503         CALL    MAP                 ;CONVERTS LOGICAL SECTOR IN HL
0262 4C             MOV     C,H                 ;INTO TRACK, H, SECTOR, L
0263 E5             PUSH    H
0264 CD1EBA         CALL    SETTRK
0267 E1             POP     H
0268 4D             MOV     C,L
0269 CD21BA         CALL    SETSEC
026C 0E01           MVI     C,1
026E CD1BBA         CALL    SELDSK
0271 CD27BA         CALL    READ                ;AND READ THE DATA
0274 E1             POP     H
0275 D1             POP     D
0276 C1             POP     B
0277 E5             PUSH    H                   ;ADVANCE THE BUFFER ADDRESS
0278 218000         LXI     H,128
027B 09             DAD     B
027C 44             MOV     B,H
027D 4D             MOV     C,L
027E E1             POP     H
027F 23             INX     H                   ;ADVANCE BLOCK COUNT
0280 1D             DCR     E                   ;THEN SEE IF WE CONTINUE
0281 C25702         JNZ     RR$LP
0284 C9             RET                         ;LEAVE, WHEN DONE
                ;
                ;
0285 =          RDSEC   EQU     $               ;READ SECTOR GIVEN BY LSN
0285 0E1A           MVI     C,DMAADD
0287 CD0500         CALL    BDOS
028A 2ADF04         LHLD    LSN
028D CDD503         CALL    MAP
0290 4D             MOV     C,L
0291 CD21BA         CALL    SETSEC
0294 4C             MOV     C,H
0295 CD1EBA         CALL    SETTRK
0298 CD27BA         CALL    READ
029B B7             ORA     A
029C C2BA03         JNZ     RWERR
029F 2ADF04         LHLD    LSN
02A2 23             INX     H
02A3 22DF04         SHLD    LSN
02A6 C9             RET
                ;
02A7 2AE504     CT      LHLD    EOB             ;clear to end of buffer
02AA 23             INX     H
02AB EB             XCHG
02AC 2AE304         LHLD    BOL
02AF CDB902     CT1     CALL    EQUAL
02B2 C8             RZ
02B3 361A           MVI     M,CTRLZ
02B5 23             INX     H
02B6 C3AF02         JMP     CT1
                ;
02B9 7D         EQUAL   MOV     A,L             ;returns zero if (DE) = (HL)
02BA BB             CMP     E
02BB C0             RNZ
02BC 7C             MOV     A,H
02BD BA             CMP     D
02BE C9             RET
                ;
02BF 7B         COMPR   MOV     A,E             ;CY set if (DE) less than (HL)
02C0 95             SUB     L
02C1 7A             MOV     A,D
02C2 9C             SBB     H
02C3 C9             RET
                ;
02C4 0680       R80B    MVI     B,80H           ;read 80H bytes to buffer
02C6 218000         LXI     H,BUFF
02C9 C5         R80B1   PUSH    B
02CA E5             PUSH    H
02CB CD0F03         CALL    RB
02CE E1             POP     H
02CF C1             POP     B
02D0 F5             PUSH    PSW
02D1 3ADA04         LDA     EOFFLAG
02D4 B7             ORA     A
02D5 CADD02         JZ      R80B2           ;not eof
```

```
02D8 F1            POP     PSW
02D9 22E304        SHLD    BOL
02DC C9            RET
02DD F1       R80B2 POP     PSW
02DE 77             MOV     M,A
02DF 23             INX     H
02E0 05             DCR     B
02E1 C2C902         JNZ     R80B1
02E4 C9             RET
              ;
02E5 0E00     WB    MVI     C,0           ;write 80H byte buffer
02E7 CD1BBA         CALL    SELDSK
02EA 118000         LXI     D,BUFF
02ED 0E1A           MVI     C,DMAADD
02EF CD0500         CALL    BDOS
02F2 115C00         LXI     D,FCB
02F5 0E15           MVI     C,WRITER
02F7 CD0500         CALL    BDOS
02FA B7             ORA     A
02FB C2BA03         JNZ     RWERR
02FE C9             RET
              ;
              ;
02FF 7E       SCBKCH MOV    A,M           ;scan to char after first blank
0300 FE20           CPI     ' '
0302 CA0903         JZ      SCBK
0305 23             INX     H
0306 C3FF02         JMP     SCBKCH
0309 7E       SCBK  MOV     A,M           ;scan to first non-blank
030A FE20           CPI     ' '
030C C0             RNZ
030D 23             INX     H
030E C9             RET
              ;
030F =        RB    EQU     $             ;read byte
030F 3AD704         LDA     TXTFLG
0312 FE02           CPI     2             ;.TEXT?
0314 C23303         JNZ     RBFB
0317 3ADB04   RBCKLF LDA    NLF           ;need LF?
031A B7             ORA     A
031B CA2503         JZ      RBCKTB
031E AF             XRA     A
031F 32DB04         STA     NLF
0322 3E0A           MVI     A,LF
0324 C9             RET
0325 3ADC04   RBCKTB LDA    NTB           ;need tab?
0328 B7             ORA     A
0329 CA3303         JZ      RBFB
032C 3D             DCR     A
032D 32DC04         STA     NTB
0330 3E20           MVI     A,' '
0332 C9             RET
0333 2ADD04   RBFB  LHLD    BUFADD        ;get byte from buffer
0336 110012         LXI     D,BLKBUF+512  ;check for end of buffer
0339 CDB902         CALL    EQUAL
033C CA7503         JZ      RBLK
033F CDBF02         CALL    COMPR
0342 DA7503         JC      RBLK
0345 3AD704         LDA     TXTFLG
0348 FE02           CPI     2             ;.TEXT?
034A C26F03         JNZ     NOTEXT2
034D 7E             MOV     A,M
034E B7             ORA     A
034F CA7503         JZ      RBLK
0352 23             INX     H
0353 22DD04         SHLD    BUFADD
0356 FE0D           CPI     CR
0358 C25F03         JNZ     CKDLE
035B 32DB04         STA     NLF           ;put LF after CR
035E C9             RET
035F FE10     CKDLE CPI     DLE
0361 C0             RNZ
0362 7E             MOV     A,M
0363 23             INX     H
0364 22DD04         SHLD    BUFADD
0367 D620           SUI     32            ;set indentation
0369 32DC04         STA     NTB
036C C32503         JMP     RBCKTB
036F 7E       NOTEXT2 MOV   A,M
0370 23             INX     H
0371 22DD04         SHLD    BUFADD
0374 C9             RET
              ;
0375 =        RBLK  EQU     $             ;read LSN block
0375 2ADF04         LHLD    LSN
0378 EB             XCHG
0379 2AE104         LHLD    LSTLSN
037C CDB902         CALL    EQUAL
037F CAA803         JZ      SETEOF        ;found last sector
0382 0E01           MVI     C,1           ;select B disk
0384 CD1BBA         CALL    SELDSK
0387 110010         LXI     D,BLKBUF      ;read four sectors
038A CD8502         CALL    RDSEC
038D 118010         LXI     D,BLKBUF+80H
0390 CD8502         CALL    RDSEC
0393 110011         LXI     D,BLKBUF+100H
0396 CD8502         CALL    RDSEC
0399 118011         LXI     D,BLKBUF+180H
039C CD8502         CALL    RDSEC
039F 210010         LXI     H,BLKBUF
03A2 22DD04         SHLD    BUFADD
03A5 C33303         JMP     RBFB
03A8 21DA04   SETEOF LXI    H,EOFFLAG
03AB 3601           MVI     M,1
03AD C9             RET
              ;
              ; Error messages
03AE 110C04   NOFLNM LXI    D,NOFLNMSG
03B1 C3C203         JMP     GENERR
03B4 112004   NOFILE LXI    D,NOFMSG
03B7 C3C203         JMP     GENERR
03BA F630     RWERR ORI     '0'           ;make it ASCII
03BC 323604         STA     WERMSG
03BF 113604         LXI     D,WERMSG
03C2 0E09     GENERR MVI    C,WRITECB
03C4 CD0500         CALL    BDOS
03C7 C30000         JMP     CBOOT
              ;
03CA 118304   CERROR LXI    D,ERRMSG
03CD F630           ORI     '0'
03CF 328304         STA     ERRMSG
03D2 C3C203         JMP     GENERR
```

```
                        ;
                        MAP                             ;TURN LSN INTO IBM TRACK-SECTOR
                        ;
                        ;       NOTE    TRACK 00 IS NOT USED SO BLOCK 0
                        ;               IS AT TRACK 01 SECTOR 1
                        ;
                        ;       ON ENTRY        HL-REG HAS LOGICAL SECTOR NO.
                        ;       ON EXIT H-REG HAS PHYSICAL TRACK
                        ;                       L-REG HAS PHYSICAL SECTOR
                        ;
                        ;
03D5 C5                         PUSH    B
03D6 D5                         PUSH    D
                        ;
03D7 CDFB03                     CALL    DIV26
03DA 7D                         MOV     A,L
03DB 87                         ADD     A
03DC 47                         MOV     B,A
03DD 3E0C                       MVI     A,12
03DF BD                         CMP     L
03E0 D2E403                     JNC     MAPC
03E3 04                         INR     B
                        MAPC
03E4 4B                         MOV     C,E
03E5 AF                         XRA     A
03E6 57                         MOV     D,A
03E7 67                         MOV     H,A
03E8 68                         MOV     L,B
03E9 3E06                       MVI     A,6
                        MAP$LOOP
03EB 19                         DAD     D
03EC 3D                         DCR     A
03ED C2EB03                     JNZ     MAP$LOOP
03F0 C5                         PUSH    B
03F1 CDFB03                     CALL    DIV26
03F4 C1                         POP     B
03F5 2C                         INR     L
03F6 61                         MOV     H,C
03F7 24                         INR     H
03F8 D1                         POP     D
03F9 C1                         POP     B
03FA C9                         RET
                        ;
                        ;
                        DIV26
03FB 01E6FF                     LXI     B,-26
03FE 1EFF                       MVI     E,0FFH
                        DIVL
0400 1C                         INR     E
0401 09                         DAD     B
0402 7C                         MOV     A,H
0403 B7                         ORA     A
0404 F20004                     JP      DIVL
0407 011A00                     LXI     B,26
040A 09                         DAD     B
040B C9                         RET
                        ;
                        ;
                        ;
040C 0D0A4D4953NOFLNMSF         DB      CR,LF,'MISSING FILE NAME$'
0420 0D0A46494CNOFMSG           DB      CR,LF,'FILE DOES NOT EXIST$'
0436 2020524541WERMSG           DB      '  READ/WRITE ERROR$'
0449 0D0A46494CPERMSG           DB      CR,LF,'FILE ALREADY EXISTS. '
0460 0D0A432F52PERMSG1          DB      CR,LF,'C/R TO CONTINUE. CTRL-@ TO ABORT$'
0483 2020455252ERRMSG           DB      '  ERROR IN CREATE OR OPEN$'
049D 0D0A2E5445TEXT             DB      CR,LF,'.TEXT file being processed$'
04BA 0D0A2E434FCODE             DB      CR,LF,'.CODE file being processed$'
                                ;
                                ;
04D7 00         TXTFLG          DB      0
04D8 0000       DENTP           DW      0
04DA 00         EOFFLAG         DB      0
04DB 00         NLF             DB      0
04DC 00         NTB             DB      0
04DD 0000       BUFADD          DW      0
04DF 0000       LSN             DW      0
04E1 0000       LSTLSN          DW      0
04E3 0000       BOL             DW      0
04E5 0000       EOB             DW      0
04E7            SYSTLE          DS      22
                                ;
                                ;
                                ; CP/M EQUATES
                                ;
BA00 =          BOOT EQU CBIOS
BA03 =          WBOOT EQU BOOT+3
BA06 =          CONST EQU WBOOT+3
BA09 =          CONIN EQU CONST+3
BA0C =          CONOUT EQU CONIN+3
BA0F =          LIST EQU CONOUT+3
BA12 =          PUNCH EQU LIST+3
BA15 =          READER EQU PUNCH+3
BA18 =          HOME EQU READER+3
BA1B =          SELDSK EQU HOME+3
BA1E =          SETTRK EQU SELDSK+3
BA21 =          SETSEC EQU SETTRK+3
BA24 =          SETDMA EQU SETSEC+3
BA27 =          READ EQU SETDMA+3
BA2A =          WRITE EQU READ+3
                ;
0005 =          BDOS EQU 5
005C =          FCB EQU 5CH
0080 =          BUFF EQU 80H
                ;
0000 =          SYSRST EQU 0 ;SYSTEM RESET
0001 =          READC EQU 1 ;READ CONSOLE
0002 =          WRITEC EQU 2 ;WRITE CONSOLE
0003 =          READRD EQU 3 ;READ READER
0004 =          WRITEP EQU 4 ;WRITE PUNCH
0005 =          WRITEL EQU 5 ;WRITE LIST
                ;
0007 =          IOSTAT EQU 7 ;INTERROGATE I/O STATUS
0008 =          ALTIOS EQU 8 ;ALTER I/O STATUS
0009 =          WRITECB EQU 9 ;WRITE CONSOLE BUFFER
000A =          READCB EQU 10 ;READ CONSOLE BUFFER
000B =          CONSTAT EQU 11 ;CHECK CONSOLE STATUS
000C =          LIFTDH EQU 12 ;LIFT DISK HEAD
000D =          RSTDS EQU 13 ;RESET DISK SYSTEM
000E =          SELECTD EQU 14 ;SELECT DISK
000F =          OPENF EQU 15 ;OPEN FILE
0010 =          CLOSEF EQU 16 ;CLOSE FILE
0011 =          SEARCHF EQU 17 ;SEARCH FIRST
0012 =          SEARCHN EQU 18 ;SEARCH NEXT
0013 =          DELETEF EQU 19 ;DELETE FILE
0014 =          READR EQU 20 ;READ RECORD
0015 =          WRITER EQU 21 ;WRITE RECORD
0016 =          CREATEF EQU 22 ;CREATE FILE
0017 =          RENAMEF EQU 23 ;RENAME FILE
0018 =          LOGINV EQU 24 ;GET LOGIN VECTOR
0019 =          LOGGED EQU 25 ;GET LOGGED DISK
001A =          DMADD EQU 26 ;SET DMA ADDRESS
001B =          ALLOCV EQU 27 ;GET ALLOCATION VECTOR
                ;
00FF =          ERNXF EQU 255 ;NON-EXISTANT FILE
0001 =          EREOF EQU 1 ;END OF FILE RETURN CODE
0002 =          ERURA EQU 2 ;READ UNWRITTEN DATA
0001 =          EREXT EQU 1 ;ERROR IN EXTENDING FILE
0002 =          EREOD EQU 2 ;END OF DISK DATA
00FF =          ERNDR EQU 255 ;NO MORE DIRECTORY SPACE
                ;
000D =          CR EQU 13
000A =          LF EQU 10
001B =          ESC EQU 1BH
0000 =          MODE EQU 0
001A =          CTRLZ EQU 1AH
                ;
```

the end

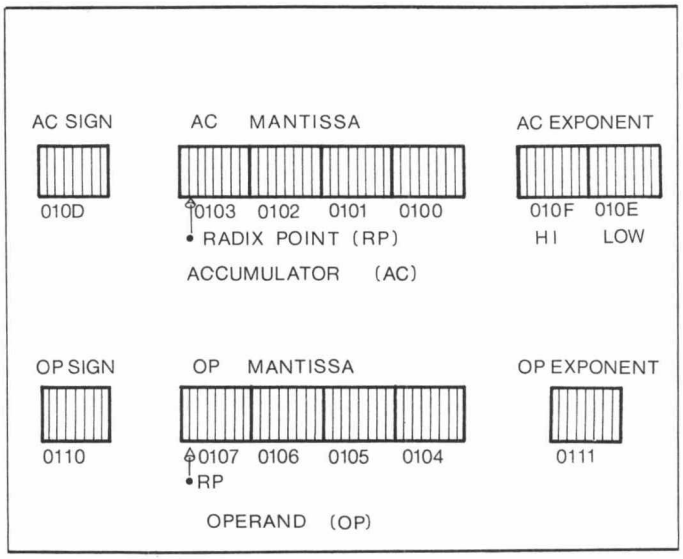
A Floating Point Subroutine Package for the 1802

BY PAUL WASSERMAN
918 Ray Avenue
Union, New Jersey 07083

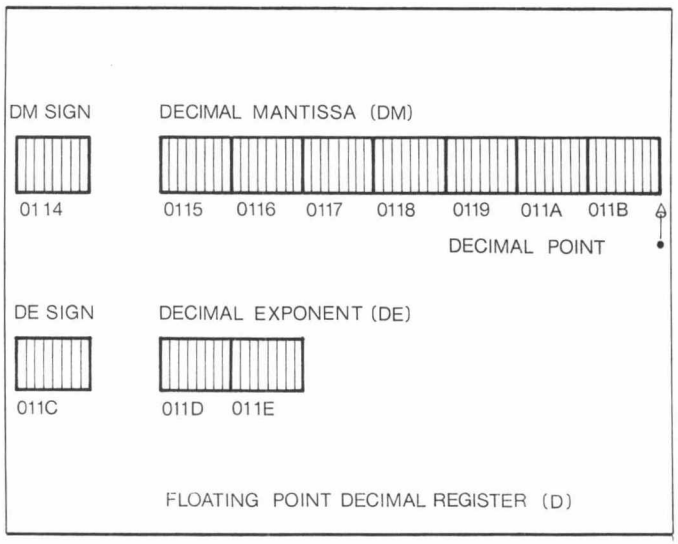
After purchasing my 1802 based microcomputer system, I was surprised to find how little software there is available for it. In particular, I was disappointed by the absence of a readily-available floating point arithmetic subroutine package. To help wipe out this void, I set out to write such a subroutine package. The results of my work appear in the rest of this article.

When this project was started, I decided that the floating point arithmetic routines would implement a set of existing algorithms rather than writing a new set. Suitable algorithms were found in a book written by Neill Graham entitled *Microprocessor Programming for Computer Hobbyists*, published by Tab Books. I strongly recommend the purchase of this book to anyone who wishes to understand or modify the internal workings of this set of floating point arithmetic routines. This article will not attempt to describe the routines' inner workings. It will, however, provide the information needed to use these routines.

All floating point numbers contain two parts: a signed mantissa and a signed exponent. $+1.16 \times 10^{+2}$ is an example of a decimal floating point number. +1.16 is the signed mantissa and +2 is the signed exponent. Similarly, $+1.1101 \times 2^{+110}$ is the binary floating point equivalent of 116. Here, +1.1101 is the signed mantissa and +110 is the signed exponent. The binary mantissa used by the floating point subroutine is 32 bits long (4 bytes). The radix point lies between the left-most bit (LMB) and the bit to the immediate right of the LMB. The sign of the mantissa is represented by a single byte which can take on two values, OO (hex) or FF (hex). If this byte is OO (hex), the mantissa is positive. Similarly, if this byte is FF (hex), the mantissa is negative. The signed binary exponent is represented by a single byte. In order to have both positive and negative integers in the single byte exponent, a convenient numbering scheme was adopted. This scheme defined 80 (hex) as zero. Anything larger than 80 (hex) is positive and anything smaller than 80 (hex) is negative. For example, 83 (hex) represents a binary exponent of +11 = +3 (decimal) and similarly 7F (hex) represents a binary exponent of −1. The exponent can vary between OO (hex) and FF (hex). These numbers represent decimal exponents between the range of 10^{-39} and 10^{38}. Figure 1 illustrates the representation of floating point numbers in the accumulator (AC) and operand (OP) registers. Notice that the accumulator exponent is 2 bytes wide. The hi byte is OO (hex) unless an over or underflow has occurred. The hexidecimal numbers placed below the diagrams in Figure 1 are the memory locations used to store (in RAM) the bytes that make up the accumulator and operand.

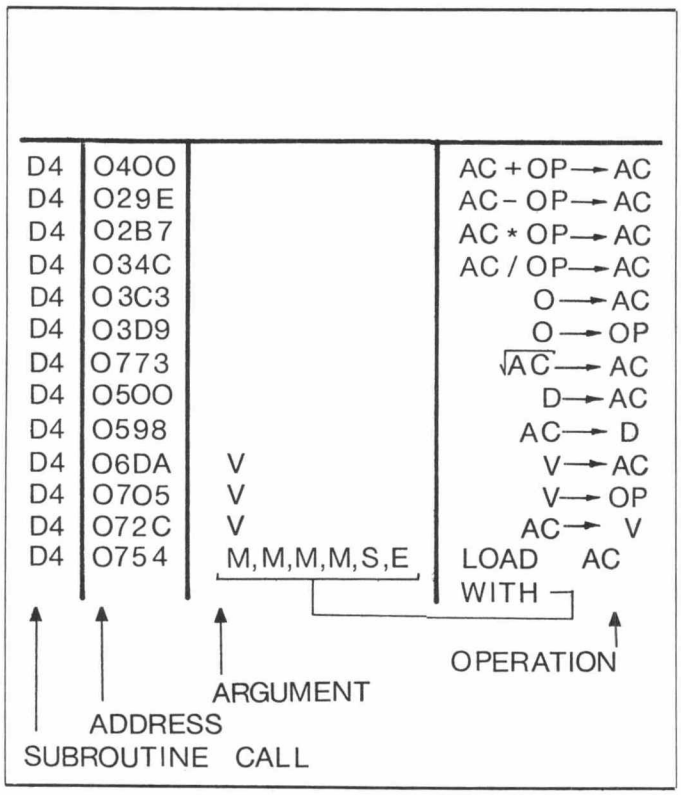


Most people can handle floating point decimal numbers with relative ease. The computer, unlike most people, is more efficient at handling binary numbers. To accommodate this difference, two routines are included in the floating point arithmetic package which allow for the conversion between binary and decimal numbers. Both of these routines use the accumulator register to hold binary numbers. An additional register is used by both routines for decimal numbers. This decimal register (D) is shown in Figure 2. The decimal register is composed of four parts: the sign of the mantissa (DM Sign), the 7-digit decimal mantissa (DM), the sign of the exponent (DE Sign) and the 2-digit exponent (DE). Both signs are represented by their standard ASCII codes; that is, 2D (hex) for negative and 2B (hex) for positive. The decimal digits (0-9) of both the exponent for the mantissa are represented by their corresponding 4-bit binary codes. These codes are placed in the four least significant bits of each byte. The four most significant bits are ignored by the conversion routines. This allows the use of ASCII codes on all entries in decimal register. For example, the number $+1.234567 \times 10^{+2} = +1234567. \times 10^{-4}$ could be represented in the decimal register in the following ways: 2B 01 02 03 04 05 06 07 2D 00 04 or 2B 31 32 33 34 35 36 37 2D 30 34 (ASCII).



These conversion routines appear in Figure 3 as D → AC (decimal to binary conversion) and AC → D (binary to decimal conversion).

All the routines in the floating point arithmetic package use the Standard Call and Return Technique (SCRT). This technique is explained in RCA's User Manual for the CDP 1802 on pgs. 61-64. The microprocessor registers R(2)–R(5) must be set up to handle the SCRT before any subroutines can be used. The user must also provide the SCRT code before using any subroutines. A subroutine is called by a D4 (hex) instruction, followed by the address of the subroutine. In some cases, additional numbers (labeled arguments in Figure 3) follow the address. Figure 3 lists all the user subroutines in the floating point arithmetic package.



The top four routines shown in Figure 3 perform floating point addition, subtraction, multiplication and division, respectively. The next two routines clear (set to zero) the contents of the accumulator or the operand. A square root routine follows next on the list. This routine uses the Newton-Raphson iterative method to find the square root of the contents of the accumulator. The next two routines perform conversions between binary and decimal numbers as previously described. The next three routines allow the user to store and recall numbers from memory. There are forty-two variable storage registers (referred to as variables) numbered 00(hex)-29 (hex). Any one of these variables can be loaded into either the accumulator or operand registers. Similarly, the contents of the accumulator can be stored in any one of the variables. For example, if we wanted to load the accumulator with the contents of variable 08, the following code would be used D4 06DA 08 (08 = V). The last routine shown in Figure 3 loads the accumulator with the binary floating point number that immediately follows the instruction. The first four bytes following the calling address (0754) make up the binary mantissa. These bytes should be in reverse order. For example, if the mantissa equals 42 6C 3D 0E, this subroutine requires 0E, 3D, 6C, 42 be placed after the calling address. As shown in Figure 3, the mantissa sign (S) and exponent (E) should be placed immediately after the mantissa.

The example shown in Figure 4 illustrates how some of these routines may be used to find the area of a circle, given its radius.

```
D4  0500                        D ——→ AC      Convert radius (R) to binary
D4  072C  01                    AC ——→ V1     Store R in variable1 (V1)
D4  0754  08,EE,87,64,00,82     π ——→ AC      Load AC with π
D4  0705  01                    V1 ——→ OP     Load OP with V1=R
D4  02B7                        AC*OP ——→ AC  AC = πR
D4  02B7                        AC*OP ——→ AC  AC = πR²
D4  0598                        AC ——→ D      Convert back to decimal
                                              πR² is in D
```

Figure 5 is a listing of the hex code for the floating point arithmetic subroutine package. These subroutines start at the beginning of memory page 01 and end at the last byte of page 08. Page 08 is used for the variable storage. If you want to change the location in memory of these subroutines you must always start them at the beginning of a page. Also, certain modifications to the routines must be made. For example, if you want to move the beginning of the subroutine package two pages, from page 01 to page 03 in memory, you must add 02 to *all* of the underlined bytes. You would also have to add 02 to the high order byte of all the routine calling addresses. This subroutine package uses the following 1802 registers: R(A), R(B), and R(C). In addition, the square root routine uses register R(D) and variables 26 (hex) and 27 (hex). Please note that the SCRT (which you must supply) uses R(2), R(3), R(4) and R(5).

Using this floating point arithmetic package, I have been able to write routines that generate exponentials, sines and cosines. I hope that you will find these routines as useful as I have.

```
0100  A0 A1 A2 A3 00 01 02 03 00 00 00 00 00 A5 AE AE
0110  05 0E 00 00 5D D1 D2 D3 D4 D5 D6 D7 5E E1 E2 00
0120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 F8 01
0130  BA F8 00 AA EA F0 BB F8 04 AA 9B F4 BB F8 00 AA
0140  9B 5A F8 05 AA F0 BB F8 01 AA 9B 74 5A F8 06 AA
0150  F0 BB F8 02 AA 9B 74 5A F8 07 AA F0 BB F8 03 AA
0160  9B 74 5A D5 F8 01 BA F8 04 AA EA F0 BB F8 00 AA
0170  9B F5 5A F8 05 AA F0 BB F8 01 AA 9B 75 5A F8 06
0180  AA F0 BB F8 02 AA 9B 75 5A F8 07 AA F0 BB F8 03
0190  AA 9B 75 5A D5 F8 01 BA F8 00 AA EA F8 01 F4 5A
01A0  1A F8 00 74 5A 1A F8 00 74 5A 1A F8 00 74 5A D5
01B0  F8 01 BA F8 00 AA EA F8 00 F7 5A 1A F8 00 77 5A
01C0  1A F8 00 77 5A 1A F8 00 77 5A D5 F8 01 BA F8 00
01D0  AA EA F0 FE 5A 1A F0 7E 5A 1A F0 7E 5A 1A F0 7E
01E0  5A D5 F8 01 BA F8 08 AA EA F0 FE 5A 1A F0 7E 5A
01F0  1A F0 7E 5A 1A F0 7E 5A D5 F8 01 BA F8 03 AA EA
0200  F0 F6 5A 2A F0 76 5A 2A F0 76 5A 2A F0 76 5A D5
0210  F8 01 BA F8 0B AA EA F0 F6 5A 2A F0 76 5A 2A F0
0220  76 5A 2A F0 76 5A D5 F8 01 BA F8 07 AA EA F0 FE
0230  F0 76 5A 2A F0 76 5A 2A F0 76 5A 2A F0 76 5A D5
0240  F8 01 BA F8 00 AA EA F0 BB F8 08 AA 9B 5A F8 01
0250  AA F0 BB F8 09 AA 9B 5A F8 02 AA F0 BB F8 0A AA
0260  9B 5A F8 03 AA F0 BB F8 0B AA 9B 5A F8 00 AA 5A
0270  1A 5A 1A 5A 1A 5A F8 01 AB D4 01 F9 D4 02 10 3B
0280  84 D4 01 2E 8B FB 1F 32 8C 1B 30 79 D5 D4 01 64
0290  F8 01 AB D4 01 E2 D4 01 CB 3B A0 D4 01 2E 30 AE
02A0  D4 01 64 F8 01 BA F8 08 AA EA F0 F9 01 5A 8B FB
02B0  20 32 B6 1B 30 93 D5 F8 01 BA F8 10 AA EA F0 BB
02C0  F8 0D AA 9B F3 5A F8 11 AA F0 BB F8 0E AA 9B F4
02D0  5A 1A F0 7C 00 5A F8 0E AA F0 FF 81 5A 1A F0 7F
02E0  00 5A D4 02 40 D4 03 00 D5 F8 01 BA F8 10 AA EA
02F0  F0 FB FF 5A D4 04 00 D5 00 00 00 00 00 00 00 00
0300  F8 01 BA F8 00 AA EA F0 1A F1 1A F1 1A F1 3A 19
0310  F8 0F AA F8 00 73 73 73 D5 F8 03 AA F0 FA 80 32
0320  31 D4 01 F9 F8 0E AA EA F0 FC 01 5A 1A F0 7C 00
0330  5A F8 03 AA F0 FA 40 3A 4B D4 01 CB F8 0E AA EA
0340  F0 FF 01 5A 1A F0 7F 00 5A 30 31 D5 F8 01 BA F8
0350  10 AA EA F0 BB F8 0D AA 9B F3 5A F8 11 AA F0 BB
0360  F8 0E AA 9B F5 5A 1A F0 7F 00 5A 2A F0 FC 80 5A
0370  1A F0 7C 00 5A D4 02 8D F8 01 BA F8 08 AA EA F0
0380  BB F8 00 AA 9B 5A F8 09 AA F0 BB F8 01 AA 9B 5A
0390  F8 0A AA F0 BB F8 02 AA 9B 5A F8 0B AA F0 BB F8
03A0  03 AA 9B 5A D4 03 00 D5 F8 01 BA F8 04 AA EA F8
03B0  00 F7 5A 1A F8 00 77 5A 1A F8 00 77 5A 1A F8 00
03C0  77 5A D5 F8 01 BA F8 03 AA EA F8 00 73 73 73 5A
03D0  F8 0F AA F8 00 73 73 73 D5 F8 01 BA F8 07 AA EA
03E0  F8 00 73 73 73 73 F8 11 AA F8 00 73 73 D5 00 00
03F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0400  F8 01 BA F8 11 AA EA F0 BB F8 0E AA 9B F7 3B 76
0410  F8 10 AA F0 BB F8 0D AA F0 AB 9B 5A F8 10 AA 8B
0420  5A F8 11 AA F0 BB F8 0E AA F0 AB 9B 5A F8 11 AA
0430  8B 5A F8 00 AA F0 BB F8 04 AA F0 AB 9B 5A F8 00
0440  AA 8B 5A F8 01 AA F0 BB F8 05 AA F0 AB 9B 5A F8
0450  01 AA 8B 5A F8 02 AA F0 BB F8 06 AA F0 AB 9B 5A
0460  F8 02 AA 8B 5A F8 03 AA F0 BB F8 07 AA F0 AB 9B
0470  5A F8 03 AA 8B 5A F8 0E AA F0 BB F8 11 AA 9B F7
0480  AC FF 1E 3B 86 D5 F8 0D AA F0 BB F8 10 AA 9B F3
0490  32 95 D4 03 A8 8C 32 9E 2C D4 02 27 30 95 D4 01
04A0  2E F8 01 BA F8 0D AA EA F0 BB F8 10 AA 9B F3 32
04B0  CA F8 03 AA F0 FA 80 32 CA D4 01 B0 F8 01 BA F8
04C0  10 AA EA F0 BB F8 0D AA 9B 5A D4 03 00 D5 D4 01
04D0  CB F8 01 BA F8 00 AA EA F0 BB 1A F0 AB F8 04 AA
04E0  9B 5A 1A 8B 5A F8 02 AA F0 BB 1A F0 AB F8 06 AA
04F0  9B 5A 1A 8B 5A D4 01 CB D4 01 CB D4 01 2E D5 00
0500  D4 03 C3 F8 00 AC D4 03 D9 8C FC 15 AA F8 01 BA
0510  EA F0 FA 0F BB F8 04 AA 9B 5A D4 01 2E D4 04 CE
0520  1C 8C FB 07 3A 06 F8 01 BA F8 0E AA EA F8 9F 5A
0530  D4 03 00 F8 01 BA F8 14 AA EA F0 FB 2B 32 45 F8
0540  0D AA F8 FF 5A F8 1D AA F0 FA 0F FE BB F8 13 AA
0550  9B 5A FE FE F4 5A F8 1E AA F0 FA 0F BB F8 13 AA
0560  9B F4 AC D4 03 D9 F8 01 BA F8 11 AA EA F8 84 5A
0570  F8 07 AA F8 50 5A D4 03 4C F8 01 BA F8 1C AA EA
0580  F0 FB 2B 32 8E 8C 32 97 2C D4 03 4C 30 85 8C 32
0590  97 2C D4 02 B7 30 8E D5 D4 03 D9 F8 01 BA F8 11
05A0  AA EA F8 84 5A F8 07 AA F8 50 5A F8 12 AA F8 7A
05B0  5A F8 0E AA F0 FF 85 3B C9 D4 03 4C F8 01 BA F8
05C0  12 AA EA F0 FC 01 5A 30 B1 F8 0E AA F0 FF 81 33
05D0  E1 D4 02 B7 F8 01 BA F8 12 AA EA F0 FF 01 5A 30
05E0  C9 F8 0E AA F0 FB 84 3A FB F8 03 AA F0 FF 51 3B
05F0  FB F8 12 AA F0 FC 01 5A D4 03 4C F8 01 BA F8 0E
0600  AA EA F0 FB 84 3A 0C D4 01 CB 30 21 F0 FF 83 33
0610  21 D4 01 F9 F8 01 BA F8 0E AA EA F0 FC 01 5A 30
0620  0C D4 03 D9 F8 01 BA F8 04 AA EA F8 86 5A D4 01
0630  2E F8 01 BA F8 03 AA EA F0 FF A0 3B 52 F8 12 AA
0640  F0 FC 01 5A F8 03 AA F8 10 5A F8 00 2A 5A 2A 5A
0650  2A 5A F8 03 AA F0 BB F8 15 AA 9B F6 F6 F6 F6 5A
0660  F8 06 AC 8C 32 86 F8 03 AA F0 FA 0F 5A D4 04 CE
0670  F8 01 BA F8 03 AA EA F0 BB 8C FD 1C AA 9B F6 F6
0680  F6 F6 5A 2C 30 63 F8 0D AA F0 32 96 F8 2D BB F8
0690  14 AA 9B 5A 30 9E F8 2B BB F8 14 AA 9B 5A F8 00
06A0  AC AB F8 1C AA F8 2B 5A F8 12 AA F0 FF 80 33 BD
06B0  FB FF FC 01 5A F8 1C AA F8 2D 5A 30 BE 5A F8 12
06C0  AA F0 FF 0A 3B CA 5A 1B 30 C1 F8 1D AA 8B 5A F8
06D0  12 AA F0 BB F8 1E AA 9B 5A D5 F8 01 BA F8 12 AA
06E0  EA 46 FE 5A FE F4 AB F8 08 BB F8 00 AA 4B 5A 1A
06F0  4B 5A 1A 4B 5A 1A 4B 5A F8 0D AA 4B 5A 1A 4B 5A
0700  1A F8 00 5A D5 F8 01 BA F8 12 AA EA 46 FE 5A FE
0710  F4 AB F8 08 BB F8 04 AA 4B 5A 1A 4B 5A 1A 4B 5A
0720  1A 4B 5A F8 10 AA 4B 5A 1A 4B 5A D5 F8 01 BA F8
0730  12 AA EA 46 FE 5A FE F4 AB F8 08 BB F8 00 AA 4A
0740  5B 1B 4A 5B 1B 4A 5B 1B 4A 5B 1B F8 0D AA 4A 5B
0750  1B 4A 5B D5 F8 01 BA F8 00 AA EA 46 5A 1A 46 5A
0760  1A 46 5A 1A 46 5A F8 0D AA 46 5A 1A 46 5A 1A F8
0770  00 5A D5 F8 0A AD D4 07 2C 26 F8 01 BA F8 0E AA
0780  0A FF 01 5A D4 07 2C 27 8D 32 AE D4 06 DA 26 D4
0790  07 05 27 D4 03 4C D4 07 05 27 D4 04 00 F8 01 BA
07A0  F8 0E AA 0A FF 01 5A D4 07 2C 27 2D 30 88 D5 00
07B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
07C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
07D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
07E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
07F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0800  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0810  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0820  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0830  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0840  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0850  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0860  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0870  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0880  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0890  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
08A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
08B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
08C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
08D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
08E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
08F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

THOSE ALL-IMPORTANT EXTRAS

BY DON COLBURN
EG&G Washington Analytical Services Center, Inc.
2150 Fields Road
Rockville, Maryland 20850

Last month we brought you Don Colburn's EXOS – A Software Development Tool Kit for the 6500 Microprocessor Family. *This article gives you a couple of extra tools for your software development, a flashlight and a sledge hammer. –SMR*

A FLASHLIGHT FOR YOUR SOFTWARE DEVELOPMENT TOOL KIT

One of the most important and most often used tools in a software development environment is the capability to display, in a usable format, the contents of memory. A good tool provides the data formatted in such a way that the address of any displayed byte is readily apparent without having to count in from the left. The following program was written on a 6502 based CGRS system utilizing EXOS® and provides a stand alone version of the EXOS "DISPLAY" command. The provided code begins at hex address $0200 and is entered by utilizing the TIM monitor to initiate execution at that address. Upon entry the user is prompted with a '>' to enter, in hex, the starting and ending address of the range of memory to be displayed. Following entry of a carriage return, the display is output. As may be seen in the accompanying example, a header field is output every 16 lines, thus providing column headers. The starting address of the first memory location in the current line is displayed in the first column in hex. This is followed, on the same line, by both the hexidecimal and ASCII representation of the following 16 bytes. Lines of data are generated until the ending address has been displayed. At this point, the monitor is re-entered.

Translation of this display algorithm to another machine may be easily accomplished due to the indepth level of commenting.

EXAMPLE OF DISPLAY COMMAND

```
.R  F003 58 1F 58 55 FF
.:  0200
.G
> 0200 0301

      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

0200 D8 20 48 02 20 8A 72 20 71 02 A2 00 86 FF 78 20   . H. .r q.....x
0210 8A 72 20 92 02 20 9A 72 20 A4 02 20 77 73 A0 00   .r .. .r .. ws..
0220 B1 EE 20 B1 72 20 97 73 20 C1 70 90 15 A5 EE 29   .. .r .s .p....)
0230 0F D0 E8 20 BD 02 58 E6 FF A9 10 C5 FF F0 C5 4C   ... ..X........L
0240 0E 02 20 B4 02 4C BD 02 20 8A 72 A9 3E 20 C6 72   .. ..L.. .r.> .r
0250 20 77 73 20 A4 73 20 87 73 20 77 73 20 A4 73 20    ws .s .s ws .s
0260 87 73 20 E9 72 C9 0D D0 DF A6 E4 D0 DB 20 8A 72   .s .r......... .r
0270 60 20 8A 72 20 89 02 A0 00 98 20 B1 72 20 77 73   `  .r ..... .r ws
0280 C8 C0 10 D0 F4 20 8A 72 60 A2 05 20 77 73 CA D0   .... .r`.. ws..
0290 FA 60 A5 EE 85 F4 A5 EF 85 F5 60 A5 F4 85 EE A5   .`........`.....
02A0 F5 85 EF 60 A5 EE 29 0F F0 09 85 FD 0A 65 FD AA   ...`..).....e..
02B0 20 8B 02 60 A9 11 E5 EE 29 0F D0 EE 60 20 9B 02    ..`....)...`  ..
02C0 A2 02 20 8B 02 20 E0 02 20 EB 02 20 C6 72 20 97   ..  ..  ..  .r .
02D0 73 20 C1 70 90 07 A5 EE 29 0F D0 EC 60 4C 86 70   s .p....)...`L.p
02E0 A5 EE 29 0F F0 04 AA 20 8B 02 60 A0 00 B1 EE C9   ..)....  ..`.....
02F0 20 90 05 C9 7F B0 01 60 A9 2E 60 00 00 00 00 00    ......`..`.....

      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

0300 2C D7                                             ,.
```

A SLEDGE HAMMER FOR YOUR SOFTWARE DEVELOPMENT TOOL KIT

Integration at any Disk System into a software development environment has a tremendous impact upon the entire system. It is because of this impact, and the rapid rate at which technology tends to obsolete today's "state of the art" hardware, that one must be careful not to tie the Disk System to a specific cpu or bus. The PerSci 1070 disk controller with a 277 dual drive provides complete independence from both the resident cpu and bus. The new CGRS S100 F10 Disk Controller/2 serial/4 parallel IO card allows the 1070 controller to be interfaced to the S100 bus for use by any cpu.

The following listing is based upon the September '77 Interface Age articles which detail the PerSci controller hardware as well as flow charts and listings for a 8080 driver. This Program is written for a 650X TIM® based system with the PerSci controller addresssed at Hex $6000. Both the Flow Charts and the logic descriptions from the articles apply to this code.

EXOS® is a Trademark of RCS Associates, NY.

TIM is a trademark of MOS Technology, PA.

```
EXOS ASSEMBLER VER 3.4
SAVE SYMBOL TABLE  Y/N? N
SYMBOL TABLE START = $2000
ENABLE RDR.  HIT ANY KEY

                        ; THIS PROGRAM DISPLAYS MEMORY CONTENTS IN AN EASILY
                        ; READIBLE FORMAT IN BOTH HEX AND ASCII.
                        ;
                        ; IT REQUIRES A TIM BASED 6500 SYSTEM WITH MEMORY
                        ; FROM $0200 TO 02FA
                        ;
                        ;
                        ; TIM Z-PAGE LOCATIONS
0000                    WRAP   = $E4      ; ADDRESS WRAP FLAG
0000                    TMP0   = $EE      ; TEMPORARY REGISTERS
0000                    TMP01  = $EF
0000                    TMP6   = $F4
0000                    TMP61  = $F5
0000                    SAVX   = $FD      ; X TEMP SAVE
0000                    LCNT   = $FF
                        ;
                        ; TIM SUBROUTINES
                        ;
0000                    CRLF   = $728A    ; CRLF TO CONSOLE
0000                    DCMP   = $70C1    ; DBL SUBTRACT TMP2-TMP0
0000                    INCTMP = $7397    ; INCREMENT (TMP0,TMP0+1) BY 1
0000                    RDOA   = $73A4    ; READ CHAR, PUT TMP0, CY=1 IF SPACE
0000                    RDOC   = $72E9    ; READ 1 CHAR
0000                    SPC    = $7377    ; PRINT SPACE (PROTECTS ALL REGS)
0000                    START  = $7086    ; ENTER TIM
0000                    T2T2   = $7387    ; TMP0 -> TMP2
0000                    WROA   = $729A    ; WRITE ADDRESS FROM TMP0
0000                    WROB   = $72B1    ; CONVERT CHAR(ASCII) TO HEX(A,X) ASCII
0000                    WRT    = $72C6    ; PRINT 1 CHAR
                        ;
0000                            .ORG $0200
                        ;
0200 D8                 DSPMEM  CLD       ; SET BINARY MODE
0201 20 0202                    JSR GETRNG ; FETCH DISPLAY RANGE
0204 20 8A72            DSPMEM1 JSR CRLF
0207 20 0202                    JSR PRHDR  ; PRINT HEADER
020A A2 00                      LDX #0
020C 86 FF                      STX LCNT
020E 78                 DSPMEM2 SEI       ; DISABLE INTERRUPTS
020F 20 8A72                    JSR CRLF  ; NEXT LINE
0212 20 0202                    JSR SAVSAD ; SAVE STARTING ADDRESS
0215 20 9A72                    JSR WROA  ; OUTPUT STARTING ADDRESS
0218 20 0202                    JSR PRLNSP ; COMPUTE,OUTPUT LINE SPACES
021B 20 7773            DSPMEM3 JSR SPC   ; 1 MORE SPACE
021E 20 C170                    JSR DCMP  ; SUB S.A. FROM E.A.
0220 90 00                      BCC EXLINE ; IF E.A. < S.A. THEN EXIT
0222 B1 EE                      LDA (TMP0),Y ; A = S.A. + INDEX
0224 20 B172                    JSR WROB  ; PRINT A IN HEX
0227 20 9773                    JSR INCTMP ; INCR S.A.
022A 20 C170                    JSR DCMP  ; SUB S.A. FROM E.A.
022D A5 EE                      LDA TMP0  ; UNTIL 4 LSB'S OF TMP0 = 0
022F 29 0F                      AND #$0F
0231 D0 E8                      BNE DSPMEM3 ; CONVERT,PRINT NEXT LOCATION
0233 20 0202                    JSR PRVASC ; END OF HEX LINE, PRINT ASCII
0236 58                         CLI       ; ALLOW INTERRUPTS
0237 E6 FF                      INC LCNT  ; NEXT LINE
0239 A9 10                      LDA #16
023B C5 FF                      CMP LCNT  ; IF LINE COUNT = 16
023D F0 C5                      BEQ DSPMEM1 ; THEN PRINT HEADER
023F 4C 0E02                    JMP DSPMEM2 ; ELSE CONTINUE
                        ;
0242 20 0202            EXLINE  JSR PRENSP ; OUTPUT TRAILING SPACES
0245 4C 0202                    JMP PRVASC ; OUTPUT ASCII FIELD
                        ;
0248 20 8A72            GETRNG  JSR CRLF  ; NEW LINE
024B A9 3E                      LDA #'>
024D 20 C672                    JSR WRT   ; OUTPUT PROMPT,SPACE
0250 20 7773                    JSR SPC
0253 20 A473                    JSR RDOA  ; READ ADDRESS M.S. BYTE
0256 20 8773                    JSR T2T2  ; SWAP TMP0,TMP2
0259 20 7773                    JSR SPC
025C 20 A473                    JSR RDOA  ; READ ADDRESS L.S. BYTE
025F 20 8773                    JSR T2T2  ; SWAMP BACK TMP0,TMP2
0262 20 E972                    JSR RDOC  ; WAIT FOR CR
0265 C9 0D                      CMP #$0D
0267 D0 DF                      BNE GETRNG ; IF NOT CR, THEN REPROMPT
0269 A6 E4                      LDX WRAP  ; ELSE IF ADDRESS WRAP,
026B D0 DB                      BNE GETRNG ; REPROMPT
026D 20 8A72                    JSR CRLF
0270 60                         RTS       ; RETURN
                        ;
0271 20 8A72            PRHDR   JSR CRLF  ; NEXT LINE
0274 20 0202                    JSR PRSP5 ; START 5 SPACES IN
0277 A0 00                      LDY #0    ; COUNT = 0
0279 98                 PRHDR1  TYA       ; DATA = COUNT
027A 20 B172                    JSR WROB  ; PRINT DATA IN HEX
027D 20 7773                    JSR SPC   ; SPACE
0280 C8                         INY       ; BUMP COUNT
0281 C0 10                      CPY #16   ; UNTIL COUNT = 16
0283 D0 F4                      BNE PRHDR1 ; PRINT NEXT DATA
0285 20 8A72                    JSR CRLF  ; THEN NEXT LINE
0288 60                         RTS       ; RETURN
                        ;
```

```
0289  A2 05     PRSP5   LDX #5        ; COUNT = 5
028B  20 7773   PRSPAC  JSR SPC       ; UNTIL COUNT = 0, OUTPUT SPACES
028E  CA                DEX
028F  D0 FA             BNE PRSPAC
0291  60                RTS           ; THEN RETURN
                ;
0292  A5 FE     SAVSAD  LDA TMPO      ; SAVE S.A.(TMPO) IN TEMP6
0294  85 F4             STA TMP6
0296  85 EF             LDA TMPO1
0298  85 F5             STA TMP61
029A  60                RTS           ; THEN RETURN
                ;
029B  A5 F4     RESAD   LDA TMP6      ; RESTORE S.A.(TEMP6) IN TMPO
029D  85 EE             STA TMPO
029F  A5 F5             LDA TMP61
02A1  85 EF             STA TMPO1
02A3  60                RTS           ; THEN RETURN
                ;
02A4  A5 EE     PRLNSP  LDA TMPO      ; PICKUP L.S. BYTE OF ADDRESS
02A6  29 0F             AND #$0F      ; MASK OFF 4 LSB'S
02A8  F0 00             BEQ PRLNSP2   ; IF 0, RETURN
02AA  85 FD     PRLNSP1 STA SAVX      ; ELSE HOLD COUNT
02AC  0A                ASL A         ; COUNT = COUNT * 3
02AD  65 FB             ADC  SAVX
02AF  AA                TAX           ; X =#SPACES
02B0  20 8B02           JSR PRSPAC    ; OUTPUT SPACES
02B3  60        PRLNSP2 RTS
                ;
02B4  A9 11     PRENSP  LDA #$11      ; MASK WITH INC ADJUSTMENT
02B6  E5 EE             SBC TMPO      ; SPACE CALCULATION
02B8  29 0F             AND #$0F      ; MAX = 16
02BA  D0 EE             BNE PRLNSP1   ; CONVERT TO COUNT AND OUTPUT SPACES
02BC  60                RTS           ; IF COUNT = 0, RETURN
                ;
02BD  20 9B02   PRVASC  JSR RESAD     ; RESTORE S.A.
02C0  A2 02             LDX #2
02C2  20 8B02           JSR PRSPAC    ; PRINT 2 SPACES
02C5  20 0202           JSR PRVASP    ; COMPUTE SPACES INTO ASCII FIELD
02C8  20 0202   ASCI    JSR VALASC    ; IF A NOT ASCII, A = '.'
02CB  20 C672           JSR WRT       ; OUTPUT CHAR
02CE  20 9773           JSR INCTMP    ; BUMP POINTER
02D1  20 C170           JSR DCMP      ; TEST FOR E.A.< S.A.
02D4  90 00             BCC ASC2      ; IF< EXIT
02D6  A5 EE             LDA TMPO      ; UNTIL 4 LSB'S = 0
02D8  29 0F             AND #$0F
02DA  D0 EC             BNE ASCI      ; DO FOR NEXT LOCATION
02DC  60                RTS           ; RETURN
02DD  4C 8670   ASC2    JMP START     ; ENTER TIM
                ;
02E0  A5 EE     PRVASP  LDA TMPO      ; PRINT SPACES IN ASCII FIELD
02E2  29 0F             AND #$0F      ; MAX 16
02E4  F0 00             BEQ ASP2      ; IF NONE, RETURN
02E6  AA        ASP1    TAX
02E7  20 8B02           JSR PRSPAC    ; ELSE PRINT SPACES, X=COUNT
02EA  60        ASP2    RTS
02EB  A0 00     VALASC  LDY #0
02ED  B1 EE             LDA (TMPO),Y; LOAD CHAR
02EF  C9 20             CMP #$20
02F1  90 00             BCC PRPER     ; CRY CLR IF <$20
02F3  C9 7F             CMP #$7F
02F5  B0 00             BCS PRPER     : CRY SET IF> = $7F
02F7  60                RTS
02F8  A9 2E     PRPER   LDA#'.        ; LOAD '.'
02FA  60                RTS
02FB                    .END
```

```
 **DEFINED **                ** UNDEFINED **

ASCI        02C8
ASC2        02DD
ASP1        02E6
ASP2        02EA
CRLF        728A
DCMP        70C1
DSPMEM      0200
DSPMEM1     0204
DSPMEM2     020E
DSPMEM3     021B
EXLINE      0242
GETRNG      0248
INCTMP      7397
LCNT        00FF
PRENSP      02B4
PRHDR       0271
PRHDR1      0279
PRLNSP      02A4
PRLNSP1     02AA
PRLNSP2     02B3
PRPER       02F8
PRSPAC      028B
PRSP5       0289
PRVASC      02BD
PRVASP      02E0
RDOA        73A4
RDOC        72E9
RESAD       029B
SAVSAD      0292
SAVX        00FD
SPC         7377
START       7086
TMPO        00EE
TMPO1       00EF
TMP6        00F4
TMP61       00F5
T2T2        7387
VALASC      02EB
WRAP        00E4
WROA        729A
WROB        72B1
WRT         72C6
1F06

EX08 1.2
```

```
                ; "EXDOS" - EXTENDED DICS OPERATING SYSTEM
                ;           FOR 65XX & PERSCI MODEL 277 8"
                ;           DUAL DISC/ 1070 CONTROLLER
                ;
                ;
                ; REVISION 1.1, 9 JAN 78
                ; RSC ASSOCIATES,  FAIRFAX, VA.  22030
                ;
                ; TIM Z-PAGE REGISTERS
                ;
0000            ASCII   = $D1 ; FLAG FOR ASCII DISPLAY, ON = 00
0000            ECHO    = $D6 ; ECHO = 00
0000            ECHOPT  = $D7 ; ECHO FLAG
0000            SAVXR   = $D8
0000            NULLS   = $E3
0000            MAJORT  = $EA
0000            MINORT  = $EB
0000            TMP1LO  = $EE
```

```
0000                    TMP1HI   = $EF
0000                    TMP3LO   = $F2
0000                    TMP3HI   = $F3
0000                    TMP4LO   = $F4
0000                    TMP4HI   = $F5
0000                    LNCNT    = $FF
                        ;
                        ; TIM S/R'S
                        ;
0000                    PR1CHR   = $72C6
0000                    PRCRLF   = $728A
0000                    PR1SPC   = $7377
0000                    RD1ADR   = $73A4
0000                    T2T2     = $7387
0000                    INCTM1   = $7397
0000                    RESTART  = $7086
0000                    PR1BYT   = $72B1
0000                    PR1ADR   = $729A
0000                    DBLSUB   = $70C1
                        ;
                        ; EXDOS CONSTANTS
                        ;
0000                    SOH      = $01  ; READ FROM DISC
0000                    EOT      = $04  ; DONE
0000                    ENQ      = $05  ; WRITE TO DISC
0000                    ACK      = $06  ; PRECEEDS DONE (EOT)
0000                    NAK      = $15  ; FATAL ERROR (MSG FOLLOWS)
0000                    RS       = $1E  ; RELATIVE MODE RECORD SEPARATOR
0000                    STRG     = $0100; COMMAND BUFFER
0000                    DDATA    = $6000; DISC CONTROLLER DATA REG
0000                    DSTAT    = $6001 ; DISC CONTROLLER STATUS REG
                        ;
0000                            .ORG $1200
                        ;
                        ;   **** ENTRY VECTORS ****
                        ;
1200  4C 0212         ENTER   JMP ENTRY
1203  4C 0212         RENTER  JMP RENTRY
                        ;
                        ;   **** MAIN COMMAND LOOP - DRIVE ****
                        ;
                        ;   ENTER FROM EXOS 'DISC' CMD & RETURN TO EXOS
                        ;   WHEN COMPLETE
                        ;
1206  D8               DRIVE   CLD
1207  20 0212                   JSR INPLN    ; INPUT LINE FROM KEYBOARD
120A  20 0212                   JSR DLINE    ; READ & SEND LINE TO DISC
120D  A9 04            DEOT     LDA #EOT     ; SEND EOT TO DISC AT END OF CMD
120F  20 0212                   JSR DOUTC    ; AS A CONTROL BYTE
1212  20 0212           DGET     JSR DINP     ; INPUT BYTE FROM DISC FOR STATUS
1215  B0 00                     BCS DCTRL2   ; WAS CONTROL BYTE
1217  20 C672           PRDATA   JSR PR1CHR   ; ELSE WAS DATA, SO ECHO
121A  4C 1212                   JMP DGET     ; & LOOP FOR NEXT BYTE
121D  4C 0212           DCTRL2   JMP DCTRL
                        ;
                        ;   **** INPUT LINE - INPLN ****
                        ;
                        ;   INPUT LINE TO (BUFFER) STRG. TEST FOR BKSPC, ESC
                        ;   DELETE & CR. ALSO TEST FOR READ/WRITE CMD
                        ;
1220  20 0212           INPLN    JSR PRSTGC
1223  455844                     .BYTE 'EXDOS 1.1',4
122D  20 0212                   JSR PRCRLN
1230  20 0212                   JSR PRSTRG
1233  434D44                   .BYTE 'CMD-> ',4
123A  A2 00                     LDX #0
123C  86 D8            INPLN1   STX SAVXR    ; STO X
123E  20 0212                   JSR RD1CHR
1241  A6 D8                     LDX SAVXR    ; RCL X
1243  C9 08                     CMP #8       ; TEST FOR BACK SPACE
1245  D0 00                     BNE INPLN2   ; WAS NOT BACK-SPACE
1247  CA                       DEX          ; WAS BACK-SPACE, SO DELETE CHAR
1248  10 F2                    BPL INPLN1   ; OK IF PLUS, GET NEW CHAR
124A  30 D4                    BMI INPLN    ; RESET X IF MINUS
124C  C9 5C            INPLN2   CMP #$5C     ; TEST FOR CANCEL
124E  D0 00                     BNE INPLN4   ; NOT CANCEL, CONTINUE
1250  F0 CE            INPLN3   BEQ INPLN    ; WAS CANCEL, RESET LINE
1252  C9 1B            INPLN4   CMP #$1B     ; TEST FOR ESCAPE
1254  D0 00                     BNE INPLN5   ; NOT EXCAPE, SO CONTINUE
1256  4C 0212                   JMP RESET
1259  20 0212           DISCX    JSR PRSTGC   ; CRLF & PRINT STRING
125C  444953                     .BYTE 'DISC READY',7,4
1268  20 0212                   JSR PRCRLN
126B  4C 2012                   JMP INPLN    ; RE-ENTER EXDOS
                        ;
126E  C9 0D            INPLN5   CMP #$0D     ; TEST FOR CR, DONE ?
1270  D0 00                     BNE INPLN7   ; IF NOT CR, STO CHAR
1272  A9 04                     LDA #EOT
1274  9D 0001                  STA STRG,X   ; PUT EOT IN BUFFER AS MARK
1277  AD 0001                  LDA STRG     ; TEST CMD AT STRG FOR READ/WRITE
127A  C9 52                     CMP #'R      ; READ CMDS
127C  F0 00                     BEQ TARGET   ; READ FILE FROM DISC TO RAM
127E  C9 4C                     CMP #'L      ; LOAD (READ FILE)
1280  F0 00                     BEQ TARGET
1282  C9 49                     CMP #'I      ; INPUT (READ SECTOR)
1284  F0 00                     BEQ TARGET
1286  C9 57                     CMP #'W      ; WRITE CMDS
1288  F0 00                     BEQ RANGE    ; GET SA - EA ADR RAM TO DISC
128A  C9 53                     CMP #'S      ; SAVE (WRITE FILE)
128C  F0 00                     BEQ RANGE
128E  C9 4F                     CMP #'O      ; OUTPUT (WRITE SECTOR)
1290  D0 00                     BNE INPLN6   ; SKIP IF NOT R/W CMD
                        ;
1292  20 0212           RANGE    JSR PRCRLN
1295  4C 0212                   JMP ADRNG    ; GET SA - EA & RETURN
1298  20 0212           TARGET   JSR PRCRLN
129B  4C 0212                   JMP GETADR   ; GET TA & RETURN
                        ;
129E  20 0212           INPLN6   JSR PRCRLN
12A1  60                        RTS          ; RETURN
                        ;
12A2  9D 0001           INPLN7   STA STRG,X   ; STO IN STRG BUFFER
12A5  E8                        INX          ; INC CHAR COUNTER
12A6  E0 20                     CPX #32      ; 32 CHAR MAX LINE
12A8  F0 A6                     BEQ INPLN3   ; IF 32 RESTART LINE
12AA  D0 90                     BNE INPLN1   ; ELSE GET NEXT CHAR
                        ;
                        ;   **** SEND LINE TO DISC - DLINE ****
                        ;
                        ;   READ LINE (CHAR) FROM BUFFER (STRG) &
                        ;   SEND TO DISC CONTROLLER FOR PROCESS
                        ;
12AC  A2 00             DLINE    LDX #0       ; INIT AT STRG,0
12AE  BD 0001           DLINE1   LDA STRG,X   ; GET CHAR FROM BUFFER
12B1  C9 04                     CMP #EOT     ; EOL, DONE?
12B3  F0 00                     BEQ DLINEX   ; EXIT IF DONE
12B5  20 0212                   JSR DOUTW    ; ELSE SEND CHAR TO DISC WHEN READY
12B8  E8                        INX          ; INC BUFFER POINTER
12B9  D0 F3                     BNE DLINE1   ; LOOP FOR NEXT CHAR UNTIL EOT
12BB  60                DLINEX   RTS
                        ;
                        ;   **** TEST FOR CONTROL CHAR - DCTRL ****
```

```
                        ;
                        ;   CHECKS DDATA FOR CONTROL BYTE (WHEN CRY SET)
                        ;
12BC  C9 04     DCTRL   CMP #EOT    ; IS DISC FINISHED ?
12BE  F0 99             BEQ DISCX   ; IF SO, EXIT TO EXOS
12C0  C9 01             CMP #SOH    ; READ PROTOCOL (I,L,R CMDS) ?
12C2  F0 00             BEQ DREAD   ; DO DISC READ TO RAM
12C4  C9 05             CMP #ENQ    ; WRITE PROTOCOL (O,S,W CMDS) ?
12C6  F0 00             BEQ DWRIT   ; DO DISC WRITE FROM RAM
12C8  C9 15             CMP #NAK    ; FATAL ERROR
12CA  D0 00             BNE DCTRL1  ; NOT FATAL ERROR, NEXT DISC BYTE
12CC  20 0212           JSR PRSTGC  ; FATAL ERROR
12CF  464154            .BYTE 'FATAL ERROR - ',7,4
12DF  4C 1212   DCTRL1  JMP DGET    ; FROM DISC CONTROLLER UNTIL EOT
                        ;
                        ;   **** INPUT BYTE FROM DISC CONTROLLER - DINP ****
                        ;
12E2  AD 0160   DINP    LDA DSTAT   ; TEST DISC STATUS
12E5  29 C0             AND #$C0    ; DISC STATUS AVAIL (CLR BITS 1-5)
12E7  F0 F9             BEQ DINP    ; NO, LOOP UNTIL AVAIL
12E9  2A                ROL A       ; BIT 7 -> CRY (SET IF CONTROL)
12EA  AD 0060           LDA DDATA   ; GET DISC BYTE
12ED  60                RTS
                        ;
                        ;   **** WAITS FOR DISC DATA - DOUTW ****
                        ;
                        ;   WAITS FOR DISC XMIT BUFFER TO EMPTY & READY FOR
                        ;   NEXT BYTE. ARBITRATES IF DISC & HOST CPU TRY TO
                        ;   XMIT TO EACH OTHER AT SAME TIME.
                        ;
12EE  48        DOUTW   PHA         ; SAVE BYTE TO XMIT TO DISC
12EF  AD 0160   DOUTW1  LDA DSTAT   ; DISC STATUS
12F2  29 C0             AND #$C0    ; IS DISC XMITTING (DATA AVAIL) ?
12F4  D0 00             BNE DOUTX   ; YES, BREAK TIE IN FAVOR OF HOST CPU
12F6  AD 0160           LDA DSTAT   ; NO, GET DISC STATUS AGAIN
12F9  29 03             AND #$03    ; CAN DISC RECEIVE (BUFFER EMPTY) ?
12FB  D0 F2             BNE DOUTW1  ; NOT YET, LOOP TIL BITS 0,1 CLR
12FD  68        DOUTX   PLA         ; GET CPU BYTE
12FE  8D 0060           STA DDATA   ; SEND DATA TO DISC
1301  60                RTS
                        ;
                        ;   **** OUTPUT CPU CONTROL BYTE TO DISC - DOUTC ****
                        ;
1302  20 EE12   DOUTC   JSR DOUTW   ; WAIT FOR DISC READY & OUTPUT DATA
1305  8D 0160           STA DSTAT   ; ALSO OUTPUT CONTROL TO STATUS REG
1308  60                RTS
                        ;
                        ;   **** WRITE TO DISC FROM RAM (TMP1LO) - DWRIT ****
                        ;
1309  20 E212   DWRIT   JSR DINP    ; GET NEXT BYTE AFTER ENQ (WRITE)
130C  C9 04             CMP #EOT    ; MUST BE EOT
130E  D0 00             BNE DWRITX  ; ELSE ERROR, SO EXIT
1310  A0 00     DWRIL   LDY #0
1312  D1 EE             LDA (TMP1LO),Y ; GET DATA FROM RAM
1314  20 EE12           JSR DOUTW   ; SEND TO DISC WHEN READY
1317  20 9773           JSR INCTM1  ; INC ADR
131A  20 C170           JSR DBLSUB  ; UNTIL DONE (CRY CLR)
131D  B0 F1             BCS DWRIL   ; LOOP TILL SA = EA
131F  4C 0D12           JMP DEOT    ; DONE, SO SEND EOT TO DISC
1322  4C 5912   DWRITX  JMP DISCX   ; EXIT TO EXOS
                        ;
                        ;   **** READ FROM DISC TO RAM (TMP3LO) - DREAD ****
                        ;
1325  20 E212   DREAD   JSR DINP    ; GET DATA AFTER SOH (READ)
1328  B0 92             BCS DCTRL   ; WAS IT CONTROL ?
132A  A0 00             LDY #0      ; NO, SO MOVE TO RAM
132C  91 F2             STA (TMP3LO),Y ; STO DATA TO RAM
132E  20 0213           JSR INCTM3  ; POINT TO NEXT RAM LOC TO STO
1331  4C 2513           JMP DREAD   ; LOOP TILL CONTROL (EOT, NAK)
                        ;
1334  E6 F2     INCTM3  INC TMP3LO
1336  D0 00             BNE TMPX
1338  E6 F3             INC TMP3HI
133A  60        TMPX    RTS
                        ;
                        ; END OF EXDOS FILE #01
                        ;
133B  A2 20     RESET   LDX #32
133D  A9 00     RES1    LDA #0
133F  9D 0001           STA STRG,X
1342  CA                DEX
1343  10 F8             BPL RES1
1345  4C 8670           JMP RESTART
                        ;
1348            DELAY1  = $7320
1348            RD1BYT  = $73B3
1348            SEREG   = $6E02 ; TIM SERIAL REG
1348            SAVXR3  = $DE
1348            SAVYR3  = $DF
1348            AUXPCL  = $E0
1348            AUXPCH  = $E1
                        ;
1348  A5 D1     DISP64  LDA ASCII   ; GET FLAG FOR ASCII DISPLAY
134A  49 80             EOR #$80    ; TOGGLE IT
134C  85 D1             STA ASCII   ; PUT IT BACK, 00 = ON, $80 = OFF
134E  4C 8670           JMP RESTART ; RETURN TO TIM
                        ;
                        ;
                        ; READ ONE CHARACTER FROM TIM SERIAL REG W/ ECHO OPTION
                        ;
                        ;
1351  48        ECHORD  PHA         ; STO A, ECHO CHAR READ
1352  A5 D6             LDA ECHO    ; ECHO IF = 00
1354  D0 00             BNE NOECHO  ; IF NOT = 00, SKIP ECHO
1356  AD 026E           LDA SEREG
1359  29 FD             AND #$FD    ; % 1111 1101 CLR BIT 1
135B  90 00             BCC ECHO1   ; OUTPUT BIT IN CARRY
135D  09 02             ORA #2      ; % 0000 0010 SET BIT 1
135F  8D 026E   ECHO1   STA SEREG
1362  68        NOECHO  PLA         ; RCL A
1363  60                RTS
                        ;
                        ;
                        ;
1364  A2 08     RD1CHR  LDX #8      ; 8 BITS
1366  AD 026E   RDCHR2  LDA SEREG
1369  4A                LSR A       ; TEST START BIT
136A  90 FA             BCC RDCHR2  ; WAIT FOR START BIT
136C  20 2073           JSR DELAY1  ; DELAY 1/2 BIT TIME
136F  20 5113           JSR ECHORD  ; ECHO START BIT ?
1372  20 0213   RDCHR3  JSR RDCHR6  ; DELAY 1 BIT TIME
1375  AD 026E           LDA SEREG
1378  4A                LSR A
1379  20 5113           JSR ECHORD  ; ECHO EACH BIT ?
137C  08                PHP         ; STO BIT
137D  98                TYA         ; CHAR IS IN Y
137E  4A                LSR A
137F  28                PLP         ; RCL BIT
1380  90 00             BCC RDCHR4
1382  09 80             ORA #$80    ; ADD NEXT BIT
1384  A8        RDCHR4  TAY         ; BUILD CHAR IN'Y
1385  CA                DEX         ; 8 TIMES (BITS)
```

```
1386  DO EA             BNE RDCHR3  ; 8 TIMES (BITS)
1388  49 FF             EOR #$FF    ; COMP FINAL CHAR
138A  29 7F             AND #$7F    ; CLR PARITY
138C  20 0213           JSR RDCHR6  ; 1 STOP BIT DELAY
138F  18          RDCHR5 CLC        ; CLR FLAG
1390  4C 5113           JMP ECHORD  ; ECHO STOP BIT
1393  20 2073     RDCHR6 JSR DELAY1  ; DELAY 2 - 1/2 BIT TIMES
1396  4C 2073           JMP DELAY1
                  ;
                  ;
                  ;
                  ; SPEED CHANGE ROUTINE
                  ;
                  ;
1399  20 0213     SPEED  JSR PRSTRG
139C  202031             .BYTE' 110/300/1200/2400 BAUD = LO/MD/HI/XH ? '
13C6  20 6413            JSR RD1CHR  ; READ 1ST CHAR                    ,4
13C9  20 6413            JSR RD1CHR  ; READ 2D CHAR (H,I,D OR O ?)
13CC  C9 48              CMP #'H
13CE  FO 00              BEQ SPDXH
13DO  C9 49              CMP #'I
13D2  FO 00              BEQ SPDHI
13D4  C9 44              CMP #'D
13D6  FO 00              BEQ SPDMD
13D8  C9 4F              CMP #'O
13DA  FO 00              BEQ SPDLO
13DC  DO 00              BNE SPDX
13DE  A9 00       SPDXH  LDA #$00    ; LOOP VALUES FOR 1 MHZ CLOCK
13E0  A2 72              LDX #$72
13E2  DO 00              BNE CHCX
13E4  A9 01       SPDHI  LDA #$01
13E6  A2 3A              LDX #$3A
13E8  DO 00              BNE CHGX
13EA  A9 06       SPDMD  LDA #$06
13EC  A2 IF              LDX #$1F
13EE  DO 00              BNE CHGX
13F0  A9 10       SPDLO  LDA #$10
13F2  A2 E8              LDX #$E8
13F4  85 EA       CHGX   STA MAJORT
13F6  86 EB              STX MINORT
13F8  20 6413            JSR RD1CHR
13FB  20 0213            JSR PRCRLN
13FE  4C 8670     SPDX   JMP RESTART
                  ;
                  ;
                  ;    **** PRINT CR/LF WITH NULLS IN $E3 ****
                  ;
1401  A9 OD       PRCRLN LDA #$OD
1403  20 C672            JSR PR1CHR
1406  A9 OA              LDA #$OA
1408  20 C672            JSR PR1CHR
140B  A4 E3              LDY NULLS
140D  A9 00       PRNUL  LDA #0
140F  20 C672            JSR PR1CHR  ; PRINT NULLS
1412  88                 DEY
1413  DO F8              BNE PRNUL
1415  60                 RTS
                  ;
                  ;
1416                     .ORG $1500
                  ;
1500  D8         DSPMEM CLD
1501  20 0114          JSR PRCRLN
1504  20 0215          JSR ADRNC
1507  20 0114    D1    JSR PRCRLN
150A  20 0215          JSR PRHDR
1500  A2 00            LDX #0
150F  86 FF            STX LNCNT
1511  20 0114    D2    JSR PRCRLN
1514  20 0215          JSR SAVSAD
1517  20 9A72          JSR PR1ADR
151A  20 0215          JSR PRLNSP
151D  20 7773    D3    JSR PR1SPC
1520  AO 00            LDY #0
1522  B1 EE            LDA (TMP1LO),Y
1524  20 B172          JSR PR1BYT
1517  20 9773          JSR INCTM1
152A  20 C170          JSR DBLSUB
152D  90 00            BCC DX
152F  A5 EE            LDA TMP1LO
1531  29 OF            AND #$OF
1533  DO E8            BNE D3
1535  20 0215          JSR PRVASC
1538  20 0215          JSR BREAK
153B  E6 FF            INC LNCNT
153D  A9 10            LDA #16
153F  C5 FF            CMP LNCNT
1541  FO C4            BEQ D1
1543  4C 1115          JMP D2
1546  20 0215    DX    JSR PRENSP
1549  4C 0215          JMP PRVASC
                  ;
154C  20 0114    PRHDR JSR PRCRLN
154F  20 0215          JSR PR5SPC
1552  AO 00            LDY #0
1554  98         HDR1  TYA
1555  20 B172          JSR PR1BYT
1558  20 7773          JSR PR1SPC
155B  C8               INY
155C  CO 10            CPY #16
155E  DO F4            BNE HDR1
1560  20 0114          JSR PRCRLN
1563  60               RTS
                  ;
1564  A2 05      PR5SPC LDX #5
1566  20 7773    PRSPAC JSR PR1SPC
1569  CA               DEX
156A  DO FA            BNE PRSPAC
156C  60               RTS
                  ;
156D  A5 EE      SAVSAD LDA TMP1LO
156F  85 F2            STA TMP3LO
1571  A5 EF            LDA TMP1HI
1573  85 F3            STA TMP3HI
1575  60               RTS
                  ;
1576  A5 F2      RCLSAD LDA TMP3LO
1578  85 EE            STA TMP1LO
157A  A5 F3            LDA TMP3HI
157C  85 EF            STA TMP1HI
157E  60               RTS
                  ;
157F  A5 EE      PRLNSP LDA TMP1LO
1581  29 OF            AND #$OF
1583  FO 00            BEQ LN2
1585  85 D8      LN1   STA SAVXR
1587  OA               ASL A
1588  65 D8            ADC SAVXR
158A  AA               TAX
158B  20 6615          JSR PRSPAC
158E  60         LN2   RTS
```

```
                        ;
158F  A9 11     PRENSP  LDA #$11
1591  E5 EE             SBC TMP1LO
1593  29 0F             AND #$0F
1595  D0 EE             BNE LN1
1597  60                RTS
                        ;
1598  A5 D1     PRVASC  LDA ASCII
159A  D0 00             BNE ASC2
159C  20 7615           JSR RCLSAD
159F  A2 02             LDX #2
15A1  20 6615           JSR PRSPAC
15A4  20 0215           JSR PRVASP
15A7  20 0215     ASC1  JSR VALASC
15AA  20 C672           JSR PR1CHR
15AD  20 9773           JSR INCTM1
15B0  20 C170           JSR DBLSUB
15B3  90 00             BCC ASCX
15B5  A5 EE             LDA TMP1LO
15B7  29 0F             AND #$0F
15B9  D0 EC             BNE ASC1
15BB  20 C170     ASC2  JSR DBLSUB
15BE  90 00             BCC ASCX
15C0  60                RTS
15C1  4C 8670     ASCX  JMP RESTART
                        ;
15C4  A5 EE       PRVASP  LDA TMP1LO
15C6  29 0F               AND #$0F
15C8  F0 00               BEQ ASP2
15CA  AA                  TAX
15CB  20 6615             JSR PRSPAC
15CE  60          ASP2    RTS
                          ;
15CF  A0 00       VALASC  LDY #0
15D1  B1 EE               LDA (TMP1LO),Y
15D3  C9 20               CMP #$20
15D5  90 00               BCC PRPER
15D7  C9 7F               CMP #$7F
15D9  B0 00               BCS PRPER
15DB  60                  RTS
15DC  A9 2E       PRPER   LDA #'.
15DE  60                  RTS
                          ;
15DF  A9 01       BREAK   LDA #1
15E1  2C 026E             BIT SEREG
15E4  D0 00               BNE NOKEY
15E6  60                  RTS
15E7  2C 026E     NOKEY   BIT SEREG
15EA  D0 FB               BNE NOKEY
15EC  A2 FF               LDX #$FF
15EE  9A                  TXS
15EF  4C 8670             JMP RESTART
                          ;
15F2  20 0114     PRSTGC  JSR PRCRLN
15F5  20 0215     PRSTRG  JSR STOXY3
15F8  18          NXTASC  CLC
15F9  68                  PLA
15FA  69 01               ADC #1
15FC  85 E0               STA AUXPCL
15FE  68                  PLA
15FF  69 00               ADC #0
1601  48                  PHA
1602  85 E1               STA AUXPCH
1604  A5 E0               LDA AUXPCL
1606  48                  PHA
1607  A0 00               LDY #0
1609  B1 E0               LDA (AUXPCL),Y
160B  C9 04               CMP #4
160D  F0 00               BEQ STRGX
160F  20 C672             JSR PR1CHR
1612  18                  CLC
1613  90 E3               BCC NXTASC
1615  20 0216     STRGX   JSR RCLXY3
1618  60                  RTS
                          ;
1619  86 DE       STOXY3  STX SAVXR3
161B  84 DF               STY SAVYR3
161D  60                  RTS
                          ;
161E  A6 DE       RCLXY3  LDX SAVXR3
1620  A4 DF               LDY SAVYR3
1622  60                  RTS
                          ;
1623  20 F515     ANRNG   JSR PRSTRG
1626  202020              .BYTE '      START-END ADR $',4
163C  20 A473             JSR RD1ADR
163F  20 8773             JSR T2T2
1624  20 F515             JSR PRSTRG
1645  202404              .BYTE ' $',4
1648  20 A473             JSR RD1ADR
164B  20 8773             JSR T2T2
164E  20 6413     RNGX    JSR RD1CHR
1651  20 0114             JSR PRCRLN
1654  60                  RTS
                          ;
1655  20 F515     GETADR  JSR PRSTRG
1658  202020              .BYTE '      TARGET ADR $',4
166B  20 B373             JSR RD1BYT
166E  85 F3               STA TMP3HI
1670  20 B373             JSR RD1BYT
1673  85 F2               STA TMP3LO
1675  4C 4E16             JMP RNGX
                          ;
                          ;   **** INIT ROUTINES ****
                          ;
1678  D8          ENTRY   CLD
1679  A9 00               LDA #0
167B  85 D6               STA ECHO     ; TURN ECHO ON
167D  85 D1               STA ASCII    ; ASCII ON
167F  A9 02               LDA #2
1681  85 E3               STA NULLS
1683  4C 9913             JMP SPEED
                          ;
1686  4C 0612     RENTRY  JMP DRIVE
                          ;
                          ; END OF EXDOS FILE #02
```

Context Switch for the 6800

M. MICKELSEN AND N. WORTH
Becton Dickinson Immunodiagnostics
180 W. 2950 South
Salt Lake City, Utah 84115

The software interrupt on the Motorola 6800-6802 can be used to save all the registers and flags upon entry to a subroutine, and a return from interrupt can be used to restore them and return from the subroutine. The method presented here is relatively fast, performing the context switch in 107 machine cycles.

The interrupt system in the 6800 places all the registers and flags on the stack. This is the key to the context switch routine. The first instruction of the subroutine performs a software interrupt. The interrupt handler then duplicates the 7 top values on the stack. It also changes the return address of the bottom set of values to point to a fixed subroutine return instruction.

The interrupt handler ends with a return from interrupt instruction. This pops the last 7 values off the stack—the ones the interrupt handler put there—and returns control to the subroutine. The registers appear to be unchanged. Since the registers and flags are now also on the stack, the programmer is now free to alter them in any way within the subroutine. He can also access the original values easily on the stack.

The subroutine return is made using a return from interrupt instruction instead of the usual return from subroutine. The return from interrupt pops the top 7 values off the stack, restoring the flags and registers. The software interrupt routine altered the program counter value in this group to point to a fixed return from subroutine instruction. When the stack is popped, this instruction is the next to be executed. It pops the true return address off the stack and gives control back to the calling program. The flags and registers have been restored to the values they had when the software interrupt was made.

The accompanying listings show the software interrupt routine and an example of its use. The interrupt routine is labeled SWIR. It is possible to make this routine smaller by using a loop when duplicating the stack values, but that is a bit slower than the version shown. The subroutine, ZABX, adds the B accumulator to the index register.

An efficient context switching mechanism is essential when working with operating systems and real time control. The method shown here, although not ideal, has proven useful and sufficiently fast for these applications.

```
  6800 Cross Assembler, version 1.05-01R
Err Loc.  Data     Seq.          Source
                      1 *
                      2 *       SWIR    SOFTWARE INTERRUPT ROUTINE
                      3 *               SAVES REGISTERS AND STATUS ON STACK
                      4 *               PROVIDES FOR SUBROUTINE ROUTINE VIA RTI TO
                      5 *               RESTORE THE REGISTERS
                      6 *               (COPYRIGHT 1979 BECTON-DICKINSON IMMUNODIAGNOSTICS)
                      7 *
    0000'30           8 SWIR    TSX             STACK POINTER TO INDEX
    0001'A6 06        9         LDA A   6,X     DUPLICATE REGISTER SAVE AREA
    0003'36          10         PSH A
    0004'A6 05       11         LDA A   5,X
    0006'36          12         PSH A
    0007'A6 04       13         LDA A   4,X
    0009'36          14         PSH A
    000A'A6 03       15         LDA A   3,X
    000C'36          16         PSH A
    000D'A6 02       17         LDA A   2,X
    000F'36          18         PSH A
    0010'A6 01       19         LDA A   1,X
    0012'36          20         PSH A
    0013'A6 00       21         LDA A   0,X
    0015'36          22         PSH A
    0016'B6 0021'    23         LDA A   ZRES    SET UP SUBROUTINE RETURN AS PART OF RTI
    0019'A7 05       24         STA A   5,X
    001B'B6 0022'    25         LDA A   ZRES+1
    001E'A7 06       26         STA A   6,X
    0020'3B          27         RTI             RETURN FROM SOFTWARE INTERRUPT WITH STACK SET
                     28 *                          FOR RESTORING REGISTERS WITH ANOTHER RTI
    0021'0023'       29 ZRES    FDB     *+2
    0023'39          30         RTS             SUBROUTINE RETURN TO RESET STACK
                     31 *
                     33 *
                     34 *       ZABX            ADD B ACCUMULATOR TO INDEX REGISTER
                     35 *               (COPYRIGHT 1979 BECTON-DICKINSON IMMUNODIAGNOSTICS)
                     36 *
    0024'3F          37 ZABX    SWI             SAVE REGISTERS ON STACK
    0025'30          38         TSX             STACK POINTER TO INDEX
    0026'EB 04       39         ADD B   4,X     B + LOW ORDER X VALUE
    0028'24 02       40         BCC     ZABXF   NO CARRY
    002A'6C 03       41         INC     3,X     CARRY - BUMP HIGH ORDER X VALUE
    002C'E7 04       42 ZABXF   STA B   4,X
    002E'3B          43         RTI             RETURN, RESTORING REGISTERS AND CHANGED X
                     44 *
                     45         END
```

Diskette Copying on Intel Single Density DOS

BY BILL COLLIS
Systems Programming Ltd. International
12-14 Windmill Street
London W1P 1HF

As soon as I saw Paul Julich's diskette copy program in the November-December 1978 issue of *Dr. Dobb's* (*DDJ* # 30), I decided to convert it for Single Density DOS. The program presented here is much smaller and simpler and, I hope, easier to understand. It also runs 2 minutes faster since a single density diskette contains less data.

Contrary to Julich's assertions, the size of the parameter block (IOPB) is the same for both single and double density DOS. The difference lies in the content. Obviously the number of sectors per track must be halved from 52 (35H) to 26. Also, the single density controller sets bit 0 of the first byte of the IOPB, locking out the IOPB from further activation. This feature can be overridden by setting bit 7, as I have done.

With these modifications the original program still aborted on the frequent but recoverable soft I/O errors. I therefore rewrote the entire program to use a general purpose execution routine, EXEC, which repeated the I/O operation after a message in case of error. The fourth consecutive error would return control to monitor so registers D and E could be examined to determine the cause. For a detailed explanation of these errors the reader should refer to the Intel MDS-DOS Hardware Reference Manual. In practice, however, I found that the errors almost always occurred on the read operation of Drive 0 and were not repeated a second time. I suspect that the track and Drive change operation causes some vibration and disc head misalignment. The second attempt to read is successful because the vibration has died down.

Having written a general purpose I/O routine like EXEC, I was able to use it to perform other I/O operations such as FORMAT (initializing a track) and VERIFY, using the same IOPB. I have now removed the FORMAT facility from the program because, although it actually increased the program speed, ISIS disc operations were considerably retarded. This is because a FORMAT operation using the IOPB as written will order the sectors sequentially. ISIS does not perform any multisector operations, but reads one sector at a time. By the time ISIS is ready to operate on the next sequence sector, the sector will have already passed by the head if it physically follows the previous logical sector. Thus it would take 26 rotations of the disc to read a track, which would be rather slow!

Operation of the program is identical to the double density version. The disk to be copied is placed in Drive 0 and a formatted disk (which will be completely overwritten) is placed in Drive 1. If a disk contains the program, then execution can be initiated in the normal way by typing in the program name at the console. Otherwise the program must first be loaded into memory using DEBUG before program execution.

```
ASM80 QUICK


ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0
INTEL MDS PHYSICAL DISKETTE COPY

  LOC  OBJ          SEQ         SOURCE STATEMENT

                      1 $NOSYMBOLS TITLE('INTEL MDS PHYSICAL DISKETTE COPY')
                      2                           ; DEFINE DISKETTE OPERATIONS
  0001                3 SEEK    EQU   1           ; MOVE HEAD TO TRACK SPECIFIED AND CHECK TRACK NUMBER
  0002                4 FORMAT  EQU   2
  0003                5 RECAL   EQU   3           ; SEEK TO TRACK 0
  0004                6 READ    EQU   4
  0005                7 VERIFY  EQU   5
  0006                8 WRITE   EQU   6
  0007                9 DELETE  EQU   7           ; WRITE RECORD(S) WITH DELETED DATA MARK(S)
                     10
  0000               11 DRIVE0  EQU   0           ; DEFINE DISKETTE DRIVES
  0030               12 DRIVE1  EQU   30H
                     13
                     14         ASEG
  4000               15         ORG   4000H
                     16
  4000 329140        17 EXEC:   STA   IOPB+1     ; STORE DISKETTE COMMAND IN IOPB
  4003 2E04          18 RETRY:  MVI   L,4         ; ALLOW 4 I/O ERRORS
  4005 119040        19 AGAIN:  LXI   D,IOPB     ; LOAD ADDRESS OF IOPB
  4008 7B            20         MOV   A,E
  4009 D379          21         OUT   79H        ; OUTPUT LSB OF IOPB ADDRESS
  400B 7A            22         MOV   A,D
  400C D37A          23         OUT   7AH        ; OUTPUT MSB OF IOPB ADDRESS AND EXECUTE
  400E E3            24         XTHL
  400F E3            25         XTHL             ; WASTE TIME WHILE CONTROLLER SETS INCOMPLETE STATUS
  4010 DB78          26 NOTRDY: IN    78H        ; CHECK COMPLETION STATUS
  4012 E604          27         ANI   4          ; WAIT FOR DISKETTE CONTROLLER TO SET BIT 2
  4014 CA1040        28         JZ    NOTRDY
  4017 DB79          29         IN    79H        ; GET RESULT TYPE AND IGNORE
  4019 57            30         MOV   D,A
  401A DB7B          31         IN    7BH        ; GET RESULT BYTE (ERROR CODE - SEE HARDWARE MANUAL)
  401C B7            32         ORA   A          ; TEST FOR ERROR
  401D C8            33         RZ               ; RETURN WITH Z SET-) NO ERROR
  401E 5F            34         MOV   E,A        ; SAVE ERROR CODE FOR INSPECTION VIA MONITOR
  401F CD8340        35         CALL  MSGF
  4022 0A            36         DB    0AH,'I/O ERROR',0DH,0AH,0
  4023 492F4F20
  4027 4552524F
  402B 52
  402C 0D
  402D 0A
  402E 00
  402F 2D            37         DCR   L
  4030 C20540        38         JNZ   AGAIN
  4033 C7            39         RST   0          ; CALL MONITOR - (RETURN WITH G COMMAND)
  4034 C30340        40         JMP   RETRY
                     41
                     42 DCPY:                    ; COPY DISKETTE FROM DRIVE 0 TO DRIVE 1
  4037 3E33          43         MVI   A,DRIVE1+RECAL
  4039 CD0040        44         CALL  EXEC       ; EXECUTE RECALIBRATE TO TRACK 0 ON DRIVE 1
  403C 3E03          45         MVI   A,DRIVE0+RECAL
  403E CD0040        46         CALL  EXEC       ; EXECUTE RECALIBRATE TO TRACK 0 ON DRIVE 0
  4041 3E04          47 TLOOP:  MVI   A,DRIVE0+READ
  4043 CD0040        48         CALL  EXEC       ; EXECUTE READ ON DRIVE 0
  4046 CD8340        49         CALL  MSGF
  4049 54524143      50         DB    'TRACK',0
  404D 4B
  404E 00
  404F 3E36          51         MVI   A,DRIVE1+WRITE
  4051 CD0040        52         CALL  EXEC       ; EXECUTE WRITE ON DRIVE 1
  4054 CD8340        53         CALL  MSGF
  4057 20575249      54         DB    ' WRITTEN',0
  405B 5454454E
  405F 00
  4060 3E35          55         MVI   A,DRIVE1+VERIFY
  4062 CD0040        56         CALL  EXEC       ; EXECUTE VERIFY ON DRIVE 1
  4065 CD8340        57         CALL  MSGF
  4068 20414E44      58         DB    ' AND VERIFIED',0AH,0DH,0
  406C 20564552
  4070 49464945
  4074 44
  4075 0A
  4076 0D
  4077 00
  4078 219340        59         LXI   H,TRACK
  407B 34            60         INR   M          ; INCREMENT TRACK ADDRESS
  407C 3E4C          61         MVI   A,76       ; GREATER THAN MAXIMUM?
  407E BE            62         CMP   M
  407F D24140        63         JNC   TLOOP
  4082 CF            64         RST   1          ; COPY COMPLETE - RETURN TO ISIS
                     65
  4083 E3            66 MSGF:   XTHL             ; PRINT MESSAGE POINTED TO BY PC ON CRT
  4084 4E            67         MOV   C,M
  4085 23            68         INX   H
  4086 E3            69         XTHL             ; SWAP HL FOR RETURN ADDRESS AGAIN
  4087 0C            70         INR   C
  4088 0D            71         DCR   C          ; TEST C
  4089 C8            72         RZ               ; RETURN IF END OF STRING
  408A CD09F8        73         CALL  0F809H     ; CALL MONITOR TO PRINT REGISTER C
  408D C38340        74         JMP   MSGF
                     75
                     76
  4090 80            77 IOPB:   DB    80H        ; LOCK OVER RIDE
  0001               78         DS    1          ; DISKETTE COMMAND
  4092 1A            79         DB    26         ; NUMBER OF SECTORS TO BE TRANSFERED
  4093 00            80 TRACK:  DB    0          ; TRACK NUMBER
  4094 01            81         DB    1          ; START SECTOR NUMBER
  4095 0060          82         DW    6000H      ; BUFFER ADDRESS
  4037               83         END   DCPY

ASSEMBLY COMPLETE,  NO ERRORS
```

end

Hardware Relative Addressing for the 8080

BY JOHN BEETEM
856 Allardice Way
Stanford, CA 94305

Position Independent Code (PIC) is code (instructions + data) which looks and executes the same no matter where it is loaded into a computer's memory. It does not have to be reassembled to be loaded somewhere else. Some of the advantages of PIC are:

1. If the computer's memory configuration changes, the software does not have to be changed.
2. Software can be run on systems with memory starting at 0000H (e.g. Altair) *and* systems with memory starting at 2000H (e.g. North Star) without changing the software: it is *system independent*.
3. Large programs can be assembled in modules which are later easily linked together.
4. Operating systems can move programs around in memory without changing them.
5. Debugging routines and device drivers can be appended to the end of programs without reassembly.

[Systems software (e.g., CP/M) should always be distributed as PIC. Unfortunately, practically nobody does.]

PIC looks like a good thing to have. Unfortunately, the Intel 8080, the industry's most popular microprocessor, does not have built-in facilities for PIC. There are some software methods for simulating PIC [1, 2], but these are cumbersome and inefficient. This article describes a simple, efficient hardware modification to allow PIC for the 8080 (it could be modified for the 8085 or Z-80).

Relative Addressing

PIC can be implemented using relative addressing. Consider the "JMP X" instruction on the 8080. JMP is a three byte instruction; the first byte is C3H (H=Hexadecimal) and the next two bytes are the low and high bytes of X, the absolute address to which to jump (assume X is in the same program as the JMP X instruction). This is not a position independent instruction, because if the entire program is moved, X is moved, so the code generated for the JMP X instruction must be changed (Fig. 1).

Figure 1. Position Dependent Code

Fig. 1A. Assembler listing with "JMP X" assembled at 1000H. Let X be at 2000H.

```
1000          C3 00 20          JMP          X
 .              .                .
 .              .                .
 .              .                .
2000                      X:
```

Figure 1B. Assembler listing after the entire program has been moved 1000H bytes. JMP X is at 2000H and X is at 3000H.

```
2000          C3 00 30          JMP          X
 .              .                .
 .              .                .
 .              .                .
3000                      X:
```

Note that the code for JMP X has changed.

Now, suppose we had a "relative jump" instruction JR X, where the first byte is the opcode and the next two bytes are the low and high bytes of the "offset" (or "distance") to X. This offset is equal to X-*, where * is the address of the first byte of the JR instruction (* or some equivalent symbol is called the "location counter" and exists in most assemblers). On execution, the offset X-* is added to PC (the Program Counter) to give the "effective address" to which to jump. But PC will be equal to *, so X-* + PC=X, so the 8080 will jump to X, which is what we want.

If we move the entire program, X will be changed. However, * is changed by the same amount. Thus X-* is unchanged and the code generated for the JR X instruction is unchanged. (Fig. 2). Hence JR X is Position Independent.

Fig. 2. Position Independent Code

Fig. 2A. Assembler listing with "JR X" assembled at 1000H. Let X be at 2000H. The offset to X is 1000H.

```
1000            XX 00 10        JR          X
  .               .               .
  .               .               .
  .               .               .
2000                        X:
```

Fig. 2B. Assembler listing after the entire program has been moved 1000H bytes. JR X is at 2000H and X is at 3000H. The offset to X is still 1000H.

```
2000            XX 00 10        JR          X
  .               .               .
  .               .               .
  .               .               .
3000                        X:
```

Note that the code for JR X is unchanged.

To achieve PIC, all we need are relative versions of all the three byte 8080 instructions: JMP, CALL, LDA, STA, LHLD, SHLD, and LXI. [LXI H,0000H is interesting . . . it loads HL with PC.] Unfortunately, relative versions do not exist, and there is no way to change the instruction set. Or is there?

Implementation

We can implement relative instructions by modifying instructions as they are read from memory. This is done as follows:

1. If an instruction is to be relative, it is preceded by a one byte "prefix." This prefix is an instruction which is never used other than for prefixing relative instructions. It is executed by the 8080, so it must not change any registers or condition codes. NOP (00H) would work, but it is used elsewhere in programs. However, Intel was nice enough to give seven other NOP equivalents, namely: MOV A,A; MOV B,B; MOV C,C; . . . MOV L,L. Any one of these could be used for the prefix. I chose MOV B,B (40H.) Remember, MOV B,B may not be used except as a prefix! (A smart assembler could transform non-prefix MOV B,B into MOV A,A.)

Adding the prefix to each relative instruction increases the size of a program and slows it down. However, the only prefixed instructions are three-byte instructions, which are already long. The worst case loss of efficiency is only 25%. Typical losses are much less. Note that if a program has no relative addressing it is unchanged, and runs at the same speed as before. (We have not degraded performance.)

After detection of the prefix, the next byte read from memory is the opcode of a three-byte instruction. When this byte is fetched, its address (the current PC) is stored in a latch.

The next two bytes read from memory are the low and high bytes of the offset. This offset is not sent directly to the 8080. Instead, it is added to the stored PC to give an effective address, which is sent to the 8080. Note that this sum is sent one byte at a time. Thus, the offset is added to the address of the instruction to give the actual address; we have relative addressing.

Software Considerations

Every relative instruction must be prefixed with MOV B, B (or equivalent prefix). This must be done either by hand or by using a MACRO assembler; e.g., a JR instruction might be defined by the code:

```
.MACRO      JR X      ; Define Relative Jump Macro
MOV         B,B       ; Prefix
JMP         X-*
.ENDM                 ; End of macro
```

The programmer can use JR without worrying about the details of the relative addressing. As an example:

```
Y:    JR      Y
```

is expanded into:

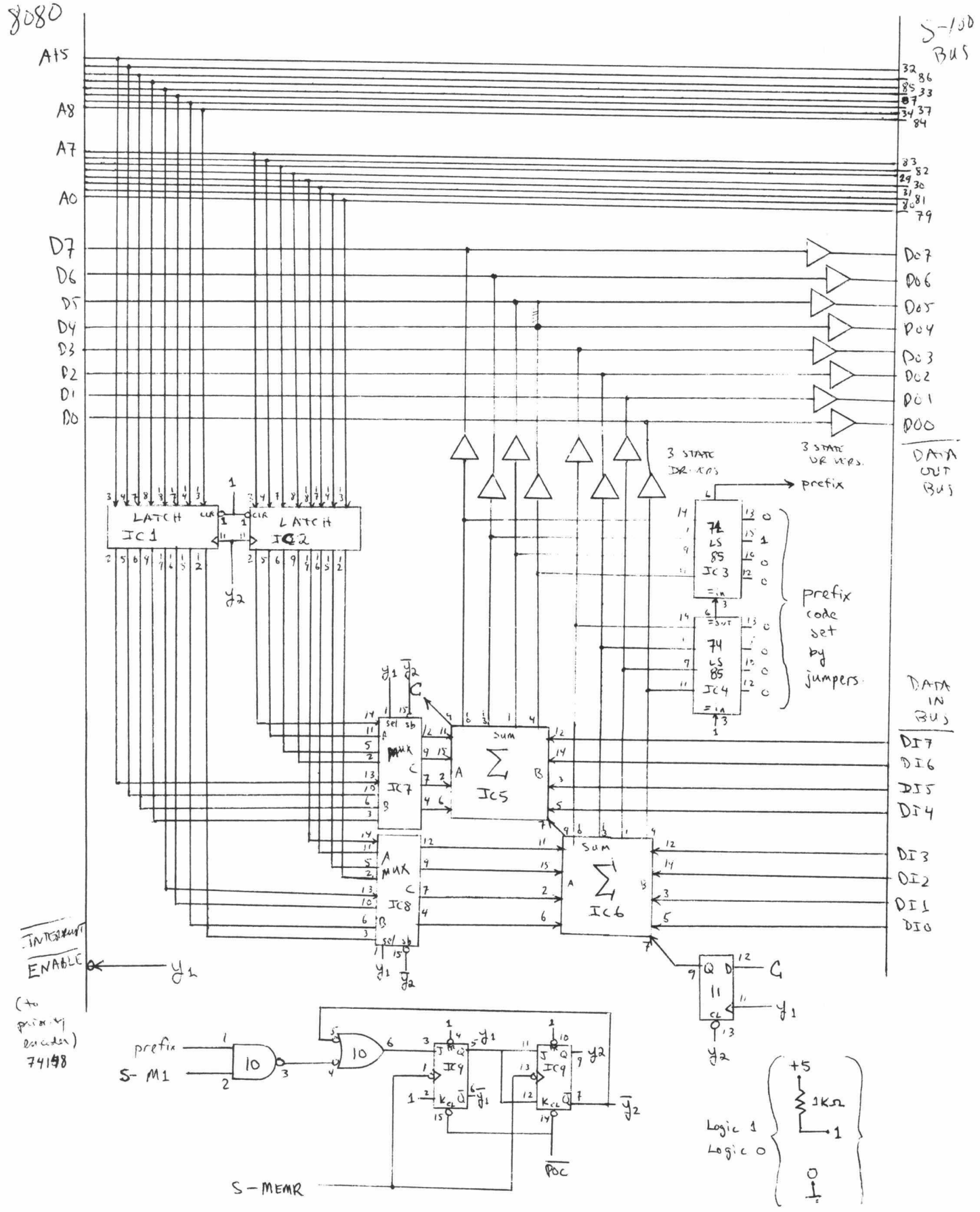
```
Y:    MOVE    B,B
      JMP     -1
```

This is a simple infinite loop, written in Position Independent Code.

The Hardware

Fig. 3 is the circuit diagram of the relative address modification to a CPU board. A15-A0 are the address lines from the CPU (Buffered). D7-D0 are the bidirectional data lines to/from the 8080. They are buffered and sent to the single-direction DO (Data Out) lines on the S-100 bus. The sum of the S-100 DI (Data In) lines and the outputs of multiplexers IC7 and IC8 are sent to the 8080 through tri-state drivers.

The circuit is controlled by a finite state machine consisting of IC9 (J-K Flip Flops) and IC10 (NAND gates). The machine has four states: 0, 1, 2, and 3 which are encoded into two bits, y_1 and y_2 (Fig. 4). State 0 is the rest state, which is entered on POC (Power On Clear). The machine stays in state 0 until the prefix instruction is fetched; i.e., the DI lines are 40H and M1 = 1 (M1 is a signal from the status latch on the CPU board, and signifies that the 8080 is fetching the first byte of an instruction). If the prefix is fetched, the machine will cycle through state 1, 2, and 3 and return to state 0,


8080
A15
A8
A7
A0
D7
D6
D5
D4
D3
D2
D1
D0
S-100 BUS
Do7
Do6
Do5
Do4
Do3
Do2
Do1
Do0
DATA OUT BUS
3 STATE DRIVERS
prefix
LATCH IC1
LATCH IC2
74 LS 85 IC3
74 LS 85 IC4
prefix code set by jumpers
DATA IN BUS
DI7
DI6
DI5
DI4
DI3
DI2
DI1
DI0
MUX IC7
MUX IC8
Sum IC5
Sum IC6
INTERRUPT ENABLE
(to priority encoder) 74148
S-M1
S-MEMR
IC9
POC
+5
1kΩ
Logic 1
Logic 0

where it will rest until another prefix instruction is fetched. N.B.: The machine changes state at the end of the MEMR (Memory Read) pulse from the CPU. This is the time when the data bus has settled down to its correct value. (See waveforms, Fig. 5).

Figure 4. Machine state encoding

STATE	Y2	Y1
0	0	0
1	0	1
2	1	0
3	1	1

Detailed State Description

State 0: If the prefix instruction is fetched, then state 1 will be entered at the end of the MEMR pulse. Otherwise (DI is not 40H *or* M1=0) the machine stays in state 0.

State 1: The 8080 fetches the first byte of the three-byte "relative instruction." During the fetch, the latch (IC1 and IC2) is loaded from the Address lines, which contain PC. At the end of the MEMR pulse, state 2 is entered.

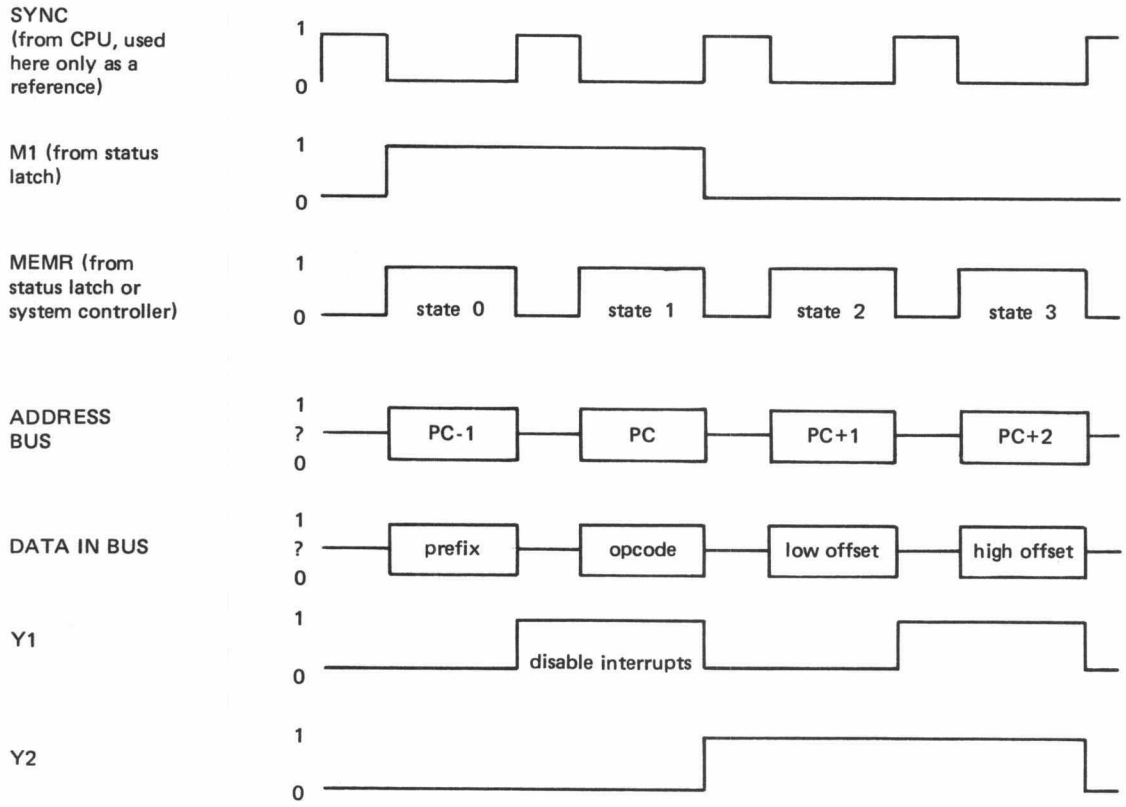
State 2: The 8080 fetches the low byte of the offset. This is added to the low byte of the latch (IC2) by gating the output of IC2 through multiplexers IC7 and IC8 and adding the DI lines in adders IC5 and IC6. The carry into IC6 is zero since type D flip-flop IC11 was cleared during state 1. The sum from IC's 5 and 6 is sent to the 8080. At the end of the MEMR pulse, the carry out of IC5 is latched into IC 11 and state 3 is entered.

State 3: The 8080 fetches the high byte of the offset. This is added to the high byte of the latch (IC1) in the same way as done in state 2. The carry into IC6 is the carry out from adding the low bytes. [Aren't you glad that the 8080 stores addresses low byte first?] This sum is sent to the 8080. At the end of the MEMR pulse, IC 11 is cleared, state 0 is entered, and the 8080 executes the relative instruction in the normal manner.

Notes: In states 0 and 1, the outputs of multiplexers IC5, 6 and carry flip-flop IC11 are zero. This is so that DI is not modified.

Fig. 5. Waveform during the fetching of a relative instruction.

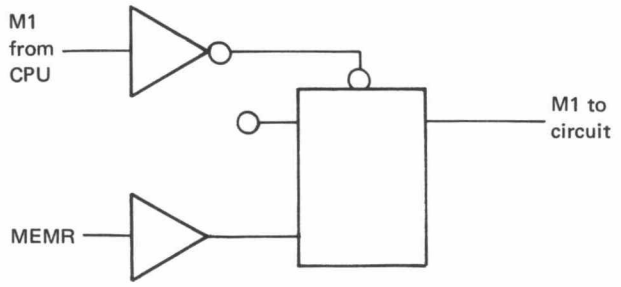
Let PC = the address of the first byte of the relative instruction.



M1 must be stable throughout the MEMR pulse, or the circuit will miss prefix instructions. If your CPU converts M1 to a short pulse (like mine does), then the pulse must be extended (see Fig. 6).

Figure 6. Pulse Extender for M1

The following circuit extends M1 to last until MEMR goes from 1 to 0. Extra gates and flip-flops from the Relative Address Modification circuit may be used.



The adders slow down the data a little, but this is not a problem in 2 MHz systems with 450 nsec memory. It might be a problem in 4 MHz systems.

It is possible to get rid of latch IC1 and IC2 by playing with IC11 and changing the software, since PC+1 and PC+2 are on the Address bus during State 2 and State 3 respectively. However, the software required from the relocation is more difficult to understand. Remember, software is more expensive than hardware when you include the time spent on it.

Interrupts

When a prefix is read, the next instruction will be relocated, come what may. If an interrupt occurs between execution of the prefix and the relative instruction, the interrupt routine will be executed incorrectly, and so will the main program if the CPU ever returns from the interrupt. It would be disastrous, and certainly is wrong.

If your system uses interrupts, it is necessary to turn them off during execution of a relative instruction. This can be done quite simply in hardware. Most S-100 systems use a priority encoder IC (74148) to handle interrupts. The EI input (pin 5), which allows interrupts when low, is usually grounded so as to always allow interrupts. If the EI input is instead connected to y_1, then an interrupt cannot occur between the prefix and the relative instruction.

Summary

This was a very exciting project because I was able to *change* the instruction set of my computer, increasing its power. However, it was very difficult to debug, because if it does not work, then the computer itself might not work. You cannot use the computer to debug itself, the way you can with most other peripherals.

Fig. 7. IC list

IC #	Part #	
1,2	74LS273	Octal latch
3,4	74LS85	Comparator
5,6	74LS283	Quad full adder (74LS83 may be substituted with modified pin-out.)
7,8	74LS157	Quad dual input multiplexer
10	74LS00	Quad NAND gate
9	74LS112	Dual negative edge-triggered J–K flip-flop
11	74LS74	Dual edge-triggered type-D flip-flop

The 3-state drivers in Fig. 3 are already on the CPU board.

References

1. Gaskell, James P., "Relative Addressing for the 8080," *BYTE* Dec. 1977, pp 162-163
2. Zolman, Leor, "A Machine Code Relocator for the 8080," *BYTE* July, 1977, pp 92-95
3. Intel, *MCS-80 User's Manual*

Letters

MORE ON FRITS' DISK

Dear Suzanne,

Just received a nice letter from Frits van der Wateren regarding my FLEX disk driver routines for his LISP which appeared in *DDJ* #34. He sent along a revision to the GETNAM routine in my drivers. While my method worked, Frits' is the correct way to retrieve the filename. I reassembled my drivers using his routine and they work fine. I enclose the new part of the source here for the benefit of your 6800 readers.

With the new drivers installed it is necessary to use the QUOTE function when giving the filename. Here's an example.

* (OPEN 2 (QUOTE RESTORE)) * (EVAL (READ 2)) * (CLOSE 2) * (RESTORE (QUOTE UTILITY))

After installing Frits' routine, the SAVE function works with the following format:

* (SAVE FILENAME FUNCTION1 FUNCTION2 . . . FUNCTION)

One additional point. My patches at $140 and $149 were used to stop Frits' automatic memory allocation routines before they reached the disk and IBM drivers I have at the top of lower memory. He suggests that readers who need to protect the drivers do it with the enclosed CLRCEL routine. Of course if the reader places the drivers in ROM or somewhere in high memory he can leave Frits' original routines intact.

Thanks for making the drivers available to your readers.

Sincerely,
Dale L. Puckett

163 Farm Acre Rd.
Syracuse, N.Y. 13210

```
PUTCAR  STX     LSPNAM      save running pointer in atom
        LDX     FILNAM      get filename pointer
        STA A   0,X         store character
        INX
        STX     FILNAM      swap pointers pack
        LDX     LSPNAM
        RTS
NAERR   EQU     $06CF       pointer to error routine

GETNAM  LDX     #FCB+4
        STX     FILNAM      setup filename pointer
        LDX     ARG2        points to an atm in LISP
        LDA A   1,X
        ROR A               is the argument atomic?
```

(please notice ...) the equ statement must be changed also to make ARG2 equal to $16:

```
ARG2 EQU $16
```

This change must be made in addition to the GETNAM routine.

```
        BCS     ATOMIC      yes, ok
        JMP     NAERR       no, print error and return nil
ATOMIC  LDX     0,X         get print name
        DEX                 clear atom mark
        LDA B   #8          setup character count
RDNAM   LDA A   0,X         get even char. from atom
        BSR     PUTCAR      and put in filenam
        LDA A   1,X         get odd char. from atom
        BMI     FILL        end of atom of neg.
        BSR     PUTCAR      else put in filename
        SUB B   #2          more than 8 char?
        BEQ     FILEXT      yes, ignore remainder
        LDX     2,X         no, get link to next char pair
        BNE     RDNAM       loop when not nil
FILL    CLR A
        BSR     PUTCAR      fill up with zero's
        DEC B
        BNE     FILL
```

```
0144                    ORG     $0144
0144 6F 00      CLRCEL  CLR     0,X         clean up cell storage
0146 08                 INX
0147 8C 3D 00           CPX     #$3D00      Done?
014A 26 F8              BNE     CLRCEL      no
014C 01                 FCB     1,1,1,1,1,1 fill with NOP's
014D 01 01
014F 01 01
```

REENTRANT ROUTINES FOR THE 6800

Dear *Dr. Dobb's*,

In *DDJ* #29, Mike Gabrielson addressed the problem of coding reentrant, non-destructive register push/pop routines for the 8080. The 6800 poses similar problems even when just dealing with the X register, as that register may not be transferred directly to the stack nor to the A or B registers. The routines shown below do not attempt to use the SWI instruction because of the well publicized NMI-SWI bug in the early 6800 chips.

Yours truly,
Harry B. Stewart

Neoteric
15816 San Benito Way
Los Gatos, CA 95030

```
XRSAV   FDB     0           TWO BYTES OF RAM REQUIRED.
*
* PUSHX AND PULLX ROUTINES
*
* "PUSHX" PUSHES THE X REGISTER TO THE STACK.
* "PULLX" PULLS THE X REGISTER FROM THE STACK.
*
* THE ROUTINES DO NOT ALTER ANY REGISTERS NOR THE CONDITION CODES,
* THEY ARE FULLY RE-ENTRANT, AND DO NOT DISABLE INTERRUPTS
* AT ANY TIME.  THEY WORK IN CONJUNCTION WITH THE NMI.
*
* PUSHX -- PUSH THE X REGISTER TO THE STACK
*
0200 34       PUSHX   DES             MAKE ROOM IN THE STACK FOR ...
0201 34               DES             ... THE ADDITION OF THE X REG.
0202 36               PSH A           SAVE THE OTHER REGS FIRST.
0203 37               PSH B           .
0204 07               TPA             .
0205 36               PSH A           .
0206 96 01            LDA A   XRSAV+1 SAVE WORKING RAM ...
0208 36               PSH A           ... IN THE STACK.
0209 96 00            LDA A   XRSAV   .
020B 36               PSH A           .
020C DF 00            STX     XRSAV   NOW STORE X REG TO RAM.
```

A GIFT FROM THE SILICON PATHWAYS

Dear *Dr. Dobb's*,

A fair bit has been written recently about so-called undocumented instructions.

These are unexpected gifts from the manufacturer of your central processor chip. They aren't mentioned in the manuals and don't show up in the instruction set. It just so happened that some brave soul wondered what would occur if he keyed in a 3F instead of a 3E. And presto, out came a brand-new function.

There are no guarantees, mind you. Since these spurious instructions crept by accident into the silicon pathways, they don't necessarily work on every single CPU chip from the same manufacturer. A code which performs a tricky double-register subtract on one machine might just as well send another limping off into outer space.

With all modesty, I must say that I'm something of an expert on the subject. My processor—let's call it a FRUMP 9000—seems to be possessed of an uncommon number of undocumented instructions. In fact, I've been tracking them down for years. I've even taken the liberty of giving them mneumonics.

Unfortunately, I can't provide the hex codes that go along with them. For some reason, these elusive instructions never turn up twice in the same place. And my disassembler can't seem to pin them down firmly. Several acquaintances have told me they experience similar unexpected surprises in their instruction sets and urged me to spread the word.

Why, they may even become an industry standard.

Sincerely,
Doug MacDonald

Box 515
Winfield, B.C.
Canada

MNEUMONIC	INSTRUCTION
BH	Branch and Hang
IIB	Ignore Inquiry and Branch
TDB	Transfer and Drop Bits
DO	Divide and Overflow
SRZ	Subtract and Reset to Zero
FSRA	Form Skip and Run Away
RAST	Read and Shred Tape
BST	Backspace and Stretch Tape
SRSD	Seek Record and Scar Disc
RIRG	Read Inter-Record Gap
UER	Update and Erase Record
SPSW	Scramble Program Status Wor
EIOC	Execute Invalid Op Code
EROM	Erase Read-Only Memory
PBB	Print and Bend Bar
SM	Scramble Memory
MLR	Move and Lose Record
CRN	Convert to Roman Numerals
DMPK	Destroy Memory Protect Key
EPI	Execute Programmer Immediate
HCF	Halt and Catch Fire

6800 ROUTINES cont.

```
020E 30          TSX             PREPARE TO MANIPULATE STACK.
020F 96 01       LDA A  XRSAV+1  PUT XR VALUE TO STACK ...
0211 E6 08       LDA B  8,X      ... & MOVE PC DOWN.
0213 A7 08       STA A  8,X      .
0215 E7 06       STA B  6,X      .
0217 96 00       LDA A  XRSAV    REPEAT FOR UPPER BYTE.
0219 E6 07       LDA B  7,X      .
021B A7 07       STA A  7,X      .
021D E7 05       STA B  5,X      .
021F DE 00       LDX    XRSAV    RESTORE X REG VALUE COMING IN.
0221 32          PUL A           RESTORE WORKING RAM.
0222 97 00       STA A  XRSAV    .
0224 32          PUL A           .
0225 97 01       STA A  XRSAV+1  .
0227 32          PUL A           RESTORE THE OTHER REGS.
0228 06          TAP             .
0229 33          PUL B           .
022A 32          PUL A           .
022B 39          RTS             RETURN
                 *
                 * PULLX -- PULL THE X REGISTER FROM THE STACK
                 *
022C 30    PULLX TSX             PREPARE TO ALTER THE STACK.
022D 36          PSH A           BUT FIRST SAVE REGS.
022E 37          PSH B           .
022F 07          TPA             .
0230 36          PSH A           .
0231 A6 03       LDA A  3,X      SWAP THE X REGISTER ...
0233 E6 01       LDA B  1,X      ... WITH THE PC.
0235 A7 01       STA A  1,X      .
0237 E7 03       STA B  3,X      .
0239 A6 02       LDA A  2,X      REPEAT FOR OTHER BYTE.
023B E6 00       LDA B  0,X      .
023D A7 00       STA A  0,X      .
023F E7 02       STA B  2,X      .
0241 3B          RTI             DATA IN RTI FORMAT, DO IT.
```

The Software ©pyright Issue

While taking a communication law course at Stanford this past winter, I became interested in the subject of software copyright. I wrote to a few magazine editors to ask their opinions on the subject: the first I knew that BYTE *had published my letter was when I began to receive replies from* DDJ *readers who had seen my letter in that magazine. Here are a few samples.—SMR*

SOFTWARE

Dear Miss Rodriguez,

I read with interest your letter to *BYTE* about legal protection of computer software. I believe there is a very clear set of precedents governing software, and that money is actually the "dividing line", as you put it, between recognition and non-recognition of this fact! The precedents to which I refer—don't laugh—are simply those that govern animal husbandry, specifically (ahem!) stud fees. If you compare programs with prize animals, I think you will see what I mean. They "propagate" with the aid of humans (just like cattle!), and no control over the possession and use of progeny is carried beyond the original sale.

Unfortunately, everyone finds this distasteful, since there is much less money in it.

Sincerely,
Daniel R. Killoran, Ph.D.

34-B Atlantic St.
Lynn, Mass 01902

BOTH SIDES NOW

To: Suzanne Rodriguez

I recently read your letter in April *BYTE* magazine on copyright and thought that I'd let you know some of my thoughts on this subject.

As an owner of an Apple II (48K ROM, 2 mini-floppys, communications gear, and selectric typewriter conversion kit on order), I find that I cannot operate efficiently, due to lack of program development tools. The Apple II desperately needs:

- Text editor
- Word processing software
- Data base management
- Assembler
- LISP, PASCAL, etc., compilers
- Generalized disk management routines
- Lesson authoring system
- Graphics development package
- Simulation monitor
- Macro preprocessor
- Business applications programs
- Emulators of other microprocessors

If I ever attempted to write all these programs, I would never have time to program the functions that I bought the Apple II for. My options are to buy programs, trade programs, copy programs from magazine articles, and convert programs written for other computers. Buying programs offers the most promise as a solution; it also offers the most problems. Because of the problems associated with buying software, I have spent much time thinking about whether the cost of a program is justified. I have come up with the following reasons why the cost of a program may be justified to a buyer:

Usable function The prerequisite to buying a program, at any price, is a high probability that the function performed by the program is a function that I need and will use. I have not found program advertising to be helpful in determining what a program will do. Rather I rely on magazine reviews, the advice of the Computerland personnel, talking with users of the program, reading the documentation (when it is available separately), and hands-on exposure to program operation.

Uniqueness There are a number of programs that do the same function. When I try to justify a program's cost, I look at what that program will do that other similar programs do not do. A program which is unique is worth more than one which is similar to several others, particularly if I already own one of the others.

Compatibility Several times I've refrained from buying programs which didn't appear compatible with other elements of my configuration. Right now, my biggest design headache is trying to use DOS while also using the Communications gear.

User interface My 7-year old son uses my computer to drill in mathematics and to play games; I often have guests interested in the computer. An easy-to-use interface to the program is important to me. Sometimes I buy a program only because of the user interface. I will then use the code for the interface in some of my own programs.

Documentation The more detailed the documentation for a program, the more of it I can use. On the Apple II, the monitor listing is published, allowing users to CALL monitor subroutines. The BASIC interpreter is not listed; therefore, users recode routines that may already be available in the BASIC interpreter.

Adaptability Rarely is a program offered for sale able to satisfy all user requirements completely. Each user thinks about customizing a program to be suitable for his particular environment. This can be helped with good *internal* and *external* documentation of the program. Also, good programming techniques help make a program easier to change.

The above-mentioned qualities relate worth of the program to the user. The cost of a program is also justified by the expenses related to its production:

Design A program must be designed to do more than perform a function. It must be designed to handle input errors; it must be designed to be easily enhanced. Often it will take a 50-statement program to interface to a one-line formula.

Implementation This includes both conversion of design to code statements and the actual entering of code statements into machine-readable instructions. If I have a choice between paying $5 for a program or getting it free from a magazine, I will pay the $5. Many user groups now assign magazine programs to individuals to enter into machine-readable format. Then the whole group can share the typing effort of one member.

Test Hopefully, each program offered for sale will be thoroughly tested. Experience shows that this is one of the greatest variables with purchased software.

Maintaining the program Some program shops are finally offering maintenance for the programs they produce. This includes both enhancements and fixing bugs. This should be included with *each* program offered for sale.

Distribution and marketing A program producer must pay to have programs reproduced for purchasers, must advertise what is for sale, and must pay mailing costs.

Number of copies sold The price of software must be such that a software manufacturer will be able to make a profit. This depends directly on how many copies of the program will be sold, which depends on the demand and on the honesty of the computer users.

Profit Without a profit motive, I imagine that few businesses would continue to endure the hassle of offering programs for sale.

Only when computer users and computer program manufacturers realize each others needs will the problem of copyright be solved. If a user does not feel the program he has purchased is giving full value for the amount of money paid, then there will be no hesitation in making copies for other users without any compensation to the manufacturer. If a program shop does not test its code thoroughly, the customer will not care if it goes out of business or not.

Similarly, if the customer does not understand the expenses of the program shop and the need to make a profit, it will be a bad situation, in which the customer comes out the loser. This may be either by having to pay higher prices for software or by not having a good selection of software from which to choose.

At the present time, I see open warfare between the computer users and the program producers. Neither side wants to consider the needs of the other. Instead, both wallow in self-pity over the treatment they receive at the hands of the other. I would look for the computer magazines to be the mediators in this situation, by trying to look at the whole situation, not just one facet of the problem (such as copyrights).

Sincerely,
DeWitt Brown

1650 9th Ave SE
Rochester, MN 55901

HELPFUL HINT

Dear Suzanne,

I read recently your letter, published in *BYTE* about copyrights, etc.

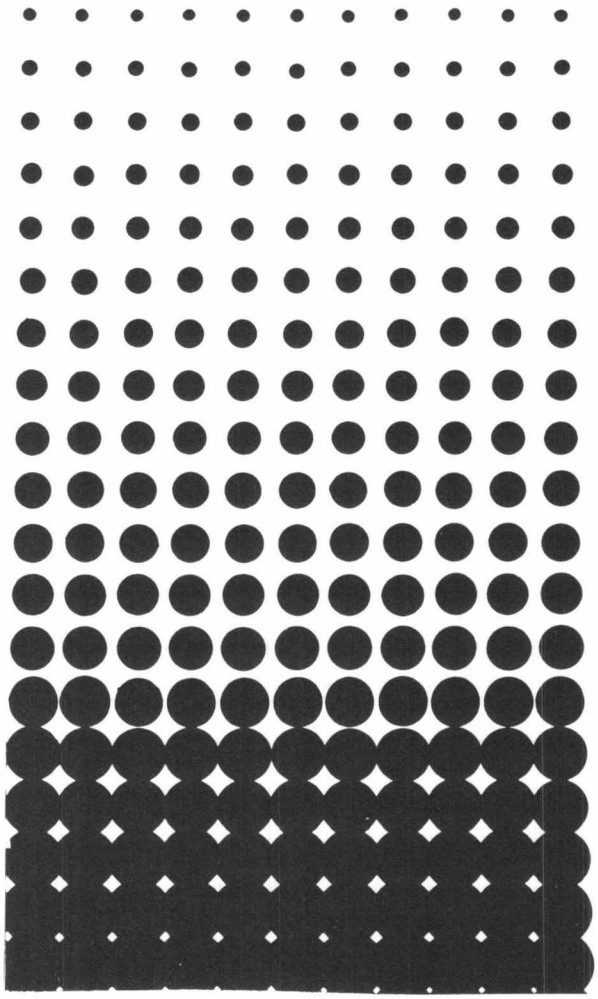
I may be the Nth person writing or telling you about this, but looking through the Computers and Data Processing title list of the U.S. Printing Office, I saw listed a $4, 267-page publication bearing the title:

Copyright in Computer-Readable Works;
Policy Impacts of Technological Change
Stock number: 003-003-01843-1; published in 1977.

Well, it seems that you are enjoying being editor of *DDJ* so keep up the good work . . .

Sincerely,
Dimitiri Marinakis

P.O. Box 1486
University, Alabama 35486



REGARDING RICE'S "FURTHER THOUGHTS ON ACKERMANN'S FUNCTION"

Eryk Vershen, a young computer whiz and college student who works around PCC from time to time, saw proofs of the Rice article and put the following note on my desk. —*SR*

The notation Rice mentions was described by Martin Gardner in his *Scientific American* column for November 1977. It was invented by D.E.Knuth. Here is the proof he wanted.

Proof that $A(m,n) = 2\uparrow^{(m-2)}(n+3)-3$ for $m \geq 3$

Notation: $a\uparrow^{x}b = a\uparrow\uparrow\ldots\uparrow b$ where $\uparrow$ occurs x times

The proof is by induction on m. Rice showed that $A(3,n) = 2\uparrow(n+3)-3$.

We must show that if $A(m-1,n) = 2\uparrow^{(m-3)}(n+3)-3$ then
$A(m,n) = 2\uparrow^{(m-2)}(n+3)-3$.

$A(m,n) = A(m-1, A(m,n-1))$;by definition of A(m,n)

$= A(m-1, A(m-1, A(m,n-2)))$

o
o
o

$= A(m-1, A(m-1, \ldots A(m-1, A(m,0)) \ldots))$;A(m-1, occurs n times

$= A(m-1, A(m-1, \ldots A(m-1, A(m-1,1)) \ldots))$;A(m-1, occurs n+1 times

Let $a = m-3$. Then

$A(m,n) = 2\uparrow^{a}([\ 2\uparrow^{a}([\ \ldots([\ 2\uparrow^{a}(4)-3\ ]+3)\ \ldots\ ]+3)-3\ ]+3)-3$;2 occurs n+1 times

Since $2\uparrow^{x}2 = 4$ for all $x \geq 1$ we have

$A(m,n) = 2\uparrow^{a}([\ \ldots([\ 2\uparrow^{a}(2\uparrow^{a}2)-3\ ]+3)\ \ldots\ ]+3)-3$;2 occurs n+3 times

$= 2\uparrow^{a}(2\uparrow^{a}(\ \ldots(2\uparrow^{a}2)\ \ldots))-3$;2 occurs n+3 times

$= 2\uparrow^{(a+1)}(n+3)-3$;by definition of $\uparrow$

$= 2\uparrow^{(m-2)}(n+3)-3$

Therefore, by induction, $A(m,n) = 2\uparrow^{(m-2)}(n+3)-3$ for $m \geq 3$.

SORRY . . .

Dear Ms. Rodriguez:

Our attempts to make last-minute changes to my manuscript "Patching the SWTPC Co-resident Editor-Assembler," *DDJ* #35, got Murphy-ized.

On Page 32, left column, 5th paragraph from the bottom, the hex digits of my correction got scrambled and put in the wrong place.

The entire paragraph should read:

> Leader and trailer tape generation for the 2P and 2T passes are done by a routine at $Ø139. It calls $E1D1 (OUTEEE) at $Ø16E. The code at $Ø16E should be changed to JSR OUTQ (BD 1A 9D).

I might mention here in passing that the recommended patch causes the "TOTAL ERRORS ØØØØØ" message at the end of the 2T pass to be punched on the tape instead of being sent to the terminal. However, since this message follows the "S9" terminating code on the tape, it does not affect loading of the tape. One would hope that nobody punches an object tape until after he's already gotten that reassuring message from a 2L pass.

I hope nobody gets messed up too seriously by the typo (the correct address is noted in Table 1 and in the pointer list on Page 35).

Yours very truly,
Geoffrey A. Gass

5240 S.W. Dosch Rd.,
Portland, Oregon 97201

MODS TO NORTH STAR HORIZON

Dear *Dr. Dobb's*:

Following is a brief description of the modifications we are now making to the North Star Horizon (TM North Star Computers Inc) in order to allow proper operation of the Micropolis Mod II and Mod I disk drives:

It has been some time since I last communicated with you—the last time was during your "Longest Delivery Time" contest. Since then, much has changed (usually for the better). We no longer carry Miniterm or Digital Group products, but instead, we have directed our efforts at the North Star Horizon/Micropolis combination, so far with excellent results. The combination gives us one of the most reliable and highest density drives in a low-priced, well designed mainframe that is very well documented for both construction and maintenance. Combine it with Seattle Computer Products 16K PLUS memory boards and you have the basis for a reliable small business or big hobbyist computer.

As usual, I find *Dr. Dobb's* to be informative and enjoyable. Unfortunately it is too short. Please keep up the good work.

Yours truly,
John R. Atwood
Manager, Retail Sales

TJB Microsystems Ltd.
10991-124 Street
Edmonton, Alberta,
Canada T5M 0H9

To modify the North Star Horizon as mentioned above:

1. Replace the heat sinks for the 5 and 12 volt regulators used to drive the disk units with larger heat sinks. We use taller heat sinks which extend past the solid portion of the side of the mainframe into the air-flow region.

2. Move the disk drive mounting plate forward by 0.5 inches by drilling new #6 holes in the mainframe base. Note that the internal card cage frame must also be re-drilled.

3. Drill new access holes in the mainframe base. Use a 0.5 inch bit or larger, and drill slowly to prevent tearing where the new hole overlaps the old. Polish all holes to remove burrs and loose pieces of metal.

4. Construct power cables to run from the disk drive power supplies on the motherboard to the J5 connector (pins 3, 4, 6, 7) on the disk drives.

5. Cut the internal card cage frame as shown in pic #3. This will allow the two smaller filter capacitors (8900 uf and 11000 uf) to be rotated 90 degrees and mounted.

6. Mount the drives using #6 hardware. A small shim may be added to the side of one drive to provide sideways support. The standard Micropolis 3 connector cable may be used (with lots of folding), or a new cable made from the supplied one. We recommend the latter—the Micropolis cable is rather long for this application.

7. Make the software changes as shown in the following illustration. These changes will guarantee that the ports are initialized properly when "linking" back to MDOS from BASIC.

SOFTWARE CHANGES FOR THE NORTH STAR HORIZON / MICROPOLIS FLOPPY DISK

1. Apply the following software changes after booting the original system diskette, but before executing location 04D6H to run the Micropolis configurator:

```
Location    Hex Code
04D0        01                                  Set configuration to #1.

2BCE        14          DB      20              Length - overlay the SIO code
2BCF        AF          XRA     A               Zero
2BD0        D3 06       OUT     06              Reset motherboard
2BD2        D3 06       OUT     06
2BD4        D3 06       OUT     06              I tell you 3 times
2BD6        3E CE       MVI     A, 0CEH
2BD8        D3 03       OUT     03              Status port - main console
2BDA        D3 05       OUT     05              Status port - printer
2BDC        3E 37       MVI     A, 037H
2BDE        D3 03       OUT     03
2BE0        D3 05       OUT     05
2BE2        C9          RET

050A        CB 06                               Address of printer support
```

2. Set the computer to execute at location 04D6H and run. Save RES as usual and you should now be up and running. Note that the printer initialization area is not used since the initialization for both ports is done in one area.

PIC # 6.

CALL FOR A COMMON BASE LANGUAGE

A standard question Japanese ask English-speaking people is whether it is better to learn "standard American English" or "Queen's English." If you are not asked, you may be *told* what someone's exclusive preference is. The parallel in the computer world is, of course, all the people who write to magazines to say that they have invented or discovered a new high level language that will render all previous languages obsolete. Certainly it is valuable to see new language features—a few of them may eventually find their way into the widely-used languages, or become widely-used languages.

My personal opinion is that the purpose of any language is communication—being able to communicate with as many people (and computers) as possible. As with human languages, the different computer languages are unlikely to vanish in favor of a new *single* standard language. Languages adapt (slowly) and adopt new vocabulary—technical words and concepts from other languages—to meet needs.

Although a single common language is unlikely, a common "base" language is valuable as an interface. For international conferences, a base language such as English can be translated as necessary. For computing to flourish, surely a "lowest common multiple" intermediate language seems necessary to ensure program portability between different computers and different high level languages.

By compiling to the intermediate language and then compiling or interpreting the intermediate language, compilers for

any language would become relatively portable. The intermediate language to machine code compiler/interpreter would be machine dependent, but the remainder would be almost completely portable. This assumes that the operating system as well as the language is the same. By using apppropriate compilers and uncompilers it would also be possible to convert between different high level languages.

Maybe it would be a good idea to standardize a compact intermediate language? There is probably a demand for a language with the power-per-instruction productivity that most high level languages provide, and the user-extensibility, speed, minimal redundancy and memory efficiency possible with assembler-level languages. The tiny-machine hobbyists and the purists could write speed-and-memory-efficient programs, operating systems and compilers in the intermediate language, and it would be a standard interface to compilers for a variety of high level langauges. For efficiency, the intermediate language should eliminate redundancy.

I would like to see:

• Improvements in existing languages—or "concise" new languages—to *eliminate redundancy in language*: I use "redundancy" to mean not only the repetition of variable and operator names/symbols within lines and between lines, but also the repetition of the same constant in multiple assignment statements and conditional statements. If a language largely eliminates redundancy, it eliminates unnecessary duplicated memory references and subscript calculations, and it doesn't need a slow sophisticated optimising compiler to eliminate such redundancy. Less redundancy means less time typing programs—fewer typing errors likely—and smaller source programs.

```
The C language eliminates repetition within statements:
p(i+j+1) + = 3          rather than  p(i+j+1) = p(i+j+1) + 3;

x + = 5     )     rather than     x = x + 5   )
y + = 4     )     rather than     y = y + 4   )
z = x / y   )                     z = x / y   )

The HP20* language eliminates repetition between lines:
(but it reads left to right, and uses a special character, a right arrow "->";
"=" could be used instead, but might confuse some people).
0 ->x ->y ->z          rather than  x = 0,  y = 0,  z = 0

x + 5 ->x / (y + 4 ->y) ->z  rather than  x = x + 5,  y = y + 4,  z = x / y

Combining these features of C and HP20* languages would give a more compact
language:
5 + = x / (4 + = y) = z  (using the example above).          *Hewlett Packard

     Other cases where most existing languages don't eliminate redundancy:
(1)  Substitution:
     How about  x : a * - b ) * + c = @ v        rather than   @ v = a * x * x - b * x + c
                                                    that is,   @ v = (a * x - b) * x + c

(2)  Multiway comparisons:
     IF (A .EQ. 0. .OR. B .EQ. 0. .OR. C .EQ. 0.) . . .
     (the constant 0. and operator .EQ. are repeated), and
     IF (CHAR .NE. COMMA .AND. CHAR .NE. CRLF .AND. CHAR .NE. PERIOD) . . .
     (the variable CHAR and operators .AND. and .NE. are repeated).
     Instead, how about GS# 0 A, B, C              (GS# meaning Group - Same - Or)
     and GD& CHAR COMMA, CRLF, PERIOD              (GD&  "  "  Group - Different - And)
     Also GS# PHRASE A(*                           (Find I for A(I) to match PHRASE).

(3)  Repeated addition:
     G+ arglist   or   G+ arrayname(*   or   G+ arrayname(row, *
```

• "Feedback" to help the programmer improve his programming:

(a) I would like to see preprocessor-optimizers that give some feedback to a programmer. Such a preprocessor-optimizer could look through a program and *list bad or doubtful programming* – variables that are evaluated but not used, unreachable statements, arithmetic between constants, multiple evaluations of the same expression. An "inverse trace" facility to *list only statements unused during a run* would help trace unreachable statements.

(b) *Execution time profilers* show the percentage of CPU time spent on each part of a program. This information would show the programmer where he could *most profitably* improve his program.

Anonymous, Japan.

SYM-1

Dear Suzanne:

I have a new SYM-1 and I have gotten up and running on it the 6502 Disassembler from Apple, by Steve Wozniak and Allen Baum which was published in *DDJ* of September 1976. I am writing to inform you of some important corrections.

Under "MODIFICATIONS (d) To add ROR and addressing modes, change the following locations:", the code given is incorrect.

What follows is the correct code to obtain disassembly of all ROR instructions.

CHANGE: 0917 to 02
0918 to 45
0919 to B3
091B to 08
091D to 09
098F to 9C
09CF to 26.

Those changes apply for using the disassembler with any 6502 system, as should be obvious.

A note on using this program on the SYM-1. All calls to PRBYTE in the listing must be changed to SYM's OUTBYT(82FA). All calls to CHAROUT in the listing must be changed to SYM's OUTCHR(8A47). One exception to the CHAROUT calls, however, might be the one at 08D5. I changed this one to SYM's CRLF(834D), and the program works correctly. I have not tried using OUTCHR there, but it may work also.

AND, one further note: the value of 0801 in the listing is given as 13. This determines the number of lines to be disassembled in one batch, and 13 makes the number to be 20 lines. This was too many for my 16 line display, so I changed 0801 to 0E. This results is 15 lines of disassembled code and leaves one line for the final carriage return.

Finally, I would like to hear from others who are using the SYM-1.

Sincerely, Rt. 2 Box 50 A-1
Stephen E. Bach Scottsville, VA 24590

MEMORY BOARD RECOMMENDATION

Dear Editor:

This is a report of good service from a relatively new company, Delta Products of Long Beach, CA 90806. I ordered one of their new memory boards, 32K assembled and tested. I am especially pleased with the quality of the board and the price, $485. Delivery was on time, as promised. They also sell memory kits.

Yours truly, 4209 Knoxville
Pete Sargent. Lakewood CA 90713

Z80 ZAP CORRECTION

Dear *Dr. Dobb's*:

An error was found in the listing of the Z80/ZAP DISASM (*DDJ* #35). Also, because of an unfortunate oversight, a group of index register OPCODES was left un-implemented.

First, the error is in the jump destination of the instruction at relative address 0154. The correct assembler code should have been JC RET instead of the JC IXHRT. (See instruction at relative address 0172.)

To correct the oversight and implement the missing index register OPCODES, the following additions are needed:

1. Add the label TYICB to the code at relative address 02BA in routine TYPIC, as shown in the following:

```
02BA    TYPIC:    CALL    PROP
                  CALL    REGM
                  CALL    COMMA
02BA    TYICB:    INX      H
                  MOV     A, M
                   .       .
                   .       .
                   .       .
```

2. Insert two new routines TYPIF and TYP20 at the end of routine TYPIE and before routine SFRET:

```
TYPIF:  CALL    PROP
        JMPR    TYICB    ; Do last of TYPE IC
;
TYP20:  INX     B        ; Pt to OPCODE entry
        LDAX    B
        STA     CCI      ; store as 1st char
        INX     B
        LDAX    B
        STA     CCI+1
        INX     B
        LDAX    B
        STA     CCI+2
        POP     B        ; Restore ireg id ptr
        INX     B        ; skip paren
        LDAX    B        ; Get ireg id
        STA     CCB      ; Store as 4th char
        LXI     B,CCI-1  ; set B for output
        MOV     A,M      ; Reload OPCODE
        CPI     29       ; Ireg, IREG case?
        JNZ     TYPE3    ; No, then same as
                                 TYPE3
        CALL    PROP     ; Yes, then do it.
        LDA     CCB      ; Reload IREG ID
        CALL    ZTYPE    ; Output it
        JMP     IXHRT    ; INX H & EOL
```

3. Add two new entries to the JMTBL. Table just before the last entry ".WORD PRDB ; UNDEFINED" :

```
JMTBL:  .WORD   TYPEO    ; 1  NONE     DAA,  EI
          .       .       . .  .        .     .
          .       .       . .  .        .     .
          .       .       . .  .        .     .
          .       .       . .  .        .     .
        .WORD   TYPIE    ; 4  IREG,OFS RLCR  D(Y)
        .WORD   TYPIF    ; 3  IREG,OFS DCRD(Y),  ADD D(X)
        .WORD   TYP20    ; 2  IREG,RP  DADX H,   DADY SP
        .WORD   PRDB     ; UNDEFINED
```

4. Insert and add the following new entries to the OTBIR table, after .ASCII 'L . . . D' and before .BYTE OF8:

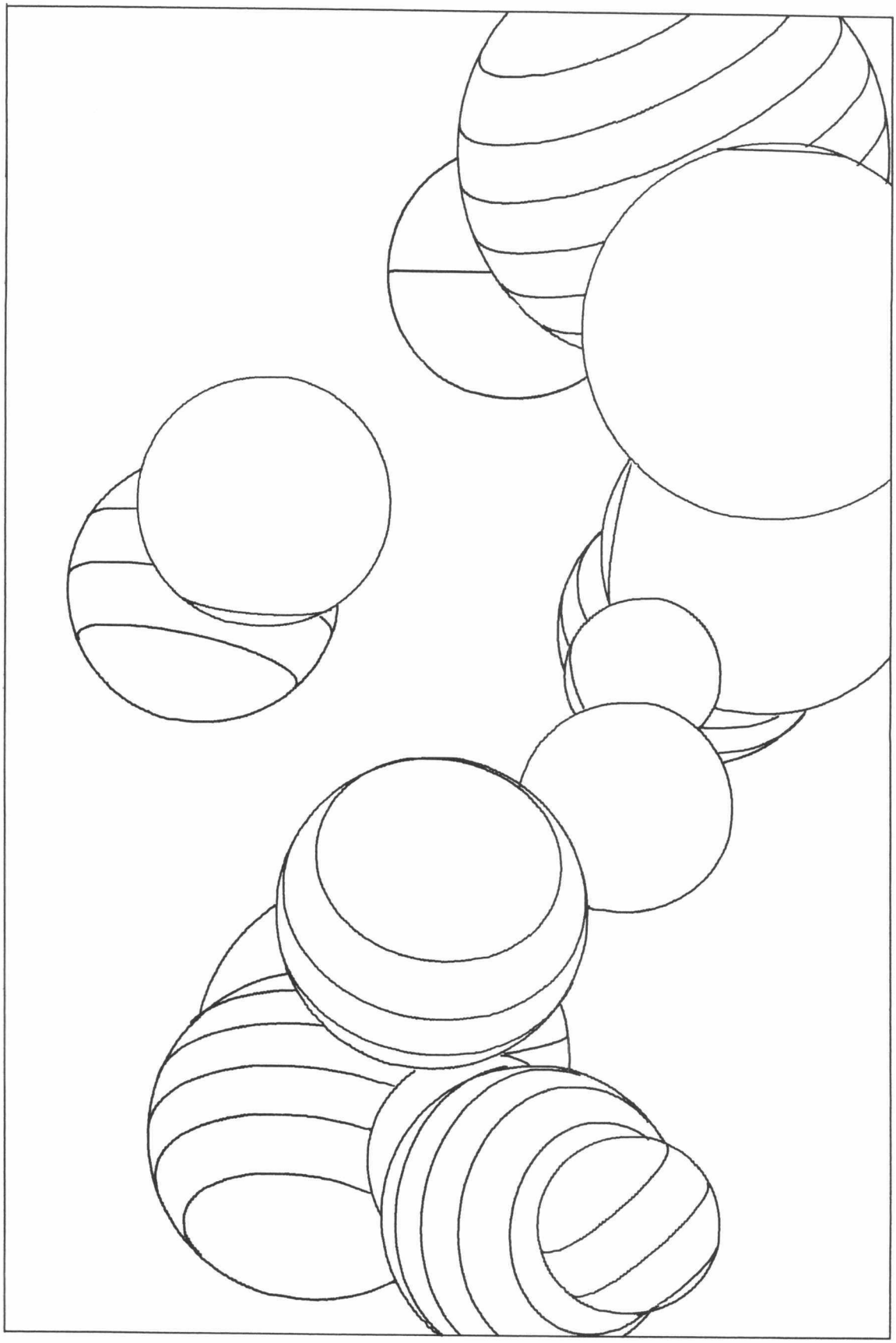
```
.BYTE    OFF
.WORD    IF34
.ASCII   'INR'
.BYTE    OFF
.WORD    IF35
.ASCII   'DCR'
.BYTE    OFF
.WORD    IF86
.ASCII   'ADD'
.BYTE    OFF
.WORD    IF8E
.ASCII   'ADC'
.BYTE    OFF
.WORD    IF96
.ASCII   'SUB'
.BYTE    OFF
.WORD    IF9E
.ASCII   'SBB'
.BYTE    OFF
.WORD    IFA6
.ASCII   'ANA'
.BYTE    OFF
.WORD    IFAE
.ASCII   'XRA'
.BYTE    OFF
.WORD    IFBE
.ASCII   'CMP'
.BYTE    OCF
.WORD    2009
ASCII    'DAD'
```

With these additions keyed into the assembler source program, it should be reassembled. The length of the program is a little longer but will still fit in the allocated memory space between 9200 Hex to 9C00 Hex. The added codes implement the following OPCODES:

```
INR D(X)    ADD D(X)    DADX 3
INR D(Y)    ADD D(Y)    DADX X
DCR D(X)    ADC D(X)    DADY H
DCR D(Y)    ADC D(Y)    DADY D
ANA D(X)    SUB D(X)    DADY B
ANA D(Y)    SUB D(Y)    DADY Y
XRA D(X)    SBB D(X)
XRA D(Y)    SBB D(Y)
CMP D(X)    DADX H
CMP D(Y)    DADX D
```

Sincerely,
B.W. Lee

1385 Rifle Range Rd.
El Cerrito, CA 94530



© David Dameron 1979

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 38 | September 1979 | Volume 4, Issue 8

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

An Electoromechanical Household Servant 4
D. J. Reynolds

Modular Programming With The Apple II 37
R. F. Zant

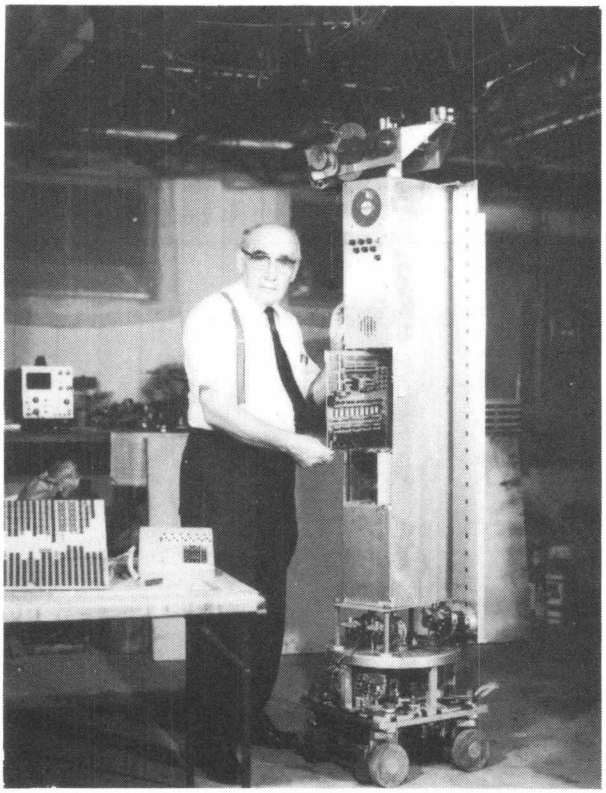
A General Purpose Data Compression Program 16
Graham K. Jenkins

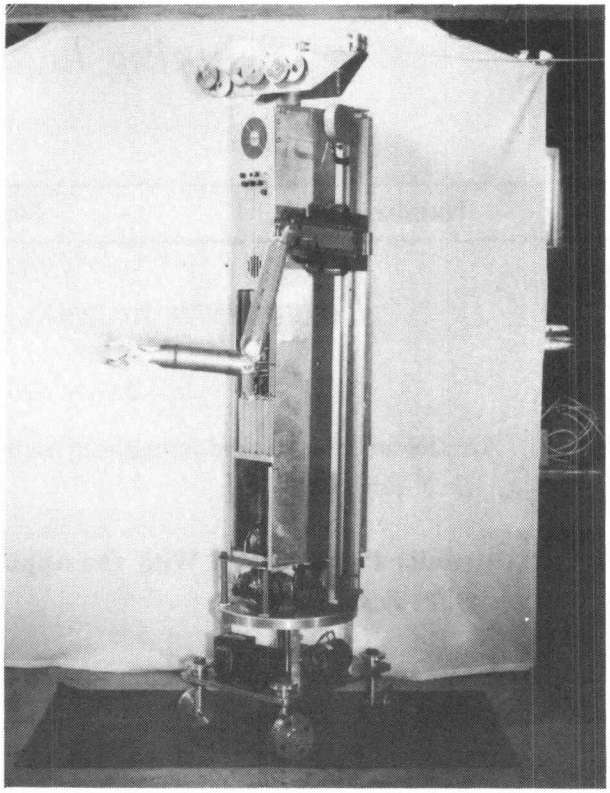
A Potpourri of Utility Functions for LISP 19
Paul Tarvydas

A Proposed Standard for A 16 Bit system Data Exchange 30
Charles B. Falconer

And Always more ...

An Electromechanical Household Servant





BY D.J. REYNOLDS
18 Silver Avenue
Ft. Mitchell, KY 41017

Delos Johnson Reynolds holds an E.E. Degree from the Cooperative University of Cincinnati. From 1939 until his retirement in 1968, Mr. Reynolds worked as a design engineer for Westinghouse. He has designed a variety of electrical, electronic and mechanical equipment.

Mr. Reynolds says that the design and building of a home robot is an attractive hobby because it does not require cooperation or permission, sometimes difficult to obtain. Robots are stand-alone items, and open-ended hobby projects; one needs to be very *patient due to the great difficulty in building a robot.*

INTRODUCTION

While modern household appliances really take the drudgery out of housework, they still require attention by the homemaker. Often the remaining part of a task requires more time-consuming attention and manual skills, which may make it very difficult to do several tasks simultaneously. A household servant in homes before World War II was the effective solution to this problem. Now available technology could be used to develop an electromechanical household servant, a "Home Robot." Homemaker activity surveys and studies have tended to show about five hours of actual housework per day done in the home for a family which includes at least one child under fifteen years of age. This excludes time devoted to child care and outside operations such as shopping.

A home robot should be able to perform these tasks:

Clothing Care: Load clothes in washer and dryer, sort clothing and put away
Room Care: Care of floors, dust furniture, make beds, clean bathroom and fixtures
Food Preparation and Cooking
Kitchen Clean-up
Special Tasks, such as housecleaning or walls, windows and drapes and other seldom-performed tasks.

It is apparent that a home robot will probably be a little slower than the typical housewife and may initially be unable to do all the listed tasks. It will likely operate at least five hours a day and will be very useful.

It will be a complex appliance and will probably have a price somewhat like that of an automobile. Present availability of 16 bit microprocessors and of large (4K or greater) static random access memory in DIP cases makes it now possible to build an adequate computer control system into the home robot. A large memory and a fast computer are necessary because of the number of programs which must be available and run on an apparently simultaneous basis to properly control motions. Programming for the home robot will become a very competitive activity.

Since the home robot will operate in already existing homes and new homes, this determines some of the design specifications of any home robot. For example, a height limit of 78 inches overall would allow robot to pass under a typical 6 feet 8 inch doorway with a threshold plate and rugs, and a width limit of perhaps 18 inches would be prudent for entering a typical 2 feet 8 inch wide doorway. Because of door-hinging, the actual opening width is only 30 inches. To avoid contact, the navigational errors would have to be held to ± 5½ inches. Another design specification decision would be on the question of whether to climb and descend stairs or steps. The writer's opinion is that operation on single level should be specified because there are many ranch-style single-level homes and in many others most of the housework is performed on one level. The big-city high rent apartments are mostly single-level. Adding stair-climbing would increase both development cost and risk and would also increase physical operational risks.

Development of the home robot now seems both possible and timely. Occasionally there are articles in the news of such efforts by small organizations or individual inventors. There appears to be no effort of the scope and size necessary to launch a home robot industry. Most knowledgeable observers will agree that the possible reward for a successful development is large. The deterrent: development risks, business risks and legal risks are substantial. Further, by the nature of the venture in the present legal environment, there is no way of assuring that the rewards for a successsful development will go to the developer to compensate the risk taken. Also, the risk is so large that failure would entail serious personal consequences for the individual who advocates the venture.

One approach which might overcome some of the above impeding factors would be to define the home robot with a provisional design specification. Additionally, one should separate a basic design into sub-assemblies and define the interfaces between them so that a coope-ative effort could proceed by individuals or organizations. This would divide the costs and reduce the risk factors. Also, some of the sub-assemblies could be of value independent of the success of the complete project. The home robot, as here defined, would be a mobile unit, which could be moved to any home with minimal requirements at the home. The major changes would be in programming to accommodate changes from home to home by reloading the computer programs. This would facilitate development trials after a functional development model is available.

CONSIDERATIONS OF THE ROBOT IN THE HOME

The great benefit obtained is automatic performance of many home chores. A complete, seven-day cycle of programs would be performed every week. Desired omissions, alterations and additions would be entered into the main program as soon as the need for change is known. At the desired day and time, the necessary sub-program would be moved from the bulk memory, and any change would be made. Then it will begin to run. The homemaker would have a little directory listing the entry call up for the many programs in bulk storage. To get these benefits, the household would need to operate in a more standardized way, particularly in respect to storage of food, cooking utensils and tableware. Since the robot would restore many of these items after use, this precaution might be stated as "let the robot work and don't interfere." Safety "stop" switch bars should be on the robot so that any danger to humans can be limited.

Another consideration is that the home robot will not be as small nor as flexible as a human and will therefore required more space to operate. Therefore, some rearrangement of furniture may be necessary to obtain the advantages of an automatic housekeeper. Longer-than-human telescopic arms will allow for fewer paths to be used so that this problem is minimized.

A general design consideration should also be to limit the available force of the various moving parts so that a human could successfully resist any accidental entanglement while stopping the robot. In many respects, the home robot design seeks performance capability of about that of a 90-pound human. Estimates of design weight for an experimental model, for which some major portions have been built or purchased, run between 250 and 300 pounds. It appears likely that a production home robot might weigh around 200 pounds. With four 5-inch diameter by 1¼ inch wide wheels on shag rug, the force needed to start pulling the robot is about 0.25 of its weight or a design figure of 75 pounds. At a speed of one foot per second, this corresponds to 0.136 horsepower. This speed is selected because humans would need to be able to evade and avoid interfering with the robot. The wheels could have tires of woven material, such as the hard wool closed-loop pile stair runners. This should reduce tracking and give slip-free propulsion. All wheels should be powered so that any wheel will climb upon a throw rug lying on a polished floor and not just push the rug along. Another safety consideration in design is to avoid close-moving parts on exposed portions of the robot.

A feature of the home robot that affects the travel requirements is the length of the arms. By having a long telescopic part to the arm, it will be able to program the robot to make a bed from one side or to set a table from one side. This is needed for small rooms. It also makes it possible to do much work in the kitchen area without traveling. Travel programs

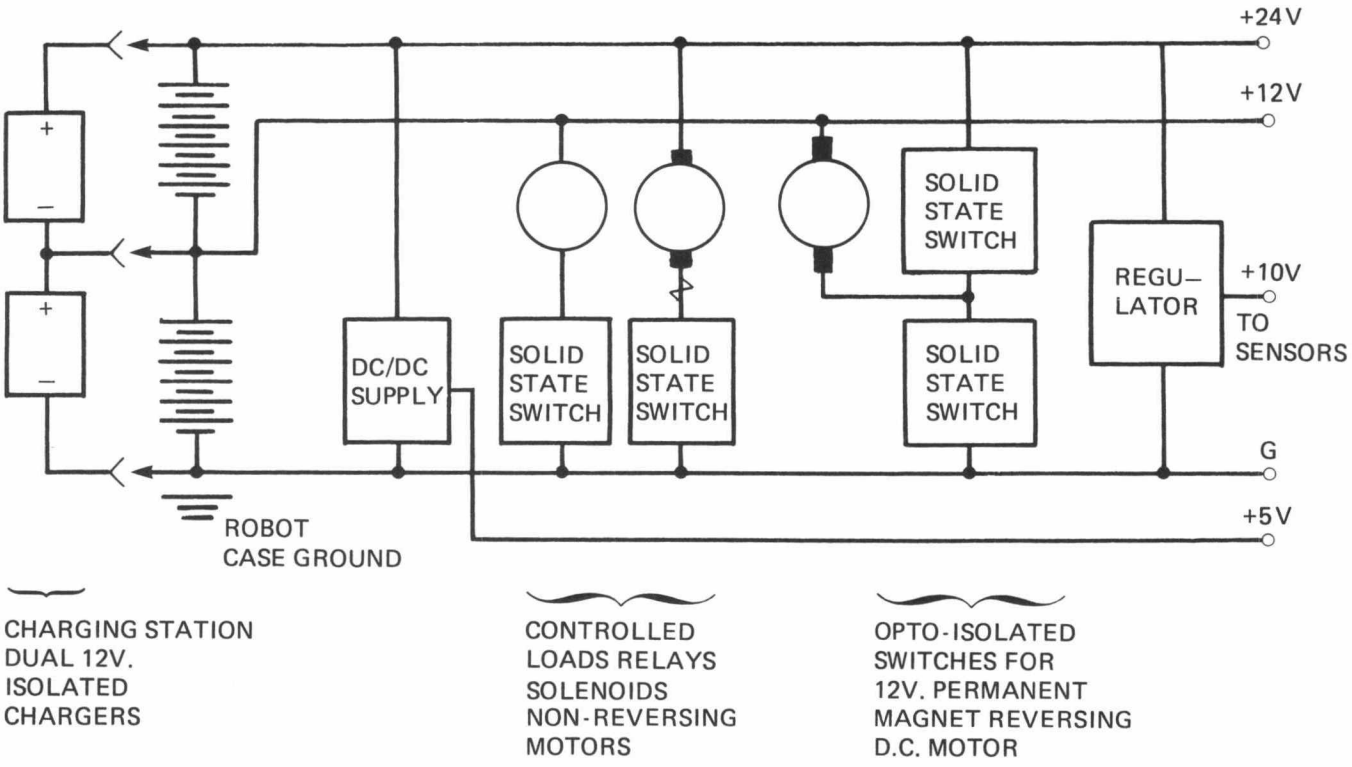


Figure 1. Typical power circuits for a 24 Volt D.C. home robot.

would control the robot from station to station in the house. The humans would soon learn to keep the robot paths clear of obstacles. Proximity sensors on the robot and additional programming would be needed to avoid or remove any remaining obstacles, when required.

Removable trays would be provided to facilitate carrying things and to reduce the number of trips required. Difficult-to-handle items, like silverware, might be handled and stored between use on a special tray.

POWER SUPPLY

The use of two 12-volt automobile batteries (or similar batteries designed for deep cycling) and available aircraft/automobile accessory motors and power semiconductors makes an attractive experimental power supply. The current drawn from the batteries would be 10 to 15 amperes when traveling and the batteries would be cycled to a fairly deep discharge once a day. Such usage would probably require yearly replacement.

If a battery-powered design is planned, space can be provided for Marine deep cycle designed batteries. A good size is 11 inches long by 6-3/4 inches wide by 9-3/8 high to top of terminals. These batteries weigh 45 pounds each and will furnish 25 amperes for 136 minutes (reserve capacity rating). Two of these 12-volt batteries will furnish about 1000 watt-hours with a good margin of capability. This would correspond to an average loading of 200 watts over a five-hour work period. These batteries ought to give a long service. Other batteries could be used on initial experimentation until the robot is to be put into service. There are also problems of safety with respect to the generation of hydrogen gas at high charge rates or overcharge and corrosive fumes from the electrolyte. Sealed cells may overcome these problems.

Most motors on the robot will require reversing. Permanent magnet small D.C. motors are available with good characteristics for most uses and save the energy normally needed for field excitation. A conventional forward-reverse relay circuit could be used with 24-volt motors. If a solid state control is desired, it should be possible to develop a circuit for control of 12-volt motors which use one battery for each direction of rotation. The propulsion motors would mostly be used in the forward travel direction. The motors on the left side could be wired so as to propel forward on the battery which the right side motors use for reverse travel and thus equalize battery drain. Such a circuit could use two power transistors and would require two opto-isolators for each motor. This scheme would also require dual-isolated charger circuits in the charger station. Figure 1 shows typical power circuits for a battery-powered home robot.

For a working home robot, 60 hertz power from the public utility should be more satisfactory.

An overhead cable of light-weight construction is suggested. This cable would hang above the level of floor lamps and below the level of ceiling-hung lamps. It would be above the level of seated persons but could interfere with the taller walking persons. Its jacket would be white or other highly visible color. It would be held taut or taken up by torque motor, and care would be taken to assure longest possible life for this flexible cable. The cable would operate from a 230-

volt 15-ampere outlet and would be about 75 feet long to reach any place in the working area. The cable would slide upon little hook-like supports placed in doorways or the corners of passageways where the cable would need support. For the A.C. powered robot, A.C. motors would be used to drive major loads. Rectified power supplied would be used for small D.C. motors and solenoids. Suitable separate power supplies would be used for electronics. With a 115/230-volt power supply at the robot, it would also be convenient to supply needed portable appliances from a receptacle on the robot. The overhead cable would also be useful as an aid to navigation.

NAVIGATION

Fixed stations may be selected as starting places for the robot to travel to for portions of each task. Many tasks may be completed with little movement of the robot, provided the arms are telescopic. In a typical 8-room home, 48 stations may be sufficient. Navigation has to take the robot through doorways and avoid obstacles such as fixed furniture. A proposed method is to use one sensor for the length of the cable, a second sensor for reading the horizontal angle of the cable to the earth's field, and a third sensor for reading compass direction of the wheeled carriage as determined by the horizontal angle of the earth's field to the carriage directional axis.

The propulsion and steering of the wheeled carriage would be designed so as to perform with good repeatability. The arrangement shown in Figure 2 provides for ten modes of movement:

1. Forward
2. Backward
3. Sideways to the right
4. Sideways to the left
5. Forward right turn
6. Backward right turn
7. Forward left turn
8. Backward left turn
9. Rotate carriage right about center of base
10. Rotate carriage left about center of base

(Centers for all turns are 21 inches to right or left from the center of the carriage).

Preferable definite stops on all steering mechanisms would aid repeatability. The command to start or finish a turn could be executed any time except when moving in modes 3,4, and 9. The time required for steering the wheels would accommodate many kinds of program control, such as making small offsets from a desired path, using time of application as the control of steering. The first four modes allow for travel into very restricted places. For modes 3 and 4, the steering mechanism positions the wheels so that the effective width of the robot on the floor is smaller. This reduces the stability, and programming would need to consider this in positioning and loading the arms. The restriction of turns to a 21-inch radius in conjunction with a maximum velocity of one foot per second would result in a centripetal acceleration of the center of mass of the robot of 0.57 feet per second. This would result in an outward sliding force at the wheels of about 5.33 pounds, which is quite tolerable. If the center of the mass were 50 inches above the floor, the tipping moment would be such that it would take an additional force of about 47 pounds applied at the 50-inch level to start upsetting the robot, provided arms and their loads are symmetrically located to the the robot body. The physical limitiations on speed and radius of turn protect against this kind of upset due to machine or program errors.

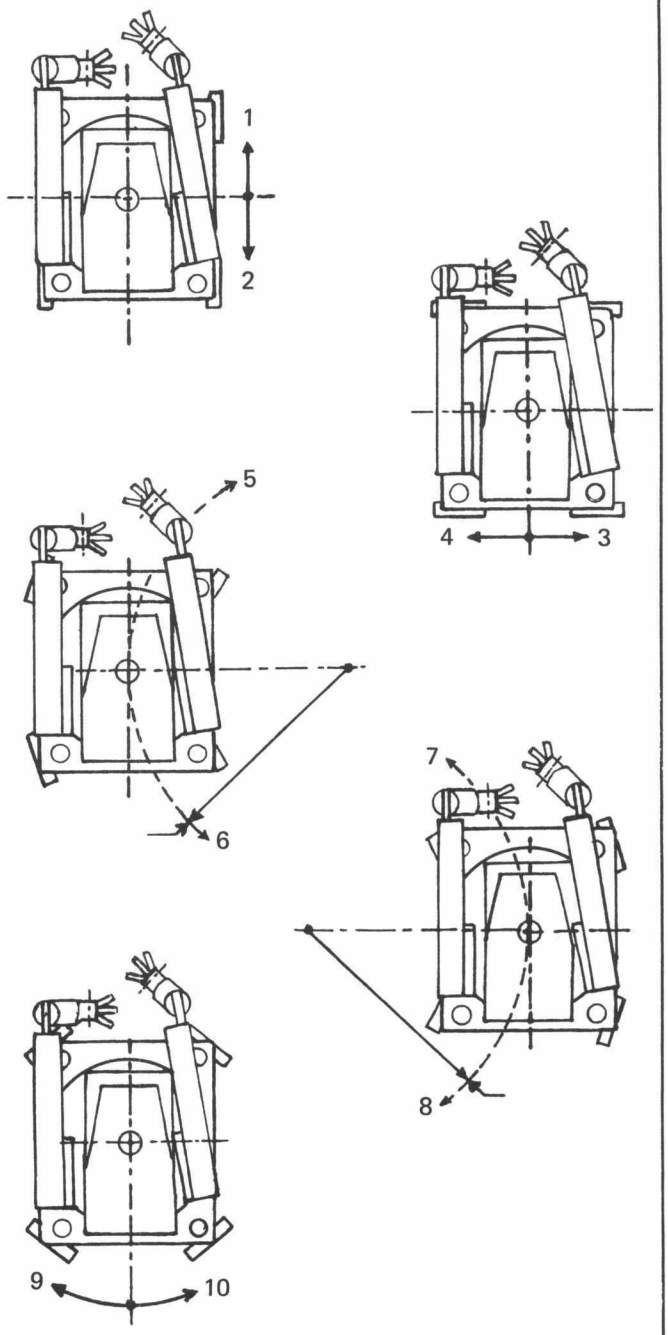


Figure 2. Modes of motion have definite steering positions.

The steering pivot is offset from the center of the tire width by 1¾ inches. When any change in steering mode is programmed while the robot is not in motion, weak forces can be applied to the propulsion motors in suitable direction for each wheel

to cause the wheel to rotate as the steering power moves the contact point with the floor in a 1¾ inch radius circle. This arrangement reduces scuffing of rugs or marking of hard floors. The wheel motors, whether hydraulic or electrically-powered, require individual controls so that this can be programmed.

ARMS AND COLUMNS

The fingers and arms are the essential functional part of the robot. The writer proposes a small human-like manipulator[1] with four fingers, to be mostly hydraulic-operated. Rotation in the forearm and upper arm would be by electric motor. In both forearm and upper arm, the rotating joint would be next to the elbow joint. The fingers would have paint[2] and film pressure sensors[3] in a pattern to furnish some information about the object being grasped. A gum elastomer rubber glove would cover and protect the parts. Miniature solenoid-operated valves would be located on a bracket at the end of a telescopic arm nearest the manipulator. Only one pressure and one return hose need be carried through the telescopic arm. The telescopic motion is in a horizontal direction only, so that no lifting is done and the motor drive can be sized to overcome friction and move a carpet sweeper on the floor. The complete assembly moves vertically on a column and a spring reel type counterbalance balances the weight of the arm assembly. The column swivels through about 97 degrees. This allows the two telescopic arms to move from about 7 degrees toward the central body axis in front of the robot to straight out sideways from the body.

For hydraulic operation, a fluid mixture of glycerine and approximately 50% water is suggested as suitable. If a filter cleans out metallic wear particles, this fluid should be reasonably clear. If leakage or accidental spills occur, the glycerine mixture is able to be cleaned up with water and conventional cleaning methods. A common reservoir and pump system can be used for both arms. This system can also be sized to perform other movements. The problem here is to obtain small solenoid-operated valves suitable to this application. This system could also provide propulsion power. Here again it is difficult to secure suitable propulsion hydraulic motors. The central hydraulic system would also be suitable for the steering system as it could move a piston to fixed stops to accommodate the various modes of steering.

The rotating parts of the cable take-up system on the A.C. powered robot would have two slip ring assemblies. One connects from the cable reel to the moving assembly and the other connects from the moving assembly to the robot body. Another rotation of parts is the rotation of the turntable on the carriage. This should be limited to 370 degrees; a spiral cable can accommodate that motion.

Similar limitation can be placed upon the rotation of the upper arm and forearm rotations. Cable retraction to accommodate the column vertical motion and telescopic arm motion can be accommodated inside the body (see Figure 9). The cable would contain two small hydraulic plastic tubes and electrical wires required from small motors, solenoid valves and sensors. The cable for each arm would be taken up in the narrow space alongside the main electronics package.

THE ROBOT EYE

This is a general purpose light-activated sensor and associated computer system. Much research is in progress on the general problem of recognition and control. The input to the information processing and decision process is usually a digitized output from a television-like rectangular scan. Apparently a large capacity memory and a very fast computer is needed by this method. Since the home robot will operate in a known environment and does not require capability to operate to close limits, a less complex input would enable a very useful robot to operate. Linear arrays of 32 or 64 elements are available. These could be used with an electromechanical shift from a vertical to horizontal position. Another method would be to build a special array arranged to detect clear openings, corners or linear features. Such an array might consist of sixteen photodiodes, as shown in Figure 3. These diodes are in TO-18 cases and present a plastic lens to concentrate the light from the camera lens upon a one-millimeter- square sensitive area. A hole in the mounting plate of 0.104 inches diameter over the diode is the effective size of the element. The small number of elements would facilitate standardization of outputs and would permit a high rate of computer processing needed for real time control. Working distance of 20 inches to 60 inches could be obtained by two lenses and a solenoid-operated mirror. Resolution would range from four-tenths inches at 20 inches distance in the center of the array to about 8 inches at 60 inches distance at side of scene.

Color separation filters should be available by electromechanical means to enable sorting of clothes by color and to increase contrast when necessary. The robot could also identify the room it is in by wall color or other unique visual features.

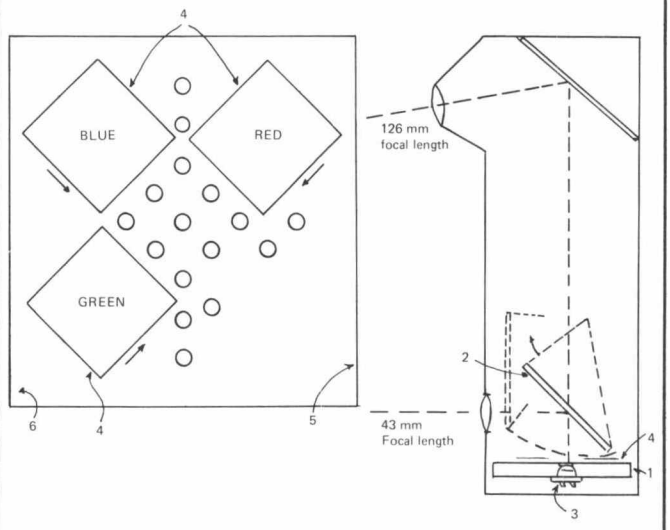


Figure 3. Robot eye.

1. Array mounting plate
2. Swinging mirror
3. Typical photo-diode
4. Color separation filters for 9 central photo diodes
   Two filters may be adequate
   minus blue and minus red
   (Wratten No.12 and 43)
5. Top of scene
6. Bottom of scene

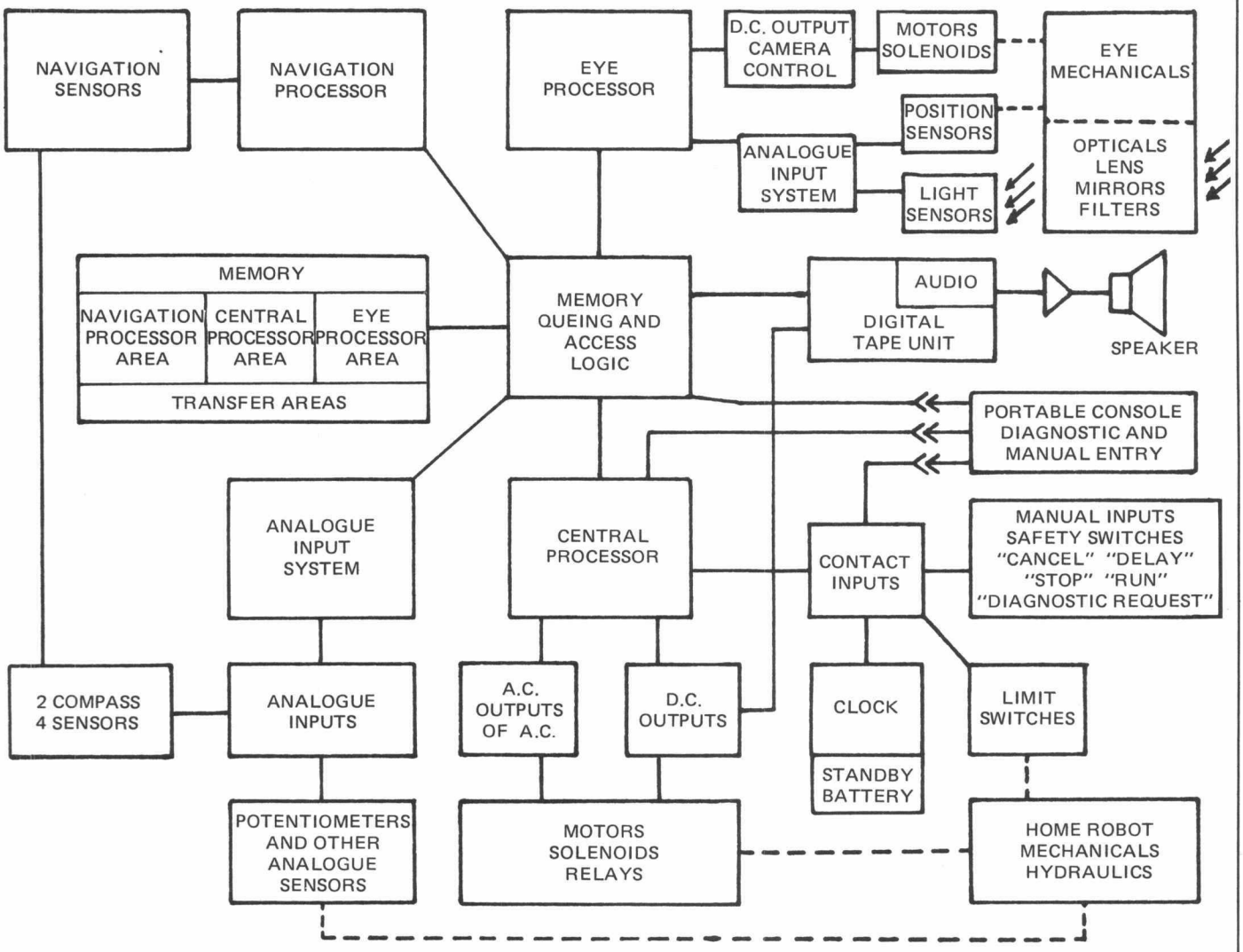


Figure 4. Computer control

Figure 4 is a block diagram showing the computer control of either the overhead cable-powered model or the battery-powered model having an inertial navigation system. A large working memory is shown which may be accessed by various peripheral units. These are analog input unit, eye processor, navigation processor (if inertial), tape unit and portable manual-input diagnostic unit. Since the analog unit will furnish 12 bits, the central processor should be of 12 bit capability or more, but a 16 bit microprocessor would be desirable for reasons of large direct addressing capability and availability of extra bits for setting flag bits along with data. The eye processor could be an 8 bit microprocessor. The navigation processor, if included, should be a 16 bit or longer word length microprocessor.

The analog input system, the eye and navigation processors should have available restricted address areas of memory which are common with part of the central processor and are used for transfer of information either to or from the central processor. The tape unit should have access to the entire memory and have a provision for 'read from' as well as 'write into' the memory. Typically, it would only write into memory whenever a new sub-program is called up. The read from memory (and write a tape) mode of operation would be used for diagnostic and program development. Each processor would also have a large area of memory for its sole operating use (except for tape unit access). Although it is not possible to forecast program requirements, a tentative choice of memory would be 8 K 16 bit words of core memory. The chosen memory had 650 nanosecond cycle time with 300 nanosecond access time and has facility to retain data when power is lost. A similar size semiconductor memory with suitable battery to retain memory would be acceptable and somewhat faster. There should be a separate digital clock circuit with a dedicated battery. Access to its reading would be via the contact input system. This true time clock in conjunction with the retention of data by the memory would permit programs which would effect recovery from power outages or other interruptions to the normal running programs.

The 12 bit analog-to-digital convertor of the analog input system requires 200 microseconds to digitize. Allowing for setting time and address times in the solid state multiplexer, the system could operate at 400 microseconds per input point. Typical electromechanical devices, such as solenoid valves, relays and small drive motors are slow in response and control within 5 to 8 milliseconds would be refined as the robot justifies. At the one foot per second travel speed the robot would move 0.096 inches in 8 milliseconds. With a 75 foot long cable, whose length played out is measured by a 12 bit A/D converter, the least significant bit corresponds to 0.219 inches of cable length. If the maximum unsupported length of the cable was 35 feet and the 12 bit analog system reads 4096 bits for 180 degree angle, then the least significant bit corresponds to 0.161 inches of circumferential travel, provided the system is linear. The analog input system could read 20 inputs in the 8 milliseconds. If 20 devices are in simultaneous operation, the central processor would need to do a compare and possible branch to output every 400 microseconds. since the processor would also have other programs, time interrupts for call back, housekeeping programs, interleaving of tasks and control of tape unit functions, it should perform the compare and branch to output as a short sub-program in 200 microseconds or less. This could be accomplished by most processors.

With the shared memory system shown, a simple hardware logic system should allow repeated access to the memory by the last user without delay. However, if one or more other units, not the most recent user, made a memory access request, the logic would step around a ring of the 6 users to give memory access to each in turn. This would insure memory access within 6 memory cycles for any unit. The critical user would be the tape unit. In the case of the chosen memory having 650 nanosecond cycle time and allowing 100 nanosecond logic delay with the six possible bidders for memory, a delay of 4.5 microseconds is possible for any one user. The tape unit should not run faster than can be accommodated to avoid dropout from this source.

ALTERNATIVE NAVIGATION CONCEPTS

The compass system consists of a compass affixed to the base carriage unit. This indicates direction of travel. A second compass is affixed to the cable takeup arm. This indicates the direction of the cable. Its readout would need to be interpreted by reference to what room of the house the robot is in. The hardware of each compass could be two Hall generators positioned at 90 degrees to each other. Another type of hardware would be a conventional floated magnetic compass whose card is a gray scale with no discontinuities. This would vary from white to black in 180 degrees and return to white in the next 180 degrees. Optical readouts would be stationed 90 degrees apart and their amplified outputs would be connected to two inputs of the analog input system. For either type hardware, the program being run would select the preferred sensor whose output is neither going through zero or through maximum at the angle of decision.

It would not be necessary to exactly calibrate or have a linear output to angle rate on the compasses. Stability and repeatibility are the desired attributes. They would be used mainly in reference to the particular house floor plan. Once all the needed robot stations have been placed in memory, the robot memory need only keep available the room number (or area number in case of L-shaped rooms) to be able to restart its navigation anyplace in the working area without human assistance.

Another attractive possibility would be to use an inertial navigation system. This could be used for a battery-powered robot or for a cable-powered robot where the cable was used for power only and not as a navigation input. The limited acceleration and velocity and the known long periods of no movement will assist accuracy by stopping all integration and resetting the velocity integrators to zero each time the robot stops. Each time the robot returns to its normal resting or battery recharging place the navigation computer would stop all integration and restore the computed location numbers in memory to the exact starting numbers desired. The sensor can be a simple suspended mass type with force sensors in two axes. There would be no need for gyroscopes in any sensors, so the system could be restarted after an shutdown without any alignment requirements.

The forces to be measured are small but, with known propulsion characteristics of the robot, they are limited. The inertia sensor will be mounted on the base carriage. A compass sensor will also be mounted on the base carriage. For inertial navigation the compass sensor and circuits must be calibrated as the navigation computer will need to use trigonometric functions to calculate the components of motion.

With a dedicated navigation processor and timing inputs from a crystal, the force that sensors read would be used in conjunction with the heading compass readouts to obtain vector forces in the direction of the coordinate axes of the system of navigation being used. These would be digitally integrated at short intervals to obtain a number proportional to velocity. Similar integrations will obtain numbers proportional to distance which become the location of the robot. If the scaling is satisfactory, these may be used in raw form for station locations by the programmer or may be converted to conventional measure, such as feet or inches for use. If done this way, the numbers to be entered for station locations could be determined by tape measure.

Another possibility for a navigation system suitable especially for a battery-powered home robot is to use a rotating beacon and receiver sensor on top center of the robot body. This is the space provided for the cable take-up system in the AC powered concepts. The system might use ultrasonics or sound waves or light. It might employ targets placed on the walls in some useful pattern with the least noticeable intrusion to the room appearance. With a height limit of 78 inches the beam could be parallel with the floor in a range of 72 to 78 inches. This would avoid interruptions except by persons standing more than six feet tall.

To get usable data for the central processor, the navigation processor for the case of three targets in each room could measure the angle between targets and the compass north reference. With a constant rotational speed of the beacon, these angles could be read out by gating a suitable frequency oscillator into a counter. Depending upon the travel path being used, the angle which is changing most rapidly would be used as the address at which stopping or some steering function is initiated.

If ultrasonic pulses are used with only the walls of the room as targets, counting up to arrival of the reflection would measure the distance to the wall. The minimum distance would be perpendicular to the wall. By selection the wall perpendicular

which is changing most rapidly, the raw distance data could be used as the arrival address to stop or initiate some other function. Such an ultrasonic system would be more accurate if the beacon is not rotating but is maintained generally perpendicular to the wall of interest. This could be done by a digital program which maintained a minimum count by testing small plus and minus increments of rotation of the beacon to hold minimum value. This would maintain the beacon perpendicular to the wall whose distance is being used for the navigation. The particular wall would be selected and changed according to need by the main navigation program.

PROGRAMMING

The proposed robot could be controlled by various program concepts, and development could proceed by various routes. The following is offered as a possible simple approach to getting started.

Based upon the house floor plan and locations of semi-fixed furniture, such as tables, beds, divans and fixed bath and kitchen equipment, a selection of robot operating stations would be chosen. The robot parking place when not in use and its cable attachment to the cable protective unit are also chosen. If the robot is battery-operated, this should be the location of the charging station.

A list would then be prepared of all needed travel sub-routines. These navigation programs would operate by sensing the cable length paid out and compass angles. The robot should arrive at a station with only a few inches error (about three inches). For a battery-operated robot, inertial navigation with suitable programs, as discussed under navigation, might bring the robot on the desired station with even smaller error. At the conclusion of the trip the eye sensor would look at the known scene on the fixed furniture (for example, a corner of a table) and correct its position by eye sensor.

The robot then would go to the task at hand program, and other sensors would be used as needed. It would be necessary for the robot figures to have sense of touch sensors as some things would be located by touch, and these would also be needed to avoid damage to fragile objects.

Many of the tasks would require the developmentof standard programs, which could be used in any home, for example, making a bed or place-setting a table. Where possible, the general sensors on the robot should be used in these task programs with minimal variation from the present human situation and methods. Specialized assistance can be provided where needed. For example, a right, head end, corner marker might be put on sheets and other bedclothing. The marker might be visual for the eye sensor to detect or could be magnetic for use with a magnetic detector. Another example would be to provide a reader for the point of sale Universal Product Code on grocery packaged goods to aid in storing and verification before use of frozen foods and other supplies.

For floor cleaning, the robot could travel to stations located in such a manner that outward sweeps of the cleaning tool from one location would clean the space covered by the robot at another location. A specialized cleaning tool having a proximity sensor on a very pliable nozzle and/or brush could sweep outward until stopping by limit stops on telescopic arm or by proximity sensor. The arm would then be retracted. The robot body could then swivel a program controlled angle from ten to about twenty-five degrees and sweep again so that the sweeps overlap. It would be most convenient to power the vacuum cleaner from the receptacle on a cable-powered robot. This floor cleaning concept can facilitate cleaning under tables and beds with suitable cleaning tools.

Programs for meal preparation are probably the area of programming of greatest variety and which are most essential to the success of the home robot. Homemaker activity studies have shown daily elapsed time spent cooking from 3 ½ hours down to about 45 minutes. Actual working time probably averages 1-½ hours, but the process of going back to the task from time to time is the measure of inconvenience to the homemaker. Programs could be as simple as that to remove a "TV" dinner from the freezer and put it into a microwave oven. More elaborate programs could provide for nearly any menu by calling up the individual item programs from bulk memory. The starting time of these programs would then be calculated so that all dishes on the menu are ready to serve at the desired meal time. These programs would then be merged; if any time conflicts exist, they would be resolved by minor adjustments in starting time or temperatures used. The resulting running program would be specific for the menu and number of place-settings of the day.

HUMAN INTERFACE

It is most convenient for the human to have access at the rear of the electronics enclosure. The magnetic tape deck or cassette units could be on the upper portion with receptacles and switches below. One concept would be to plug in a portable console, like a hand calculator. Since automatic operation is a major advantage, programs ought to require the least possible human intervention. In addition to safety bar switches, a switch to omit entirely a program that has started and a switch to delay for some interval (such as 20 minutes) a program which has just started would be desirable. At least in the initial development, voice inputs seem unnecessary. Recorded voice outputs could be useful.

The tape unit capability could include several voice messages selectable under computer control. These would include warnings issued when a robot is about to move or do something which humans may wish to cancel, such as trying to make up a bed having human occupants. It should also include some diagnostic comments which would only be available on demand. For example, if the homemaker rushes home at the programmed hour and finds a bare table, he/she could press the "Diagnostic" button and the robot would voice the message, "Dinner will be one-half hour late because of power outage."

CONCEPT SUMMATION

The home robot becomes a group of several active systems acting cooperatively. Looking toward a practical, useful working robot, it becomes evident that many systems and programs are necessary to obtain the "quantum leap" in benefits of a robot in the home. Simplification is desirable, but the robot is necessarily a complex machine. It would be well to avoid any "science fiction" aspects. Although speech inputs to the robot are a technical possibility, the chance for human errors increases risks with no large benefit. A remotely controlled automaton or tele-operator would not be a great benefit, as the principal advantage of a robot in the home is that it should not require human attention. Referring to Figure 4, at least three microprocessors can be used, and 'read only'

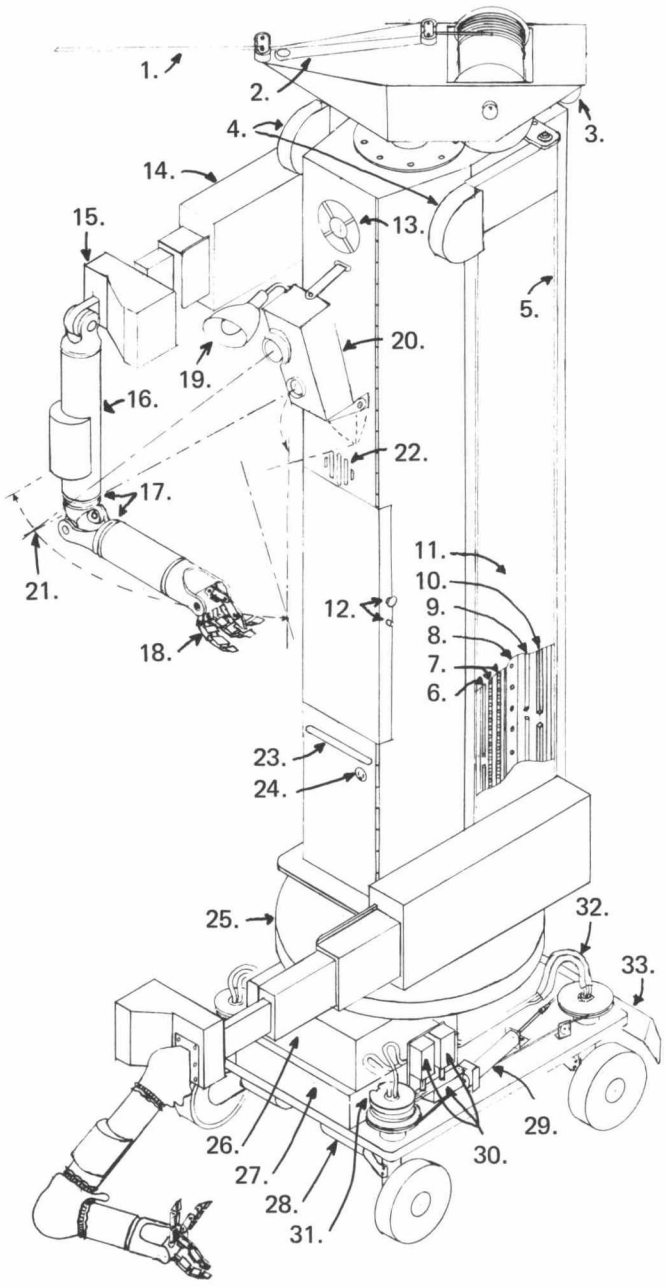


Figure 5. A.C. powered home robot.

1. Power Cable
2. Level Wind Mechanism
3. Torque Motor and Brake
4. Spring Counterbalance for Arms
5. Column Assembly
6. Counterbalance Cable
7. Chain Drive for Vertical Motion of Telescopic Arm
8. Hardened Steel Track
9. Electrical and Hydraulic Cable to Arm
10. Square Drive Shaft for Telescopic Motion
11. Protective Shutter Curtain
12. Attachment Point for Trays
13. Ventilation Fan
14. Telescopic Arm
15. Solenoid Valve Package
16. Upper Arm
17. Rotary Motion Joints
18. Fingers
19. Lamp
20. Eye Assembly
21. Crossover Point For Lenses
22. Speaker
23. Emergency Stop Bar Switch
24. Power Outlet
25. Turntable
26. Solenoid Valve Package
27. Hydraulic Reservoir
28. Hydraulic Motor
29. Steering Cylinder
30. Streer Select Solenoids
31. Steer Position Lock System
32. Hoses to Propulsion Motor
33. Proximity Sensor as Required

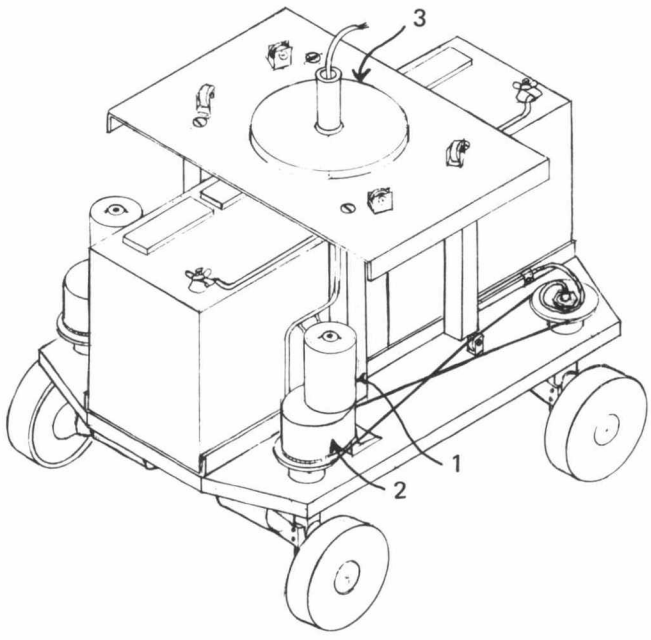


Figure 6. Carriage for D.C. powered robot.
1. Steering Motor
2. Mechanical Stops and Switch Unit
3. Bull Gear

memory may be used to good advantage for some of the memory block shown. For example, two buffer registers might be used, one in each direction at each interface, to provide isolation between the I/O busses of the eye, navigation and central microprocessors. This could serve to limit the effects of a failure and, by temporarily holding information, would result in less time delays in the processor programs. It would be desirable to have some isolation of the various systems to aid recovery after some failures. The desired goal is to be fail-safe but also to have programs to effect recovery in the largest possible categories of failure.

A complete robot of the AC-powered type is shown in Figure 5. The carriage covers are removed to show the hydraulic steering concept. A separate steering unit is provided for the wheels on each side of the robot. The carriage unit for the DC-powered type robot is shown in Figure 6. The turntable, which rotates with the electronics enclosure, is approximately three inches higher than its position in the AC version. Because of the higher speed of small DC motors, they are lighter than AC motors. That fact plus the need from some DC power supplies and the cable reel on the AC version results in about equal weights of the two versions despite the battery weight. The beacon for the navigation sensing on the DC-powered robot can project a shorter distance above the electronics enclosure than the power cable reel assembly, so that the overall height of the DC version will be the same as the AC version.

The writer has worked on this general concept to build an operational model and has sufficient construction finished to be a mock-up of the robot, as shown in Figure 7.

The writer has been limited to design and building a home robot of the general concept presented here. Other concepts and designs need investigation. As a criterion of success, completeness of approach appears most important to a good demonstration. Hardware and software methods of handling failures are very important, as the home robot's most valued contribution to life will be performed when no human backup is available. Mechanically, a light-weight design of moving parts, especially any long arms, is most essential. Necessary heavy components, such as batteries, transformers, hydraulic reservoirs, and large motors should be located low and centrally to help stability of robot.

CONCLUSION

A home robot is now practical with the use of available technology. There is a large potential field of use for the robot. The design, production, and programming of home robots could become a sizeable industry of importance to the computer society. Much mechanical hardware development needs to be done to get started.

Present microcomputer hardware development, as regards to microprocessors and analog input systems, is adequate. This article has outlined a basic concept, which has sub-assemblies that could be individual developments. All sub-assemblies are listed in detail in the Appendix. Tentative interface definition would make practical the consideration of alternate designs of the sub-assemblies. Since the microsystems group brings together the various subsystems for real time acquisition of data and control, they would be the logical choice as lead group on this project.

A cooperative effort by individuals or institutions could accomplish the mechanical developments adequate to get started. The home robot should prove to be a very broad field for additional developments and improvements of interest to mechanical and computer engineers.

As a new field, program developers can develop from the very beginning program basic concepts and strategy, which borrow from the present mature business and scientific programming knowledge without needing to conform to prior art.

References

1. "Hydraulic Muscles Begin to Flex Artificial Limbs," *Product Engineering*, Vol. 39, No. 5, Feb. 26, 1968, pp. 50-53.
2. Pressure Sensitive Paint, made by Clark Electronics Laboratories.
3. C.A. Rosen and D. Nitzan, "Use of Sensors in Programmable Automation," *Computer*, Vol. 10, No. 12, Dec. 1977, pp. 12-23.

Bibliography

Young, John F., "Robotics," John Wiley & Sons, New York, 1973.

Appendix: Development, Sub-Assemblies and Interfaces

The home robot may be readily divided into several logical sub-assemblies, within which considerable independence of design and latitude for innovation would be possible. These sub-assemblies are:

1. Wheeled Carriage
   - Propulsion, brakes, steering systems
   - Batteries, if battery-powered
   - Heavy power supplies
   - Hydraulic power units
   - Bull gear and central hollow shaft
   - Bearings for turntable
2. Turntable Base and Electronics Enclosure
   - Turntable assembly
   - Motor and drive and pinion for turntable rotation
   - Means for turning columns and position sensors
     - Motors and gear trains
     - Hydraulic cylinders and system
3. Right and Left Columns and Arms and Manipulators
   - These need not be mirror-image designs. It may be determined that some differences will be advantageous. Since the effective output of the entire home robot occurs at the fingers, their capability and dexterity as enhanced by the fore and upper arms should be given priority over other sub-assemblies. These sub-assemblies contain:
     - Vertical column, consisting of:
       - Upper pivot journal pin
       - Column enclosure and structure
       - Column track and rollers
       - Retraction pulleys and system to manage electric/hydraulic cable from electronic enclosure to upper arm
       - Spring counterbalance to reduce load on raise/lower arm assembly system
       - Gear motor and roller chain or other system to raise and lower arm assembly
       - Gear motor and power train or other system to operate telescopic motion of arm
       - Application of position sensors to vertical and telescopic motions
       - Protective metal curtain and parts to allow vertical motion
       - Lower pivot journal pin
     - Telescopic arm
       - Power train and cables to extend and retract arm

Provision for cable to upper arm and hydraulics
Safety covers
Provision for attachment of upper arm
Provision for attachment of hydraulic package

Manipulator consisting of:
Hydraulic package of solenoid valves
Upper arm
Forearm–wrist finger assembly
Position sensors
Finger touch sensors
Enclosures–elastomer dust protectors and gloves

4. Power Cable Reel Assembly for AC Power Version
   Torque motor and gear train with brake
   Level wind mechanism and provision to sense length paid out
   Two slip ring assemblies
   Provision for mounting compass (directional sensor)
   Enclosures
5. Battery Tray and Ventilation Facilities for Battery-powered Version
6. Personnel Protective Unit for A.C.-Powered Version
   This is not on robot and is mounted at starting location of cable. It connects between cable and building electrical system.
7. Charging Station for Battery-operated Version
   Two isolated 12-volt battery chargers
   Protective equipment
   Connectors and cable suitable for the robot to plug itself into charging station
8. Electronics and Controls
   This group would coordinate the entire development and would design or select all electrical circuits in cooperation with programming considerations and mechanical possibilities.

   Electronics
   Central processor
   Eye processor
   Navigation processor
   Tape unit or units
   Input and output systems
   Sensors
   Eye unit
   Navigation (as required)
   Compass direction sensors
   Inertial sensor
   Zero speed sensor
   Beacon sensor
   Proximity units
   Pressure and displacement sensors
   Diagnostic and manual entry systems

Continued from pg. 18

```
514       FOR J=1 TO LEN(X$)
516         W$=SST(X$,J,1)
518         CHANGE W$ TO W
520         IF W(1)<128 THEN 524
522         W$=Q$(W(1)-128+1)
524         IF J>1 THEN 530
526         S$=W$
528         GO TO 532
530         S$=S$&W$
532       NEXT J
534       GOSUB 900
536       PRINT #2, S$
538       GO TO 510
540       REM
542       PRINT "DECOMPRESSION COMPLETED; OUTPUT IS ON FILE ";F2$;"."
544       STOP
600 REM   ** LINE DECOMPOSITION ROUTINE **
602 REM   .. DECOMPOSES S$ INTO STRINGS Q$(1)..Q$(Q) ..
604 REM   A STRING IS DEFINED AS A CONTIGUOUS GROUP OF ALPHANUMERIC AND/OR
606 REM   PERIOD CHARACTERS, INCLUDING ANY APPROPRIATE SINGLE TRAILING
608 REM   DELIMITER. A CONTIGUOUS GROUP OF TWO OR MORE IDENTICAL DELIMITERS
610 REM   IS ALSO REGARDED AS A STRING. IN THIS SUBROUTINE, THE ENTRY STRING S$
612 REM   IS PROGRESSIVELY ERODED AS EXECUTION PROCEEDS.
614 REM
616 Q=0
618 L1=LEN(S$)
620 FOR L2=1 TO L1
622   W$=SST(S$,L2,1)
624   IF W$="."       THEN 640
626   IF W$<"0"       THEN 648
628   IF W$<="9"      THEN 640
630   IF W$<"A"       THEN 648
632   IF W$>="["      THEN 648
634   IF W$<="Z"      THEN 640
636   IF W$> "^"      THEN 640
638   GO TO 648
640 NEXT L2
642 S$=S$&" "
644 L1=L1+1
646 L2=L1
648 X$=SST(S$,1,L2)
650 IF Q<1 THEN 660
652 W$=SST(Q$(Q),1,1)
654 IF X$<>W$ THEN 660
656 Q$(Q)=Q$(Q)&X$
658 GO TO 664
660 Q=Q+1
662 Q$(Q)=X$
664 IF L1-L2<1 THEN 670
666 S$=SST(S$,L2+1,L1-L2)
668 GO TO 618
670 RETURN
672 REM
674 REM
700 REM   ** STRING INSERT ROUTINE **
702 REM   INSERTS STRING S$ INTO THE STRING TABLE OR AMENDS COUNT OF ITS
704 REM   OCCURRENCES TO DATE.
706 IF T1<1 THEN 718
708 FOR T3=1 TO T1
710   IF S$<>T$(T3) THEN 716
712   T(T3)=T(T3)+1
714   GO TO 724
716 NEXT T3
718 T1=T1+1
720 T$(T1)=S$
722 T(T1)=1
724 RETURN
726 REM
728 REM
800 REM   ** STRING TABLE SORT ROUTINE **
802 REM   STRING TABLE IS SORTED ACCORDING TO THE CURRENT SAVINGS-POTENTIAL
804 REM   FUNCTION FOR EACH ELEMENT THERE-IN.
806 REM
808 IF T1<2 THEN 838
810 T2=T1
812 F2=0
814 T2=T2-1
816 FOR T3=1 TO T2
818   IF FNS(T3) > FNS(T3+1) THEN 834
820   T4=T(T3)
822   W$=T$(T3)
824   T(T3)=T(T3+1)
826   T$(T3)=T$(T3+1)
828   T(T3+1)=T4
830   T$(T3+1)=W$
832   F2=1
834 NEXT T3
836 IF F2 > .5 THEN 812
838 RETURN
840 REM
842 REM
900 REM   ** TRAILING BLANK SUPPRESSION ROUTINE **
902 REM   .. PROGRESSIVELY DELETES ANY DETECTED TRAILING BLANKS FROM S$
904 REM   (ENSURING THAT ZERO LENGTH LINE IS NOT PRODUCED) ..
906 W$=SST(S$,LEN(S$),1)
908 IF W$<>" " THEN 916
910 IF LEN(S$)<2 THEN 916
912 S$=SST(S$,1,LEN(S$)-1)
914 GO TO 906
916 RETURN
918 END
```

A Fast Repertory Telephone Dialer Program

BY S. CASH OLSEN
Central Applications
Signetics Corporation
811 E. Arques Avenue
Sunnyvale, California 94086

Dear Suzanne:

I used this program to familiarize myself with language syntax of the Dyna Byte Basic Controller. It is one of the most useful utilities to demonstrate a computer to friends; after all, we can all relate to the telephone. Local and long distance calls are dialed in approximately 2 seconds.

I am a long time computer hobbyist, circa 1973, beginning with Intel 4004 and Motorola 6800. I have a bewildering selection of microprocessors and have written software for all of them. They include 4040, 6800, F8, 8080, Z80, 6502, 2650, 8048 and three bit slice emulators. Total memory exceeds 1/4 Mega Bytes but the largest system has only 32K Bytes. Major problems involve reconfiguration of peripherals to make use of the microprocessor. My "system" is a Z80 from Monolythic Computer Systems on the Intel Multibus.

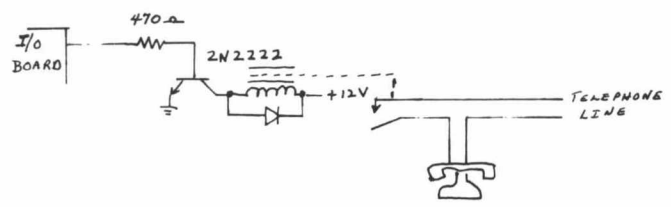
I enjoy Dr. Dobb's *very much and hope you can minimize your advertising while keeping quality of software articles at their current high level.*

Sincerely,
Cash Olsen

Introduction

If you're looking for another neat way to use your system, try this telephone dialer written in ZIBL (Dynabyte Z80 Industrial Basic Language). It's close enough to Basic that only three keywords need to be changed, depending on your system. They are TURNON RELAY, TURNOFF RELAY, DELAY. DELAY in ZIBL is for introducing a programmed wait and has a 5 millisecond overhead plus expression*1 millisecond delay times.

Only the following hardware and wiring connections are necessary:



The relay should be a heavy duty REED type relay with normally open contacts.

About The Program

Line 17 closes the relay so that the phone is operative. Line 30 will skip dialing the area code if > = 999. Line 100 is needed because the telephone dials 10 pulses for an 0 or Operator. Lines 110 thru 150 actually pulse the relay open, then closed again. Line 145 provides for interdigit time.

A Data Table starting at line 1000 contains the name and telephone numbers in your "directory". When asked CALL-WHOM?, pick up the telephone handset, then enter the name. The program will search through your telephone directory and display the number, then dial the phone. You will be able to hear the dialing with this connection.

Little Known Facts

If you have Touch-Tone service, no problem; the phone system allows dial service in these areas with no extra equipment or charge. The telephone has some shocking voltages so make connections using care and good practices.

Try setting up a telephone network with other hobbyists and get that Micro working for you!

```
>>LIST
DIAL
 15 V = 4
 17 TURNON RELAY V
 20 PR "CALL WHOM ";:GOTO 2000
 30 IF A >= 999 THEN GOTO 60
 35 F = A:E = 100:GOSUB 300
 60 IF P > 999 THEN GOTO 20
 65 F = P:E = 100:GOSUB 300
 80 IF L > 9999 THEN GOTO 20
 85 F = L:E = 1000:GOSUB 300
 90 GOTO 20
100 IF D = 0 THEN D = 10
110 FOR N = 1 TO D
120 TURNOFF RELAY V:DELAY 33
130 TURNON RELAY V:DELAY 16
140 NEXT N
145 DELAY 200
150 RETURN
300 D = F/E:F = F-(D*E)
310 GOSUB 100
320 IF E = 1 THEN RETURN
330 E = E/10:GOTO 300
500 END
1000 REM *** BEGINING HERE INSERT DATA STATMENTS ***
1001 REM *** "NAME",AREA CODE,PREFIX,LAST FOUR DIGITS
1010 DATA "MOM&DAD",415,555,1212
2000 INPUT N$,
2005 IF N$ = "STOP" THEN END
2010 DO : READ M$,A,P,L
2030 UNTIL N$ = M$
2031 PR A,P,L
2032 RESTORE
2040 GOTO 30

>>RUN
CALL WHOM ?MOM&DAD
 415  555  1212
CALL WHOM ?STOP
 END
```

A GENERAL PURPOSE DATA COMPRESSION PROGRAM

BY GRAHAM K. JENKINS
Telecom Australia Research Labs
770 Blackburn Rd.
Clayton, Vic
Australia 3168

WHY COMPRESS?

Computer vendors have in recent times come to announce reductions in memory prices with such a disturbing degree of regularity that one feels almost a sense of disappointment if a month elapses during which one does not discover a further announcement. And the reductions cover every conceivable type of memory device, ranging from high-speed primary memory through rigid and flexible rotating memories (and their more recent solid-state equivalents) to the various types of tape unit. In the light of these considerations, one might well wonder at the merits of data/program/text compression.

Whatever may be the magnitude of announced reductions, it is cold comfort for the owner of a recently acquired twin-floppy unit to learn that twice the capacity could have been had for half the price! This article will at least provide you with the means for increased capacity at no cost outlay.

EIGHT BITS IN EVERY BYTE

Various techniques for data compression are in common usage, and the optimum selection in any particular instance depends upon a number of factors such as structure of the source file, available processor memory, and the cost of processor time.

As a simple algorithm, consider the fact that most processors store ASCII characters as eight-bit bytes, even though only seven bits are significant. There is some justification for doing this if a parity bit is included in the eighth position for storage integrity checking. This is not always the case. It will be evident also that storage of one character per byte in this fashion enhances the ease of character access and speed of operation. But if only seven bits are stored for each ASCII character, and the characters are packed tight regardless of byte/word boundaries, it will be apparent that a twelve per cent enhancement in storage efficiency will result.

The algorithm presented here goes a step further in that the eighth bit position is used to define an extra set of 128 "characters" or data-item symbols. According to implementation, an eight-bit byte may be said to represent a data-item symbol if its most significant bit is set (as shown in the accompanying listing), cleared, or of a particular (odd/even) parity; that is the bit is the inverse of what is normally used in the particular computer for character representation.

STRINGS AND SYMBOLS

So what do we do with the extra 128 characters we now have? These symbols are used in the compressed file to represent frequently occurring characters, or strings. Thus the symbol whose hexadecimal value is 80 might be used to represent the string "character" in a condensation of this paper, and

81 (hex) might represent "compression," and so on. The exact translations used are obviously dependent upon the source file content, and an appropriate translation table is included at the start of the compressed file.

The determination of how to best break up each line or block of the source file into appropriate strings for potential compression could require a considerable degree of program time (to say the least!). Our program therefore uses a general-purpose strategy which is applicable for programs, data and text, but which is not necessarily the best strategy in each instance. In this strategy, contiguous groups of alpha-numeric and period characters are regarded as words, and characters which are excluded from this set (such as spaces and arithmetic operators) are regarded as delimiters. A string may be constituted of a word with its trailing delimiter, or of a contiguous group of delimiters. Each line of text is considered (for compression purposes) to terminate in a space. The reader may wish to try variants of this strategy which could be more appropriate to his particular needs.

AN IMPLEMENTATION

The BASIC program accompanying this article was implemented for experimental purposes on a Honeywell 6000 series machine, and may easily be adapted for execution on most BASIC systems by appropriate modification of some of the string handling statements. In particular, it will probably be necessary to replace the "CHANGE S$ TO T" statements which translate the string S$ to an equivalent sequence of integer ASCII character values stored in T(1), T(2), etc., and return the length of S$ in T(0); the converse statement "CHANGE T TO S$" has an exactly converse effect (requiring T(0) to be appropriately set prior to execution thereof). As will be apparent, the character "&" is employed for string concatenation, the length function LEN (S$) returns the length of S$, and the substring function SST (S$, J, N) returns N characters from S$ starting at position J.

Two passes are made through the source file during execution. In the first pass, a table of strings is formed. Whilst it would be desirable to store and accumulate occurrence counts for every detected string, there are obvious limitations on storage (unless the machine has a virtual array capability). In our implementation, up to 500 strings may be accumulated in the table, the least significant 100 being purged every time the table capacity limit is attained; the significance of a table entry is computed from the string length and occurrence frequency, as detailed near the start of the program. Depending on envisaged application, a much smaller table capacity (*e.g.*, 200 with purge of 30 entries when limit is attained) might be appropriate. It should be noted that there is an option to ignore line numbers for compression purposes; this is useful for compression of line-numbered FORTRAN programs or text, where the line numbers might otherwise tend to saturate the table.

At the end of the first pass, all those current table entries (up to a limit of 128) which can produce some file compression are written in descending order of significance to the output (compressed) file.

During the second pass, the source file is copied to the output file using the special character symbols for abbreviation wherever possible. Counts of source and compressed file contents (assuming in our case a one-character overhead at the end of each line) are also generated.

The decompression process is trivial, and involves first reading the translation table from the compressed file, then proceeding through that file expanding each data element symbol as it is encountered (as illustrated in the accompanying program listing).

DOING IT!

The degree of compression attainable in any instance is obviously dependent upon source file content; in practice, compression ratios of 40 percent are not uncommon. The program whose listing accompanies this article is not particularly efficient in terms of execution speed, but this should not bother the average micro-system enthusiast. Much of the execution inefficiency in the program is in fact a consequence of its deliberate construction in a form which is easy to comprehend. Thus a considerable improvement could be effected by inclusion of a more complex sort routine and/or by storing values of the savings potential in the table instead of occurrence counts (thereby averting the necessity for repetitive re-computation). Further improvement would result from compacting expressions such as "K – 1 + 128", and (by use of appropriate in-built functions on short assembly-code segments) simplifying the map-to-upper-case, delimiter-detection and parity-modify/detect portions of the program.

In some computer systems, an increase in compression efficiency and/or execution efficiency might be effected by storing the compressed file in binary rather than ASCII form.

OTHER TECHNIQUES

If your computer cannot be made to accept the extra 128 character symbols created by modification of the parity bit, the algorithm can still be made to work by employing 128 or fewer character pairs (which might, for instance, always commence in the escape code) in place of the symbols, and modifying the savings potential function accordingly.

In some instances, one might wish to compress only files whose characters are all upper-case; a first step in such an instance might be to convert all stored characters to 5-bit (*e.g.*, Baudot code) or 6-bit form.

Where the occurrence rate for certain characters differs significantly from the average (as in plain text), one may wish to define a special variable length code, representing the most common characters by elements of two or three bits length in much the same way as is done with Morse code. It is possible that an algorithm employing such a technique will be the subject of a subsequent article.

ACKNOWLEDGMENT

The permission of the Director Research Telecom Australia, to publish this paper is hereby acknowledged. ⟹

About The Author

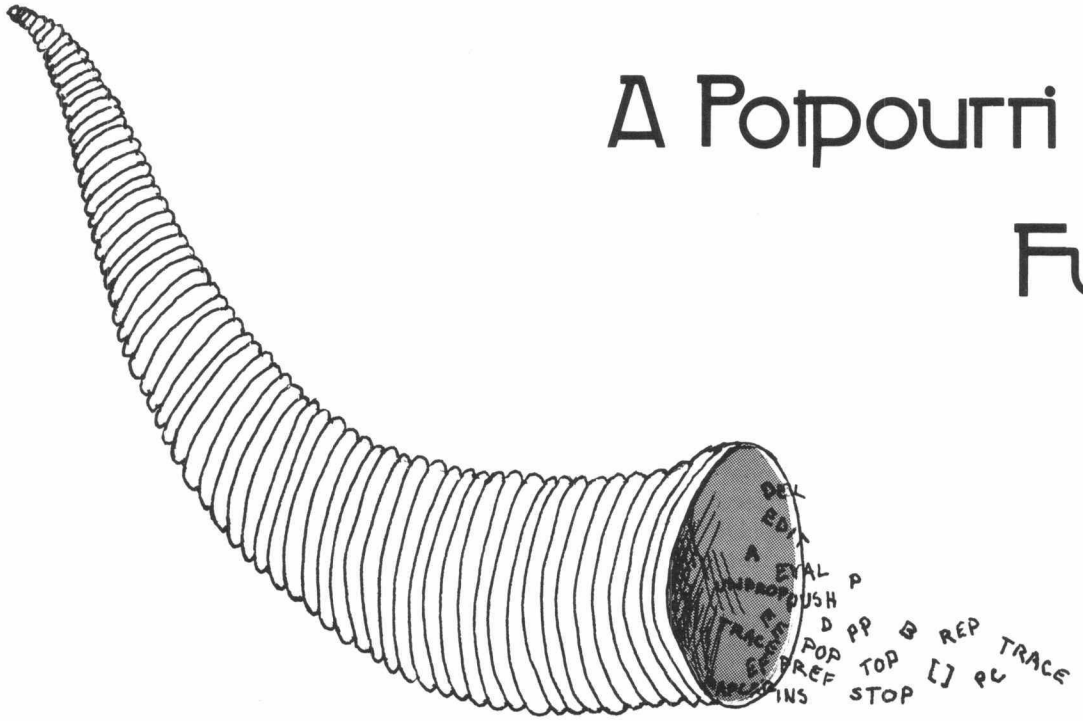
Mr. Graham K. Jenkins is now a technical specialist in Telecom Australia Research Laboratories, where he was until recently Manager of the Computer Applications and Techniques Section. His recent interests have included cross-assemblers, cross-compilers, and simulators for microprocessors. He has also developed minicomputer-based metering equipment for telephone exchanges, and directed the development of an experimental wired-teletext system for assessment in the Australian telecommunications network. His non-professional interests include squash and body-surfing.

LISTING

```
100 REM   LINE-BY-LINE SOURCE FILE COMPRESSION ALGORITHM, IMPLEMENTED
102 REM   BY GRAHAM K JENKINS, MELBOURNE, AUSTRALIA, DURING APRIL 1979.
104 REM
106 REM   128 SPECIAL SYMBOLS (REPRESENTED BY ASCII CHARACTERS WHOSE PARITY
108 REM   BITS ARE SET OR INVERTED - ACCORDING TO IMPLEMENTATION) ARE USED
110 REM   IN THE COMPRESSED FILE TO REPRESENT SELECTED CHARACTER STRINGS.
114 REM
116 REM   STRINGS ARE STORED IN TABLE, WHERE T(J) IS THE NUMBER OF OCCURRENCES
118 REM   OF THE STRING T$(J). Q$ IS AN ARRAY (WITH POINTER Q) USED FOR
120 REM   WORKING.
122 DIM   T$(500), T(500), Q$(80)
124 REM   IF THE STRING T$(J) IS REPLACED EVERYWHERE BY A SINGLE CHARACTER, THE
126 REM   RESULTANT SAVING (ALLOWING FOR ONE COPY OF T$(J) IN A TRANSLATION
128 REM   TABLE) IS :
130 DEF   FNS(J)=(T(J)-1)*LEN(T$(J))-T(J)
132 REM
134 REM   TWO FILES ARE USED - ONE FOR INPUT, ONE FOR OUTPUT!
136 FILES *;*
138 PRINT "SOURCE-FILE COMPRESSION ALGORITHM; AVAILABLE COMMANDS ARE -"
140 PRINT "C,SFILE,CFILE  ..  COMPRESSES SFILE INTO CFILE   .."
142 PRINT "X,SFILE,CFILE  ..  DITTO, WITH LINE-NUMBER COMPRESSION INHIBITED .."
144 PRINT "D,CFILE,SFILE  ..  DECOMPRESSES CFILE INTO SFILE .."
146 PRINT "SFILE AND CFILE MAY BE ANY APPROPRIATE QUICK-ACCESS, SUB-CATALOGUED"
148 PRINT "OR (DEFAULT) TEMPORARY SEQUENTIAL FILES."
150 PRINT
152 PRINT "WHAT IS YOUR COMMAND";
154 INPUT S$,F1$,F2$
156 FILE #1, F1$
158 FILE #2, F2$
160 DELIMIT #1,(CR)
162 SCRATCH #2
164 CHANGE S$ TO T
166 IF T(1)<96 THEN 172
168 T(1)=T(1)-32
170 CHANGE T TO S$
172 IF S$="D" THEN 500
174 Q1=1
176 IF S$="C" THEN 190
178 Q1=2
180 IF S$="X" THEN 190
182 PRINT "ILLEGAL COMMAND .. TRY AGAIN .."
184 PRINT
186 GO TO 132
188 REM
190 REM   ** FILE COMPRESSION SEGMENT **
192 REM   PASS 1 - PACK STRINGS IN TABLE -
194 REM   EACH TIME TABLE CAPACITY REACHED, SORT TABLE AND DELETE THE 100
196 REM   ENTRIES WHICH ARE AT THAT TIME LEAST SIGNIFICANT.
198 REM
200 T1=0
202 K8=0
204 IF END #1 THEN 228
206 INPUT #1, S$
208 K8=K8+LEN(S$)+1
210 GOSUB 600
212 FOR J=Q1 TO Q
214   S$=Q$(J)
216   GOSUB 700
218   IF T1<500 THEN 224
220   GOSUB 800
222   T1=400
224 NEXT J
226 GO TO 204
228 REM
230 REM   END OF PASS 1 - SORT THE TABLE, EXTRACT THE MOST SIGNIFICANT ENTRIES
232 REM   AND WRITE THEM (PRECEDED BY A COUNT THERE-OF) AT THE HEAD OF THE
234 REM   COMPRESSED FILE.
236 GOSUB 800
238 IF T1<128 THEN 242
240 T1=128
242 FOR J=1 TO T1
244   IF FNS(J)<=0 THEN 250
246 NEXT J
248 J=T1+1
250 T1=J-1
252 IF T1>0 THEN 258
254 PRINT "NO COMPRESSION POSSIBLE; PROGRAM TERMINATED."
256 STOP
258 K9=4
260 PRINT #2, USING 261, T1
261 :###
262 FOR J=1 TO T1
264   PRINT #2, T$(J)
266   K9=K9+LEN(T$(J))+1
268 NEXT J
270 REM
272 REM   PASS 2 - BREAK EACH LINE INTO STRINGS, REPLACE STRINGS BY ABBREVIATION
274 REM   FROM TABLE WHERE-EVER POSSIBLE.
276 RESTORE #1
278 W(0)=1
280 IF END #1 THEN 318
282 INPUT #1, S$
284 GOSUB 600
286 FOR J=1 TO Q
288   FOR K=1 TO T1
290     IF Q$(J)<>T$(K) THEN 298
292     W(1)=K-1+128
294     CHANGE W TO Q$(J)
296     GO TO 300
298   NEXT K
300   IF J>1 THEN 306
302   S$=Q$(J)
304   GO TO 308
306   S$=S$&Q$(J)
308 NEXT J
310 GOSUB 900
312 K9=K9+LEN(S$)+1
314 PRINT #2, S$
316 GO TO 280
318 PRINT "COMPRESSION COMPLETED; INPUT ";K8;" CHARS, OUTPUT ";K9;" CHARS."
320 STOP
321 REM
500       REM   ** DECOMPRESSION SEGMENT **
502       INPUT #1, T1
504       FOR J=1 TO T1
506         INPUT #1, Q$(J)
508       NEXT J
510       IF END #1 THEN 542
512       INPUT #1, X$
```

Continued on pg. 14

A Potpourri of Utility Functions for LISP



BY PAUL TARVYDAS
65 Botany Hill Road
Scarborough, Ontario, M1G 3K4
CANADA

From the initial moment of having powered up a LISP interpreter on my microcomputer, I found myself writing and using a number of utility routines to ease the creation of LISP programs. Since the internal representation of programs in LISP is somewhat different than that of "standard" languages, the creation of utility routines such as these may not be obvious to the first-time LISP user so I thought I would share them with you.

First the disclaimers: these routines were written with specific personal, temporal and machine limitations in mind; they were written to achieve a specific purpose in as little time as possible. Considerations such as style, error checking, etc., were chucked out the window as long as the purpose was achieved. Thus, these routines should be viewed as a starting point to get one's system off the ground.

These programs were developed using F.v.d. Wateren's LISP for the 6800. Comments about the code are contained in /* . . . */ separators. Commenting is not supported by this version of LISP and was just superimposed for clarification: the comments should be omitted when actually entering the code. In this version of LISP, the atoms "LAMBDA," "EXPR" and "FEXPR" evaluate to themselves.

In some of the routines, I use a construct which may not be totally obvious at first:

((LAMBDA (. . .) (. . .)) . . .)

This demonstrates a method of avoiding SETQ's using LISP's natural lambda binding of values to variables. For example, if one wants the value of "(CAR X)" in a number of places in the code, instead of evaluating "(CAR X)" each time it is required, one might be prone to SETQing (assigning) the value to some temporary variable (atom) and using this variable in the successive computations. This is unnecessary and adds extra shlock to the code. Instead, we use the lambda expression. The first list following the "LAMBDA" is a list of the new variables and the second list is the actual code to be executed. The new variables are defined with particular values only in the following block of code, very much like an ALGOL BEGIN . . . END block with local variables. The LISP interpreter EVAL expects, when handed a list, that the first element of the list is a function. If, at first glance, the first element isn't a function, EVAL looks at the various values of this first element until it finds some sort of function definition (or until it runs out of things to look at, in which case an error message is given). In our case, the first element is

(LAMBDA (. . .) (. . .))

which isn't a function name, so EVAL eval's this list: the first element is LAMBDA which EVAL knows about—it now knows what the function is: the block of code contained in the LAMBDA expression (i.e., the second list). Now, all that's left to do is to EVALuate the arguments to this function (i.e., the stuff following "(LAMBDA (. . .) (. . .))" in the initial list) and to bind (assign) these values sequentially to the local variables declared in the lambda expression and then to execute the block of code. One will note that this lambda expression is exactly the value "assigned" to an atom when we define it as a function, so what we really have done is to have specified a function without giving it a name—instead of giving EVAL a function name, we give it just the function body.

There's no end to the fun that one can have with EVAL. In the function "EDIT", I have written

(COND . . . ((MEMBER X EDITFNS) (X)) . . .)

In English, this reads "if the value of X is an element of the list EDITFNS, then return the value of "(X)" ". Now, "(X)" is a list, so the first element of this list must be a function—lo and behold, every element of the list EDITFNS is defined as a function with no arguments, so EVAL has the function name (the value which has been assigned to variable X), realizes that there are no arguments to evaluate and then just executes the function (whatever it is). It is very simple to add new functions to this data driven program: just define the function and stick its name into the list EDITFNS.

LISP EDITOR

Since LISP stores programs as lists, editing programs with LISP is fairly simple (LISP is a language geared towards the manipulation of lists). The hard part is trying to decide how to specify to the editor what to do: there are no line numbers or the other usual landmarks in LISP. The best way I could find is to direct a pointer from element to element (of the edited list) until the spot in question is reached, where some editing function can be performed. The editor prints the element at which the pointer is currently pointing. The variable CURR is the internal running pointer. The car of CURR is the current element to be edited (the list containing the element is actually edited, not the element itself). A list, PREV, keeps track of which elements have been visited, so that one may retrace his steps to back up out of a list. The car of PREV is the most recently visited element.

The editor contains a pretty printer - a set of routines which will produce a formatted listing of the function. It has proved to be invaluable for finding missing or incorrectly placed parentheses: the incorrect paragraphing resulting from pretty-printing a function with such errors is quite noticeable.

The pretty-printer only considers three cases when deciding on how to format the output:

1. the first element of the list to be formatted (i.e., a function name) is one of the "MFNS" (functions with multiple arguments)—each argument is printed on a new line, indented;
2. the first element is a list itself – this list is pretty-printed and anything following it goes on a new line;
3. everything else is just printed sequentially.

This pretty-printer is far from perfect—it's not even smart enough to know when it has run over the line boundary. Wateren's LISP outputs a CR-LF when more than 55 characters have been printed on a line. I've found that increasing this number helps the appearance of pretty-printed output (80 hex, say, at address $038A). Note that the formatting of the functions presented here is not exactly that of the pretty-printer.

EDITOR COMMANDS

To start:

(EE name) – edit the EXPR "name"

(EF name) – edit the FEXPR "name"

note: do not QUOTE the function name in the above two commands

(EDIT list) – edit any list: the value of the atom "list"

Once inside the editor:

A – "across" to next element of the edited list

D – "down" into the current element (must be a list) – current item becomes the first element of this list

B – "backup" – go back to the most recently visited item

REP – "replace" current item with next input from the keyboard (this input is not EVALed)

DEL – "delete" current item

INS – "insert" next input after the current item

PUSH – place one level of parentheses around the current item

POP – remove one level of parentheses from around current item

[(left square bracket) – akin to PUSH, but groups any number of items into a list – once the list square bracket has been entered, only "A" commands are accepted until a right square bracket is entered, to close the list (the command aborts if anything but "A" or a right square bracket is entered)

PREF – insert next input in front of first element of the current item (which must be a list) - "preface"

PP – pretty-print the whole edited list

P – pretty-print the current item in context – i.e. the current item plus all of the elements following it in the list which contains the current item

PC – pretty-print the current item only

TOP – go back to beginning of edited list

STOP – exit the editor.

Since LISP uses an input buffer, a number of commands may be entered at once, separated by blanks.

LISP TRACE ROUTINES

These functions allow one to trace the execution of user-defined functions (EXPR's and FEXPR's). If it is desired to trace a system function (SUBR's and FSUBR's) then define a function with a similar name which just calls the system function and trace this.

These routines modify the specified functions so that they call the run time trace routine (#TR2) which prints out the function name, arguments and value.

(TRACE list) – sets up the required information for each function in the list "list"

(UNTRACE list) – undoes the tracing for each function in the list "list"

Once TRACE is executed for some function, each call to this function will result in the output of the name, args, and value of the function, until UNTRACE is executed for this function.

LISP TAPE I/O

In LISP, all output is done via the "PRINT" command and all input via "READ". Most LISPS provide for a method of switching I/O devices—Wateren's LISP uses a device table and the user is required to specify a device number in the PRINT or READ command. Obviously, then, a cassette port, once included in the device table, can just be PRINTed to or READ from, in ASCII, just as if it were the console. This, unfortunately, does not solve the whole problem.

On input, LISP checks the OBLIST for each input atom and must create lists out of the input (LISP doesn't store programs as ASCII text). This, along with the possibility of garbage collection, produces a highly unpredictable rate of input. The best solution is to use some sort of controlled reader device, but I have succeeded, to an extent, in storing programs in 1200 baud cassette format. I imagine that the particulars will vary from machine to machine, but, as an example, here is what I did to interface LISP with my cassette format:

1. write a leader generator routine and assign OPEN and CLOSE device vectors to it;
2. assign a READ device vector to the "read one character from cassette" routine;
3. assign a PRINT device vector to a routine which outputs one character to tape; if the character is a carriage return (CR), output it, then delay about ½ to 1 sec.—when reading the tape back, LISP will wait for the carriage return before working over the input, the delay allows LISP to finish its work before refilling the buffer;
4. "print" only short lines—I adjusted the previously mentioned 55 character limit to about half that—a small LISP routine could be written to print only a few atoms, followed by a TERPRI (CR-LF);
5. always force a garbage collection prior to READing or PRINTing a cassette (the above delays are not sufficient to allow for a garbage collection, in time, on my machine);
6. print one function at a time, preceded and followed by a leader (use MAPCAR to do this any number of times for a list of functions), i.e., in LISP:

(OPEN . . .) (PRINT . . .) (CLOSE . . .) .

Now, with the cassette mechanics out of the way, we still have to PRINT something meaningful: if we just PRINT a function onto tape, it will not get defined when the tape is read back in. The act of defining a function is a function in itself. So what is stored on tape should look exactly like the sequence of ASCII characters one would type at the console to define the function.

The function UNPROP produces a list of PUTPROPS (DEFINE is just a special case of PUTPROP) which, when EVAL'ed will reconstruct the given atom with all of its properties (functions, values, etc.). The function LISTPROPS does this for a list of atoms.

So, to record the functions A, B, C, one could:

```
(MAPCAR  (QUOTE  (A B C) )  (QUOTE DUMP) )
```

where DUMP is defined:

```
(DE DUMP(X)
   (PROG ( ) (OPEN "cassette")
      (PRINT (UNPROP X) "cassette")
      (CLOSE "cassette") ) ) .
```

To read them back and reconstruct them we must READ each of the lists separately. Each of these lists will consist of sublists. Each sublist, when EVALed, results in one property of the atom. One way of doing this might be:

```
(MAPCAR  (QUOTE (1 2 3) ) /*      provide three dummy
                                  args so that next block
                                  of code is executed
                                  three times */
   (QUOTE (LAMBDA  (X)
      (MAPCAR  (READ "cassette") /* get one list */
         (QUOTE (LAMBDA  (Y) /* and evaluate all sublists */
            (EVAL  Y  (ALIST) ) ) ) ) ) ) ) .
```

ALIST is a function in this version of LISP.

CONCLUSION

In the preceding groups of functions, I've defined some which are fairly standard in LISP. Of these, APPEND and MAPCAR are extremely useful. MAPCAR is LISP's do-loop (very vaguely, but it is as useful). The definition of REMPROP results in a copy of the property list of the atom. This should be used with some caution, since, for example, the cold start in Wateren's LISP expects the initial OBLIST to still be where it initially was. This would not be the case if this version of REMPROP were used on one of the initially defined atoms and could result in great rectal discomfort.

I hope that the functions presented here will be of some value.

Happy LISPing.

```
Assorted Required Functions

/* do the next two first, they define the defining functions */

(PUTPROP (QUOTE DE)             /* define EXPR */
         (QUOTE (LAMBDA (A L)
                  (PUTPROP (CAR A)
                           (CONS LAMBDA (CDR A))
                           EXPR)))
         FEXPR)

(PUTPROP (QUOTE DF)             /* define FEXPR */
         (QUOTE (LAMBDA (A L)
                  (PUTPROP (CAR A)
                           (CONS LAMBDA (CDR A))
                           FEXPR)))
         FEXPR)

/* To define an EXPR :  (DE name (local variables) (body of function))
   to define an FEXPR:  (DF name (local variables) (body of function)) */

/* MEMBR - returns T if X is EQ to some element of list Y, replace with MEMBER
           if available */

(DE MEMBR (X Y)
   (COND ((NULL Y) NIL)
         ((EQ X (CAR Y)) T)
         (T (MEMBR X (CDR Y)) )) )

/* REMPROP - see cautionary note in text */

(DE REMPROP (ATM IND)
   (RPLACD ATM (DELETE IND (DELETE (GET ATM IND) (CDR ATM)) )) )

/* DELETE - returns a list which has all occurences of X deleted from list Y
          - uses EQ */

(DE DELETE (X Y)
   (COND ((NULL Y) NIL)
         ((EQ X (CAR Y)) (DELETE X (CDR Y)))
         (T (CONS (CAR Y) (DELETE X (CDR Y)) )) ))


/* APPEND - appends two lists together, result is one list with all elements
            of X and Y */

(DE APPEND (X Y)
   (COND ((NULL X) Y)
         (T (CONS (CAR X) (APPEND (CDR X) Y)) )) )


/* MAPCAR - maps the function onto successive cars of the list
          - ie. applies the function to each element of the list, in sequence
            and returns a list of all of the values */

(DE MAPCAR (X FN)
   (COND ((NULL X) NIL)
         (T (CONS (FN (CAR X)) (MAPCAR (CDR X) FN)) )) )
```

```
LISP EDITOR

/* EDITFNS is the lookup table of valid editor operations */

(SETQ EDITFNS (QUOTE (PC A D B P REP DEL INS ⊏ PUSH POP PP PREF)))

/* edit EXPR: (EE name) */

(DF EE (L A)
   ((LAMBDA (X)
       (COND (X (EDIT X))
             (T (PRINT (QUOTE (NOT AN EXPR)) )) ))
    (GET (CAR L) EXPR) ))

/* (EF name) - edit FEXPR */

(DF EF (L A)
   ((LAMBDA (X)
       (COND (X (EDIT X))
             (T (PRINT (QUOTE (NOT A FEXPR)) )) ))
    (GET (CAR L) FEXPR)))

/* (EDIT list) - main driver */

(DE EDIT (L)
   (PROG (CURR PREV X)
         (TEREAD)
    TOP  (PS L)
         (SETQ PREV (LIST L))
    LOOP (TERPRI)
         (PRINT (QUOTE EDIT:))
         (COND ((EQ (SETQ X (READ)) (QUOTE STOP))
                (RETURN (QUOTE EXIT)))
               ((EQ X (QUOTE TOP))
                (GO TOP))
               ((MEMBR X EDITFNS) (X))
               (T (PRINT (QUOTE (UNDEFINED OPERATION)) )) )
         (GO LOOP) ))

/* PS - "print and set" - advances edit pointer and displays current item */

(DE PS (L)
   (PRINT (CAR (SETQ CURR L)) ))

/* PSB - "print, set and adjust backup pointer"
       - perform given function on the current item, if the result is atomic
         then end of list, "EOL", has been reached, allow no more editting
         until user backs up */

(DE PSB (FN)
   (COND ((ATOM (FN CURR))
          (PRINT (QUOTE EOL)))
         (T (PROG () (SETQ PREV (CONS CURR PREV))
                  (PS (FN CURR)) )) ))

 /* REP - replace current item with item input by user */

(DE REP ()
   (PRINT (CAR (RPLACA CURR (READ) )) ))

/* across */

(DE A ()
   (PSB (QUOTE CDR)))

/* down */

(DE D ()
   (PSB (QUOTE CAR)))

/* backup - current item becomes the one previous to this one, retraces
            A and D commands */

(DE B ()
   (COND ((NULL (CDR PREV))
          (PRINT (QUOTE TOP:)))
         (T (PROG ()
                  (PS (CAR PREV))
                  (SETQ PREV (CDR PREV)) )) ))

/* PUSH - nest current item in one more level of parentheses */

(DE PUSH ()
   (PS (RPLACA CURR (LIST (CAR CURR)) )) )

/* pop - remove one level of parentheses */

(DE POP ()
   (COND ((ATOM (CAR CURR))
          (PRINT (QUOTE (NOT A LIST))))
         (T (PROG ()
                  (RPLACD (LASTCDR (CAR CURR)) (CDR CURR))
                  (PS (CONZ CURR (CAR (CAR CURR)) (CDR (CAR CURR)) )) )) ))

/* lastcdr - returns last non-null cdr of its argument */

(DE LASTCDR (X)
   (COND ((ATOM X) (LIST X))
         ((NULL (CDR X)) X)
         (T (LASTCDR (CDR X)) )) )

/* CONZ - like cons, except, instead of using a new cell, use cell X; used for
          changing the first element of a list */

(DE CONZ (X Y Z)
   (RPLACA (RPLACD X Z) Y ))

/* preface */

(DE PREF ()
   (PS (RPLACA CURR (CONS (READ) (CAR CURR)) )) )

/* INS - insert next input after current item */

(DE INS ()
   (PROG () (RPLACD CURR (CONS (READ) (CDR CURR)))
         (A) ))

/* DEL - delete current item */


(DE DEL ()
   (PROG ()
         ((LAMBDA (X)
             (COND ((EQ CURR X)   /* very first element ? */
                    (CONZ CURR (CAR (CDR CURR)) (CDR (CDR CURR)) ))
                   (T ((COND ((EQ CURR (CDR X))  /* no, so want either RPLACD...*/
                              (QUOTE RPLACD))
                             (T (QUOTE RPLACA))) /* or RPLACA as the function */
                       X  (CDR CURR)) )) )       /* args for RPLACA/RPLACD */
          (CAR PREV))  /* = X*/
         (B)))  /* and back up*/

/* group items into a list - left square bracket */

(DE ⊏ ()
   (PROG (Y)
         (SETQ Y CURR)  /* Y = first item in this group */
     L   ((LAMBDA (X)
             (COND ((EQ X (QUOTE A))
                    (COND ((NULL (CDR Y)) /* move across if possible */
                           (PRINT (QUOTE EOL)))
                          (T (PRINT (CAR (SETQ Y (CDR Y)) )) )) )
                   ((EQ X (QUOTE ⊐)) /* time to close the list */
                    (RETURN (COND ((EQ CURR Y)
                                   (PUSH))       /* only one, so push */
                                  (T (PROG ()
                                           (CONZ CURR
                                                 (CONS (CAR CURR) (CDR CURR))
                                                 (CDR Y))
                                           (RPLACD Y NIL)
                                           (PRINT (CAR CURR)) )) )) )
                   (T (RETURN (PRINT (QUOTE *ABORTED*)) )) ))
          (READ))  /* arg X for above lambda */
         (TERPRI)
         (GO L)))

/* pretty-print the whole mess */

(DE PP ()
   (#PP1 L))

/* print current item in context */

(DE P ()
   (#PP1 CURR))

/* pretty-print current item */

(DE PC()
   (#PP1 (CAR CURR)))

/* set up for pretty-printer */

(DE #PP1 (X)
   (PROG (TFLG PG LC)
         (SETQ PG 14)       /* # of lines on screen-2 */
         (SETQ LC 0)        /* init line counter */
         (SETQ TFLG NIL)    /* flag to avoid excess TERPRI's */
         (#PP X 0 )         /* zero indentation at start */
         (RETURN (TERPRI)) ))

/* pretty-print driver */

(DE #PP (X INDTN)  /* X=list to be printed, INDTN=# of spaces after CR-LF */
   (COND ((ATOM X)
          (PR*NT X))
         ((ATOM (CAR X))  /* atomic function name */
          (COND ((SASSOC (CAR X) MFNS)  /* special case ? */
                 (MARG X INDTN (CDR (SASSOC (CAR X) MFNS)) ))
                (T (PPAF X INDTN)) ))   /* naw, just print it */
         (T (PPLF X INDTN)) ))    /* car of list is a list */

/* MARG - pretty-print a function with multiple arguments
       - print each arg on a new line
       - L is a list of two elements 1) # of spaces to indent after CR-LF
                                     2) "T" if first arg to be printed on a
                                        new line, NIL otherwise  */

(DE MARG (X INDTN L)
   ((LAMBDA (IND2 FL)  /* IND2=new indentation value, FL=first arg flag */
       (PROG ()
             (TERPL INDTN)
             (PR*NT LPAR)
             (PR*NT (CAR X))  /* print function name */
             (COND (FL (TERPL IND2))   /* new line for first arg ? */
                   (T (PR*NT BLANK)))
             (#PP (CAR (CDR X)) IND2)  /* first arg */
             (COND ((CDR (CDR X))  /* rest of args, if any */
                    (MAPCAR (CDR (CDR X))
                            (QUOTE (LAMBDA (Y)
                                           (PROG () (TERPL IND2)
                                                    (#PP Y IND2)) )) ) )
             (PR*NT RPAR)))
    (PLUS INDTN (CAR L)) /* = IND2 */
    (CAR (CDR L))))      /* = FL */
```

```
/* pretty-print atomic function and args, non-special case, all args on same line */

(DE PPAF (X INDTN)
   (PROG ()
         (PR*NT LPAR)
         (PR*NT (CAR X))   /* function name */
         (PARG (CDR X) INDTN)   /* args */
         (PR*NT RPAR)))

 /* PPLF - pretty-print non-atomic function and args */

 (DE PPLF (X INDTN)
    ((LAMBDA (IND1)   /* IND1=INDTN+1 for first "(" */
        (PROG ()
              (PRINT LPAR)
              (SETQ TFLG T)     /* don't want TERPRI */
              (#PP (CAR X) IND1)
              (TERPL INDTN)
              (PARG (CDR X) IND1)
              (PR*NT RPAR) ))
      (PLUS INDTN 1) ))     /* = IND1 */

 /* PARG - print args of a function ,one by one */

 (DE PARG (X INDTN)
    (PROG ()
        L (COND ((NULL X)       /* finished ? */
                 (RETURN NIL))
                ((PRINT BLANK))  /* always NIL */
                ((ATOM (CAR X))  /* this arg is an atom */
                 (PR*NT (CAR X)))
                (T (#PP (CAR X) INDTN)))   /* arg is a list */
          (SETQ X (CDR X))  /* advance to next arg */
          (GO L)))

 /* PR*NT - prints X and resets TFLG (ie. something printed, so want TERPRI)
         - relies on fact that value of PRINT=NIL */

 (DE PR*NT (X)
    (SETQ TFLG (PRINT X)))

 /* TERPL - terpri line and indent N spaces
         - also provides paging of VDM (prints 14 lines, stops and waits for CR) */

 (DE TERPL (N)
    (PROG ()
          (COND (TFLG (RETURN NIL))  /* no terpri if flag=T */
                ((GREATERP LC PG)    /* screen full ? */
                 (PROG ()            /* yup, so... */
                       (TERPRI)
                       (PRINT MMSG)  /* inform user */
                       (TEREAD)(READCH)  /* wait for CR */
                       (SETQ LC 0)   /* reset line counter */
                       (PR*NT RVLF)))/* over-write message */
                (T (PROG ()          /* all clear, do it */
                         (SETQ LC (PLUS LC 1))  /* increment line counter */
                         (TERPRI)) ))
          (SETQ TFLG T)  /* don't TERPRI again until something's been printed
                            avoids blank lines */
      L   (COND ((EQ N 0)  /* now, tab over N spaces */
                 (RETURN NIL)))
          (PRINT BLANK)
          (SETQ N (MINUS N 1))  /* count down */
          (GO L)))

   /* MFNS is the table of special cases - set up like an association list:
      function name, indentation, flag (NIL: print first arg on same line;
      T: start args on new line) - adjust to suit */

   (SETQ MFNS (QUOTE ( (COND 6 NIL)
                       (PUTPROP 9 NIL)
                       (MAPCAR 8 NIL)
                       (PROG 3 NIL)
                       (AND 5 NIL)
                       (OR 4 NIL)
                       (PROGN 7 NIL)
                       (LIST 6 NIL)
                       (LAMBDA 2 NIL) )))

   /* assorted atoms with special print-names */

   (SETQ BLANK (QUOTE "␣"))        /* BLANK prints as a blank */
   (SETQ RPAR (QUOTE ")" ))        /* RPAR prints as a right parenthesis */
   (SETQ LPAR (QUOTE "(" ))        /* left  parenthesis */
   (SETQ MMSG (QUOTE "...MORE␣␣"))/* screen full message, a single atom */
   (SETQ RVLF (QUOTE "??"))        /* two non-printing characters, reverse line feed
                                      and clear line; substitute for the ?'s
                                      (cntl-n, cntl-o ?)  */


      Trace Routines

      /* TRACE - modifies each function in the list (EXPR's and FEXPR's only)
                 so that it will be traced, returns list if successful */

      (DE TRACE (L)
         (COND ((NULL L) NIL)
               ((ATOM L) NIL)  /* has to be a list */
               (T (PROG ()
                        (SETQ PG 10)  /* init screen pager */
                        (SETQ LC 0)
                        (MAPCAR L (QUOTE #TR1))
                        (RETURN L)) )) )

/* UNTRACE - undoes tracing, returns list if successful */

(DE UNTRACE (L)
   (COND ((NULL L) NIL)
         ((ATOM L) NIL)
         (T (PROG ()
                  (MAPCAR L (QUOTE #UT2))
                  (RETURN L)) )) )

/* PRINTL - print elements of list, omit surrounding parentheses
          - goes through screen pager */

(DE PRINTL (X)
   (PROG ()
         (MAPCAR X (QUOTE (LAMBDA (X)
                                  (PROG ()
                                        (PRINT X)
                                        (PRINT BLANK)) )) )
         (SETQ TFLG NIL)
         (TERPL 0)))  /* call screen pager */

/* #UT2 - called by UNTRACE to restore cdr of atom */

(DE #UT2 (NAME)
   (COND ((NULL (GET NAME (QUOTE TRACE))) NIL)
         ((NULL (GET NAME (QUOTE FEXPR))) NIL)
         (T (#UT1 NAME (CDR (CDR (CAR (CDR (CDR (GET NAME (QUOTE FEXPR)) )) )) )))))

(DE #UT1 (NAME L)
   (PROG ()
         (REMPROP NAME (QUOTE TRACE))
         (REMPROP NAME FEXPR)
         (PUTPROP NAME
                  (CAR (CDR (CAR (CDR L)) ))
                  (CAR L)) ))

 /* #TR1 - called by TRACE to modify cdr or atom - replaces EXPR or FEXPR by
           appropriate call to tracing function */

 (DE #TR1 (NAME)
    ((LAMBDA (IND FN)
        (COND ((NULL IND)  /* no EXPR or FEXPR on this atom...*/
               (PRINTL (LIST NAME (QUOTE NOT) (QUOTE TRACEABLE)) ))
              ((GET NAME (QUOTE TRACE)) NIL)  /* already traced */
              (T(PROG ()
                      (REMPROP NAME IND)  /* annihilate EXPR/FEXPR */
                      (PUTPROP NAME
                               T
                               (QUOTE TRACE))  /* mark it as traced */
                      (PUTPROP NAME   /* set up call to #TR2 */
                               (LIST LAMBDA
                                     (QUOTE (A L))
                                     (LIST (QUOTE #TR2)
                                           (LIST (QUOTE QUOTE)
                                                 NAME)
                                           IND
                                           (LIST (QUOTE QUOTE)
                                                 FN)
                                           (QUOTE A)
                                           (QUOTE L)))
                               FEXPR)) )) )
      (COND ((GET NAME EXPR) EXPR)  /* IND = EXPR, FEXPR or NIL */
            ((GET NAME FEXPR) FEXPR))
      (COND ((GET NAME EXPR))  /* FN = the appropriate lambda form */
            ((GET NAME FEXPR)) ) )

/* #TR2 - called when a traced function is called - prints args, executes the
          function and then prints its value - is transparent to calling function */

(DE #TR2 (NAME IND FN A L)
   ((LAMBDA (ARGS)     /* ARGS=list of args to function */
       (PROG (VAL)
             (PRINTL (LIST (QUOTE ARGS)(QUOTE OF) NAME))
             (PRINTL ARGS)
             (SETQ VAL (APPLY FN ARGS L))  /* execute function */
             (PRINTL (LIST (QUOTE VALUE)(QUOTE OF) NAME))
             (PRINTL (LIST VAL))
             (RETURN VAL)))
    (COND ((EQ IND FEXPR)  /* ARGS: args for EXPR and FEXPR are different */
           (LIST A L))
          (T (MAPCAR A (QUOTE (LAMBDA (X)
                                      (EVAL X L)) )) )) ))
 LISP Tape I/O Routines

 /* LISTPROPS - produce a list of putprops which can reconstruct property list of
                all atoms in the list  */

 (DE LISTPROPS (L)
    (COND ((NULL L) NIL)
          (T (APPEND (UNPROP (CAR L)) (LISTPROPS (CDR L)) )) ))

 /* UNPROP - produce a list of putprops for one atom */

 (DE UNPROP (AT)
    (UNPROP1 AT (CDR AT)))

 (DE UNPROP1 (AT L)
    (COND ((NULL L) NIL)
          (T (APPEND (LIST (LIST (QUOTE PUTPROP)
                                 (LIST (QUOTE QUOTE) AT)
                                 (LIST (QUOTE QUOTE) (CAR (CDR L)))
                                 (LIST (QUOTE QUOTE) (CAR L)) ))
                     (UNPROP1 AT (CDR (CDR L)) )) )) )
```

MACHINE LANGUAGE PROGRAM DISPLACEMENT TABLE

BY COLIN KEAY
The University of Newcastle
New South Wales, Australia 2308

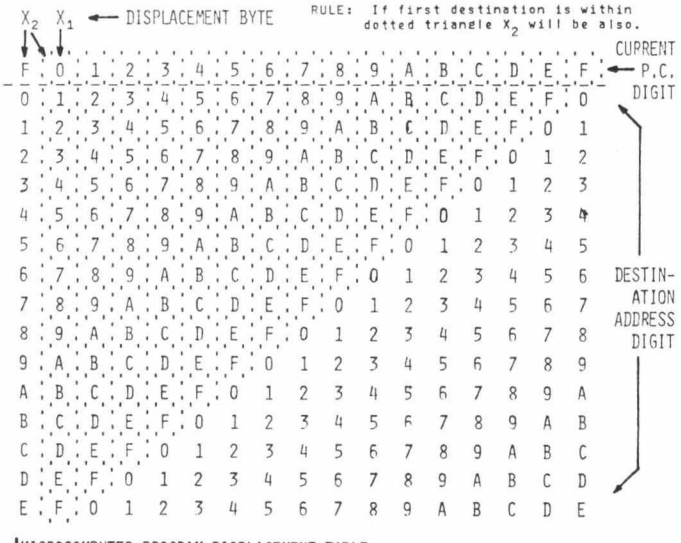
Unless you happen to be proficient at subtraction in hexadecimal arithmetic, the determination of relative address displacements is the most irksome task faced by microcomputer programmers who attempt to assemble programs manually. Even when using a microcomputer with a built-in displacement calculator, the required button-pushing takes time, especially if you forget or mistake the addresses in memory where the origin and destination bytes must be lodged for processing. Furthermore, each microcomputer with this facility has a different procedure to follow.

Alternatively, one may work from sequential hexadecimal number tables which require tedious counting, buy a T.I. Programmer calculator or use a Relative Jump Ruler as described by John S. MacDougall (*Interface Age*, March 1978). This requires that the program be coded on a line-per-byte basis, which is a considerable nuisance for the majority of microcomputers.

There had to be a better way, and it was found by analogy with the unique way the old IBM 1620 computer performed its decimal arithmetic with binary coded numbers. This is the reverse problem: getting decimal-oriented humans to work in hexadecimal. The solution is a simple but universal look-up table for finding hexadecimal differences one digit at a time, with provision for borrowing when necessary between digits. The resulting Microcomputer Program Displacement Table shown in Figure 1 has been used to assist the manual assembly of programs for a variety of different microprocessors and has proved very successful as a programming teaching aid.

Using the Displacement Table is almost self-explanatory. A few examples will make the procedure clear. First, a program branch back from A7 to 54 as demonstrated in Figure 2. Take the low order digit first, descending column 7 to the row containing the destination digit 4 which also contains the digit D in the column under X_1. Similarly, descend from A to 5 and in the same row as 5 the digit A is found in the column under X_2. The required displacement byte is therefore AD.

Note: This material may be reproduced for educational purposes provided due acknowledgement is given. Reproduction for commercial purposes permitted only with the permission of the author.



MICROCOMPUTER PROGRAM DISPLACEMENT TABLE

FIGURE 1

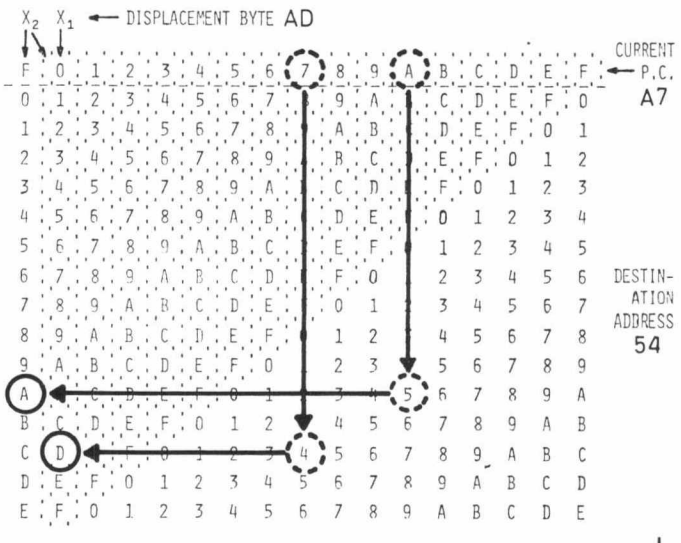


FIGURE 2

Had the first destination digit been located within the dotted triangle the second digit of the displacement byte would have been obtained from the column under X_1. This takes into account the absence of a "borrow" from the subtraction of the first digit. So for our second example consider a branch from A7 to 8D as shown in Figure 3. Descending from 7 we find D is within the dotted triangle. In the row containing D we find the digit 6 under X_1, but for the next digit we must this time find it again under X_1 and not X_2. Descending from A to 8 we find in the row containing 8 the digit E under X_1. The required displacement byte is therefore E6.

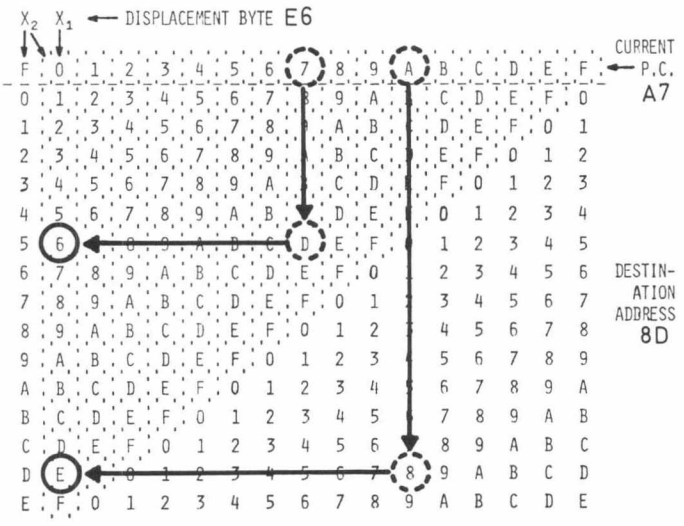


FIGURE 3

We have assumed that the high-order byte is identical for the current address and the destination. Whenever the high order bytes differ, a single byte may be insufficient for the displacement. In such a case the Displacement Table can be employed to obtain X_3 and X_4 in exactly the same way as for X_2. One must simply obey the rule that whenever a destination digit falls within the dotted triangle the subsequent displacement digit must also be within the dotted triangle. For this reason the dotted region of the table has been nicknamed "The Bermuda Triangle"—when landing in it one stays in it, but only for the next turn.

Besides calculating program displacements the table is useful for determining the length of program segments, subroutines, etc. For example, can a program extending from 27B9 to 3C63 be relocated in a gap between 0FA2 and 1455? Using the table we find that the program is 4AA bytes in length and the gap is found to be of 4B3 bytes, so it will fit with 4B3-4AA equal 19 (hex) bytes to spare.

This technique uses hexadecimal numbers exclusively and thereby avoids the confusion inherent in the Address Calculator Tables (such as Ray Boaz's Table in *Byte Magazine*, April 1978) and Multi-base Conversion Charts (wall-size or otherwise) where the actual arithmetic is performed in decimal notation.

Finally, when time permits, it is a good idea to check the results obtained using the Displacement Table by attempting to perform an actual hexadecimal subtraction and if necessary, counting digits in hexadecimal to verify the result. After a few weeks (or months?) of practice the terrors of hexadecimal arithmetic will vanish and the Displacement Table can then be thrown away—even when calculating fearsome reverse jump displacements.

About his work as an associate professor of physics, Colin S. L. Keay has this to say:

I supervise a course on Electronics and Instrumentation in which we endeavour to give some of our Science majors, particularly in disciplines other than Physics, some degree of preparation for the day when they find themselves in a highly instrumented laboratory environment. In achieving this we feel that a sound introduction to microcomputer programming is essential and we have small development systems based on each of the more common microprocessors, including that 6800, Z80, 6502, 2650, SC/MP and COSMAC. It is in the environment that the enclosed displacement table has been well and truly tried and tested by the students.

• MITS •
Sequential File Dump

BY A. L. BENDER, M.D.
© 1979, Neurological Services, Inc.,
336 Center Avenue
Westwood, NJ 07675

There is a proverb which is always true when it comes to data files: "A picture is worth a thousand words." Whenever a program is prepared to read or write a data file it becomes imperative to look at the file to see if it "looks like it should" or to see what's wrong with it.

This program allows you to read a sequential data file and display the file on an 80 character wide line. It was designed to run under MITS DOS. If you own MITS Disk BASIC you might want to write a program of this type in BASIC, which is really a trivial task. You are then conned into booting up BASIC every time you want to look at a file for a second. This disrupts the "interactive programming" which one is supposed to enjoy with MITS DOS. I wrote this dump for myself only at first, but decided to make it useful for other MITS users by removing "custom features."

Systems Requirements: 8080 CPU or Z-80 CPU. 16KB storage. MITS DOS with at least one diskfile.

Program Operation: Loads into the transient program area from 24000_8 up. The program may be called in two ways: Either as a processor program by specification of the file and drive on the call to dump program or by use of a "RUN" statement and providing the parameters to the dump program explicitly. One file is dumped from start to finish by each call. Further modifications to the program to provide for selective dumping of records, different formats or any other special customization can be worked into the main program with ease.

When called as a processor program you need to give the file name and the drive on which it resides. For example to dump the file "*RELFILE" on drive 3: .FDMP *RELFILE,3.

When called as a user program the program will solicit a response for file name and drive.

Dump Format: For each of the 128 bytes in a sector, the dump program will print 8 characters followed by two spaces. Eight bytes are printed on each line. The format of the eight characters is as follows:

000bHHbA[bb]

Where '0' represents an octal digit, H represents a hex digit and A represents a printable ASCII character. If the character is not able to be printed, a space is printed instead. The lower case B ["b"] represents the character "blank." No special screen or forms control is used so that the program can be used on any device.

Getting FDMP Up and Running: Type in the source code, make your modifications and link edit the relocatables. It should run on any system without problems. I can supply a copy of this program on YOUR 8 inch, hard sectored disk, packed in a reusable mailing carton with a check or money order for $8 to cover my expenses in handling and shipping. I cannot be responsible for read/write errors on the disk as I am only providing this as a service to you to avoid the labor of typing in each line. If you need another form of input I cannot be of any help. The distribution disk contains the source, relocatables and absolute for the dump. Other programs on the disk will not be disturbed.

Program Description: The program uses several entry points in DOS which are not documented elsewhere. They are included here as an appendix. Between line numbers 100 and 899 the program is initialized and the file name and drive are obtained. At first the program name is set in DOS and the output conversion program is set to work in decimal. PSUB1 gets the file name parameters and this name is moved to FNAME by use of the resident routine at X'006E' which in this system is called MOVE8. PSUB2 is then called to get the second parameter, the disk number. This is saved and the file is opened by OPEN$ which opens the file with suppression of the monitor error message. This allows OPEN$ to get control if an error occurs. The error code is printed out and the program aborts if an error return is taken from IO.

At START the sector or block counter is zeroed and the stack pointer is reset. LOOP is the loop which is executed for each block or sector read from the disk file. RET identifies each block as to sector number and NEXT outputs each byte in the dump format by means of "OUTPUT." When all of the bytes are dumped, the internal byte counter in the IO supervisor is set to zero and the next block is read. The subroutines are short and probably don't require a detailed description.

Notice: This program may be used without restriction by any "non-commercial" user. It may not be sold or modified and sold without explicit written permission from: Neurological Services, Inc., 336 Center Ave., Westwood, NJ 07675.' 07675.

Appendix: This is a narrative description of the non-published entry points in MITS DOS. Not all of these entry points are described, only those used in the file dump program.

Address	Description and SYSENT label name [Neurological Services System]
X'005D'	LDEM - Loads the DE register pair from the memory locations pointed to by the HL register pair. HL is incremented by two on exit. (m,m+1) are loaded into DE.
X'006E'	MOVE8 - Moves eight bytes from DE to HL. BC=0 on exit. DE and HL are updated.
X'0071'	MOVE - Moves the number of bytes contained in BC from DE to HL on exit DE and HL are updated and BC=0.
X'10A6'	[NOT IN OUR SYSENT] - An ASCII String "ENTER FILE NAME" used by MSG, ASK$$ and other routines. CR/LF preceed the message and it ends with the high order bit set.
X'101D'	[NOT IN OUR SYSENT] - An ASCII String "ENTER DRIVE" similar to the above string in construction.
X'02EE'	CCEL1 - Conversion cell for binary to ASCII conversion. Used to set base of conversion to either 8 or 10 depending on how you want the output (octal or decimal).
X'02EC'	CCELL - See above. You set both of them before use.
X'02B7'	BINCV$ - Converts the contents of HL to an ASCII digit string in memory. If CCELL and CCEL1 are set to 10, conversion is decimal and if they are set to 8, conversion is octal. The call to this subroutine should be followed by a DW containing the address of where the first of the six digits of ASCII is to be stored. An example: CALL BINCV$; DO CONVERSION DW ASCOUT ; WHERE THE ASCII GOES
X'0F75'	ASK$$ - This subroutine is used to get the first or next parameter from a processor call. It must be followed by a DW which describes a string to be output in the event that the parameter is missing. On return, DE is pointing to the first byte of the parameter. If ASK$$ finds a C/R code (x'0D') it will output the message described at the DW and wait for a response to be typed in.
X'0FBE'	AANS$ - Converts ASCII characters at DE to binary. Result in A register on exit. This must be followed by an error return address in case an error in conversion occurs, the program will jump to the error return address. Conversion stops on a comma, blank or C/R.

LISTING

```
                        ENTRY   FILEDM
                        PROGRAMMED BY A. RENDER, M. D.
                        NEUROLOGICAL SERVICES INC. - WESTWOOD NJ 07675
                        C O P Y R I G H T  1 9 7 9  NEUROLOGICAL SERVICES INC.
                        DUMPS A BINARY OR ASCII FILE IN OCTAL AND HEX ALSO IN ASCII
                        FILE MUST BE IN SYSTEM FORMAT.
                        ;
                        CALL IS - .FDMF FILE,UNIT
                        ;
                        IF FILE IS NOT ASSIGNABLE, GIVES THE REASON
                        ;
                        ;
                        ** THIS IS A DISTRIBUTION VERSION [1.0] **
                        NON-COMMERCIAL USE MAY BE WITHOUT RESTRICTION.  COMMERCIAL USERS
                        SHOULD APPLY FOR USE TO - NEUROLOGICAL SERVICES INC.
                                                  400 OLD HOOK ROAD
                                                  WESTWOOD NJ 07675 (201-6644982)
                        ;
000200                  DS      150
000210  STACK   EQU     *
000220                  EXT     IO,MSG,ABORT,EXIT,TASKNM
000223                  EXT     MOVE8,OUTEEE
000225                  EXT     HEXOUT,OCTOCT,ASCOUT
000230                  EXT     OPEN$   ; SUBROUTINE TO OPEN FILE
000240                  EXT     CLOF$   ; SUBROUTINE TO CLOSE FILE
000242  ASK$$   EQU     X'0F75' ; RESIDENT "ASK" ROUTINE
000243  CCEL1   EQU     X'02EE' ; CONVERSION CELL
000244  BINCV$  EQU     X'02B7' ; BINARY TO DECIMAL
000245  CCELL   EQU     X'02EC' ; CNVCELL
000246  AANS$   EQU     X'0FBE' ; RESIDENT AANS
000247  NAME    DB      13,10
000249          DC      "DISK FILE DUMP VER 1.0"
                        ;
000250  FILEDM  LXI     H,NAME
000260          SHLD    TASKNM
000270          LXI     SP,STACK ; SET STACK POINTER
000280          CALL    MSG
000290          DW      NAME    ; PUT OUT NAME
000300          MVI     A,10    ; SET TO DO DECIMAL CONVERSION
000310          STA     CCELL
000320          STA     CCEL1   ; ASK
000330          CALL    PSUB1   ; FILE NAME
000340          LXI     H,FNAME ; MOVE IT TO OPEN AREA
000350          CALL    MOVE8   ; GET DEVICE NUMBER
000360          CALL    PSUB2   ; OPEN FILE FOR INPUT
000370          STA     IN      ; INPUT FILE OPEN FCB
000380          CALL    OPEN$
000390          DW      INFILE
000400          JMP     START
000410  PSUB1   CALL    ASK$$   ; GET NAME FROM RESIDENT OR DIRECTLY
000420          DW      X'10A6' ; SAYS "ENTER FILE NAME"
000430          RET
000440  .A      CALL    X'0EB7'
000450  PSUB2   CALL    ASK$$   ; GET DEVICE NUMBER
000460          DW      X'101D' ; ENTER DRIVE
000470          CALL    AANS$   ; CONVERT
000480          DW      -A      ; ERROR IN CONVERSION
000490          RET
000900  START   LXI     H,0
000910          SHLD    CTR
000920          LXI     SP,STACK        ; RESET STACK POINTER
001200  LOOP    CALL    IO      ; BLOCK INPUT 128 BYTES
001210          DW      INBLK
001220          LDA     INBLK+1 ; GET STATUS RESPONSE
001221          ORA     A
001226          RET     ; JUMP IF STATUS=0
001230          CPI     5       ; LESS THAN 1 FULL BLOCK READ
001240          JZ      BFRLER  ; BUFFER LENGTH ERROR
001250          CPI     025     ; END FILE ERROR
001260          JZ      EOF
001270          CPI     012     ; IOERR ; I/O ERROR
001280          JZ      IOERR
001281          CPI     011     ; FILE LINK ERROR
001286          JZ      FLIERR
001290  RET     CALL    SECTID  ; PUT OUT SECTOR ID
001300          LHLD    INBLK+3 ; GET BYTE COUNTER
001302          MOV     B,M
001307          XCHG
001309          LHLD    INBLK+5 ; GET BUFFER ADDRESS
001310  NEXT    MOV     A,M     ; MOVE BUFFER WORD TO CONVERT
```

```
000444 315 001065'      001320          CALL    OUTPUT  ; OUTPUT IN PROPER FORMAT
000447 043              001330          INX     H       ;       INCREMENT MEMOORY POINTER
000450 005              001340          DCR     B
000451 302 000443'      001370          JNZ     NEXT
000454 257              001375          XRA     A
000455 022              001378          STAX    D       ; CLEAR MEMORY COUNTER IN IO SUPERVISOR
000456 303 000370'      001380          JMP     LOOP    ; NEXT RECORD
000461 315 000000*      001410 EOF      CALL    MSG
000464 000525'          001420          DW      EOFMSG
000466 076 001          001425          MVI     A,1     ; CLOSE FILE 1
000470 315 000000*      001430          CALL    CLOF$
000473 303 000000*      001450          JMP     EXIT
000476 107 000 001 000  001470 INBLK    DB      0107,0,1,0,0,0,0        ; INPUT A BLOCK
000502 000 000 000
000505 304 000 001 201  001490 INFILE   DB      0304,0,1,1+0200,6
000511 006
000512 000              001500 DN       DB      0
000513 000515'          001510          DW      FNAME
000515                  001520 FNAME    DS      8
000525 015 012 105 117  001530 EOFMSG   DB      13,10,'EOF ON INPUT FILE',0200
000531 106 040 117 116
000535 040 111 116 120
000541 125 124 040 106
000545 111 114 105 200
000551 015 012 104 125  001540 IOMSG    DB      13,10,'DUMP INPUT FILE ERROR',0200
000555 115 120 040 111
000561 116 120 125 124
000565 040 106 111 114
000571 105 040 105 122
000575 122 117 122 200
000601 015 012 102 106  001550 BFRMSG   DB      13,10,'BFR LEN ERR DUMP INPUT',0200
000605 122 040 114 105
000611 116 040 105 122
000615 122 040 104 125
000621 115 120 040 111
000625 116 120 125 124
000631 200
000632 015 012 104 125  001560 CONT     DB      13,10,'DUMP CONTINUED DESPITE ABOVE ERROR',0200
000636 115 120 040 103
000642 117 116 124 111
000646 116 125 105 104
000652 040 104 105 123
000656 120 111 124 105
000662 040 101 102 117
000666 126 105 040 105
000672 122 122 117 122
000676 200
000677 015 012 106 111  001565 FLMSG    DB      13,10,'FILE LINK ERROR ON DUMP INPUT FILE',0200
000703 114 105 040 114
000707 111 116 113 040
000713 105 122 122 117
000717 122 040 117 116
000723 040 104 125 115
000727 120 040 111 116
000733 120 125 124 040
000737 106 111 114 105
000743 200
000744 015 012 012 123  001570 SECTMS   DB      015,012,012,'SECTOR # '
000750 105 103 124 117
000754 122 040 043 040
000760 000 000 000 000  001580 SECNO    DB      0,0,0,0,0,0     ; SECTOR NUMBER
000764 000 000
000766 012 015 212      001590          DB      012,015,0212
000771 000000           001595 CTR      DW      0       ; SECTOR COUNTER
000773 345              001600 SECTID   PUSH    H
000774 325              001610          PUSH    D
000000                  000110          EXT     OCTOCT,OUTEEE,IO,ABORT,MSG,LDEM
177777                  000112 $$       EQU     -1
000000                  000120 ; OPEN - GENERALIZED ROUTINE TO OPEN A FILE - CALL IS:
000000                  000130 ;        CALL    OPEN$
000000                  000140 ;        DW      FILE-CONTROL-BLOCK-OPEN ADDRESS
000000                  000150 ;
000000                  000160 ; THIS ROUTINE WILL OPEN THE FILE, CHECK FOR ERRORS AND RETURN
000000                  000170 ; CONTROL IN-LINE IF OK.  IF NOT OK IT WILL REPORT THE REASON
000000                  000180 ; AND RETURN CONTROL TO ABORT.
000000                  000190 ;
000000                  000200 ; THE FCB WILL HAVE THE 0104 OP CODE FOR OPEN CHANGED TO 0304
000000                  000210 ; TO ALLOW PROGRAMMED RECOVERY
000000                  000220 ;
000000                  000230 ; PROGRAMMED BY A. L. BENDER, M. D.
000000                  000240 ; MAY 18, 1979
000000                  000250 ;
000000 343              000260 OPEN$    XTHL            ; GET RETURN ADDRESS
000001 315 000064*      000270          CALL    LDEM    ; GET ADDRESS OF RETURN
000004 343              000280          XTHL    ; PUT IT BACK
000005 353              000290          XCHG    ; HL=>FCB ADDRESS
000006 042 000021'      000300          SHLD    OPENER
000011 076 200          000310          MVI     A,0200
000013 266              000320          ORA     M
000014 167              000330          MOV     M,A
000015 043              000340          INX     H
000016 315 000000*      000350          CALL    IO
000021 177777           000360 OPENER   DW      $$      ; ADDRESS OF FCB
000023 257              000370          XRA     A
000024 266              000380          ORA     M       ; EXAMINE STATUS RESPONSE
000025 310              000390          RZ              ; RETURN TO USER CODE
000026 365              000400          PUSH    PSW     ; SAVE STATUS
000027 315 000000*      000410          CALL    MSG
000032 000057'          000420          DW      FMSG1
000034 043              000430          INX     H
000035 176              000440          MOV     A,M
000036 366 060          000450          ORI     "0"
000040 315 000000*      000460          CALL    OUTEEE
000043 315 000000*      000470          CALL    MSG
000046 000067'          000480          DW      FMSG2
000050 361              000490          POP     PSW
000051 315 000000*      000500          CALL    OCTOCT
000054 303 000000*      000510          JMP     ABORT
000057 015 012 106 111  000520 FMSG1    DB      13,10,'FILE:',0240
000063 114 105 072 240
000067 040 103 117 125  000530 FMSG2    DC      ' COULDN'T BE OPENED - REASON: '
000073 114 104 116 047
000077 124 040 102 105
000103 040 117 120 105
000107 116 105 104 040
000113 055 040 122 105
000117 101 123 117 116
000123 072 240
000125 062 000140'      000540 CLOF$    STA     CLOSER
000130 315 000000*      000550          CALL    IO
000133 000136'          000560          DW      CLOSE
000136 311              000570          RET
000137 105 000          000580 CLOSE    DB      0105,0
000140 000              000590 CLOSER   DB      0
000141                  000600          END
000775 305              001620          PUSH    B
000776 052 000771'      001630          LHLD    CTR     ; GET SECTOR NUMBER
001001 043              001635          INX     H       ; INCREMENT SECTOR COUNT
001002 042 000771'      001638          SHLD    CTR
001005 315 001267       001640          CALL    BINCV$  ; PUT OUT ASCII SECTOR COUNT
001010 000760'          001650          DW      SECNO
001012 315 000000*      001660          CALL    MSG
001015 000744'          001670          DW      SECTMS
001017 301              001680          POP     B
001020 321              001690          POP     D
001021 341              001700          POP     H
001022 311              001710          RET
001023 315 000000*      001720 IOERR    CALL    MSG
001026 000551'          001730          DW      IOMSG
001030 315 000000*      001740          CALL    MSG
001033 000632'          001750          DW      CONT
001035 303 000370'      001760          JMP     LOOP
001040 315 000000*      001770 BFRLER   CALL    MSG
001043 000601'          001780          DW      BFRMSG
001045 315 000000*      001790          CALL    MSG
001050 000632'          001800          DW      CONT
001052 303 000430'      001810          JMP     RET
001055 315 000000*      001820 FLIERR   CALL    MSG
001060 000677'          001830          DW      FLMSG
001062 303 000000*      001840          JMP     EXIT
001065                  001850 ; OUTPUT - OUTPUTS 1 BYTE AT A TIME.  PUTS OUT OCT, HEX AND ASCII
001065 345              001860 OUTPUT   PUSH    H
001066 325              001870          PUSH    D
001067 305              001880          PUSH    B
001070 365              001890          PUSH    PSW
001071 076 007          001900          MVI     A,7
001073 240              001910          ANA     B
001074 314 001141'      001920          CZ      CRLF    ; EVERY EIGHTH TIME PUT OUT CRLF
001077 361              001930          POP     PSW     ; RECOVER BYTE TO PUT OUT
001100 315 000000*      001940          CALL    OCTOCT  ; PUT OUT ooo
001103 315 001131'      001950          CALL SPACE      ; PUT OUT b
001106 315 000000*      001960          CALL    HEXOUT  ; PUT OUT xx
001111 315 001131'      001970          CALL SPACE      ; PUT OUT b
001114 315 000000*      001980          CALL    ASCOUT  ; PUT OUT a
001117 315 001131'      001990          CALL    SPACE
001122 315 001131'      001992          CALL    SPACE
001125 301              002000          POP     B
001126 321              002010          POP     D
001127 341              002020          POP     H
001130 311              002030          RET
001131 365              002040 SPACE    PUSH    PSW
001132 076 040          002050          MVI     A,' '
001134 315 000000*      002060          CALL    OUTEEE
001137 361              002070          POP     PSW
001140 311              002080          RET
001141 076 015          002090 CRLF     MVI     A,13
```

```
001143 315 000000*      002100          CALL    OUTEEE
001146 076 012          002110          MVI     A,10
001150 315 000000*      002120          CALL    OUTEEE
001153 311              002130          RET
001154                  002140 ; FINAL OUTPUT LINE WILL APPEAR ooobxxbabb
001154                  027000          END     FILEDM
```

```
000000                  000100          ENTRY   OPEN$
000000                  000105          ENTRY   CLOF$
000000                  000100          ENTRY   HEXOUT
000000                  000110          EXT     OUTEEE
000000 365              000120 HEXOUT   PUSH    PSW
000001 315 000021'      000130          CALL    HEXL
000004 315 000000*      000140          CALL    OUTEEE
000007 361              000150          POP     PSW
000010 365              000155          PUSH    PSW
000011 315 000025'      000160          CALL    HEXR
000014 315 000000*      000170          CALL    OUTEEE
000017 361              000180          POP     PSW
000020 311              000190          RET
000021 037              000200 HEXL     RAR
000022 037              000210          RAR
000023 037              000220          RAR
000024 037              000230          RAR
000025 346 017          000240 HEXR     ANI     017
000027 376 012          000250          CPI     10
000031 332 000036'      000260          JC      +A
000034 306 007          000270          ADI     7
000036 306 060          000280 .A       ADI     "0"
000040 311              000290          RET
000041                  000300          END
```

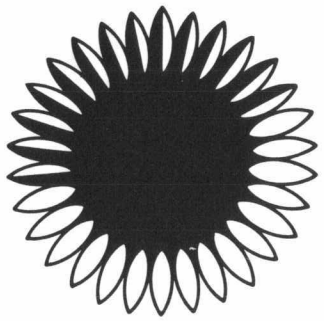
```
000000                  000100          ENTRY   ASCOUT
000000                  000110          EXT     OUTEEE
000000 365              000120 ASCOUT   PUSH    PSW
000001 305              000130          PUSH    B
000002 376 040          000140          CPI     " "
000004 332 000015'      000150          JC      BLOUT+1
000007 376 176          000160          CPI     0176
000011 322 000015'      000170          JNC     BLOUT+1
000014 001 020076       000180 BLOUT    LXI     B,X'203E'       ; SOMETIMES MVI A," "
000017 315 000000*      000190          CALL    OUTEEE
000022 301              000200          POP     B
000023 361              000210          POP     PSW
000024 311              000220          RET
000025                  000230          END
```

```
000000                  000100          ENTRY   OUTEEE
000000                  000110          EXT     IO
000000 062 000016'      000120 OUTEEE   STA     OUT+3
000003 365              000130          PUSH    PSW
000004 315 000000*      000140          CALL    IO
000007 000013'          000150          DW      OUT
000011 361              000160          POP     PSW
000012 311              000170          RET
000013 003 000 101 000  000180 OUT      DB      3,0,0101,0
000017                  000190          END
```

```
000000                  000100          ENTRY   IO,ABORT,EXIT,TASKNM,MSG
000000                  000110          ENTRY   LDEM,SDEM,AANS$,ASK$$,MOVE,MOVE8,BINCV$
000000                  000115          ENTRY   CCELL,CCEL1
000135                  000120 LDEM     EQU     X'005D'
000142                  000130 SDEM     EQU     X'062'
000156                  000140 MOVE8    EQU     X'06E'
000161                  000150 MOVE     EQU     X'071'
```

```
007565                  000160 ASK$$    EQU     X'0F75'
007676                  000170 AANS$    EQU     X'0FBE'
001267                  000180 BINCV$   EQU     X'02B7'
001354                  000190 CCELL    EQU     X'02EC'
001356                  000195 CCEL1    EQU     X'02EE'
000424                  000200 IO       EQU     0424
002674                  000400 ABORT    EQU     02674
002733                  000410 EXIT     EQU     02733
007654                  000600 MSG      EQU     07654
011015                  000610 TASKNM   EQU     011015
000000                  000700          END
```

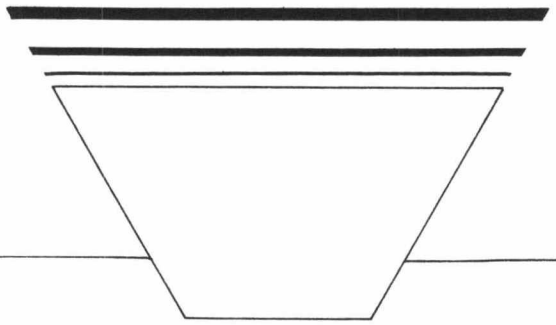
```
000000                  000100          ENTRY   OCTOCT
000000                  000110          EXT     OUTEEE
000000 365              000120 OCTOCT   PUSH    PSW
000001 305              000130          PUSH    B
000002 107              000135          MOV     B,A     ; TEMP
000003 007              000140          RLC
000004 007              000150          RLC
000005 346 003          000160          ANI     3       ; MASK HO BITS
000007 315 000030'      000170          CALL    OCTYPE+2
000012 017              000180          RRC
000013 017              000190          RRC
000014 017              000200          RRC
000015 315 000026'      000210          CALL    OCTYPE
000020 315 000026'      000220          CALL    OCTYPE
000023 301              000230          POP     B
000024 361              000240          POP     PSW
000025 311              000250          RET
000026 346 007          000260 OCTYPE   ANI     7       ; MASH 1 BYTE
000030 366 060          000270          ORI     "0"     ; OR IN NUMERIC BITS
000032 315 000000*      000280          CALL    OUTEEE  ; TYPE CHARACTER
000035 170              000290          MOV     A,B     ; RESET CHARACTER
000036 311              000300          RET
000037                  000310          END
```



A PROPOSED STANDARD

FOR A 16 BIT SYSTEM

DATA EXCHANGE



BY CHARLES B. FALCONER
Chief Instrumentation Engineer
Yale University School of Medicine
New Haven, Connecticut 06504

© 1979, Charles B. Falconer

A point not made in the article is that the "VALID" signal is replaced by "INACK" for interrupt acknowledgement, and that the interruptor then supplies identification on the DB buss. Action is identical to a normal buss read. – SMR

INTRODUCTION

A standard, such as this, is only of use if freely available to all, and if easily physically realized. The system must provide for most, but not all, desireable system inter-action.

The proposed system is based on an existing buss, in use at the Yale School of Medicine for the last three years. The extension to a 16 bit system reflects the experience gained on the 8 bit system.

Signals have been deliberately grouped to:

1. Provide mnemnonic relations to pin nos
2. Avoid transient signals coupling to address/data
3. Isolate power pins (until spares are used)

Several spare pins have been provided, whose use is up to the systems designer. However, if modules are built using only the signals defined below, and following timing restrictions not yet included in this definition, all such modules should be compatible.

Notice that the address space provided is somewhat extreme. This is intended to allow the use of virtual memory systems in the future, with hardware implemented segment absence trapping. It may well be useful in the future simply because memory density is increasing and prices are dropping. In addition, the address space can be used in paging and relocation algorithms.

The system has also been designed so that 8 bit microcomputers, such as the 8080, 6800, 650X, 1802, Z80, 2650, etc., can be interfaced and utilize 16 bit wide memory modules. Thus peripheral design need not be repeated when using various devices, possibly because of software availability.

The system does not permit the use of local power supply regulators, except for analog sub-system power derived from the unregulated 20V lines. This prevents heat dissipation in the most sensitive areas, i.e. the modules themselves. The presence of higher voltage lines permits power-on sequencing to be built into modules, where necessary.

BACKPLANE SIGNAL LIST FOR 16 (AND 8) BIT SYSTEMS

Signal names terminated with "/" are active low, others active high. Backplane lines are to be terminated with 10K ohms to +5 V, and 100PF in series with 100 ohms to ground. This provides default signal conditions and absorbs rings on signal transitions. The termination is to be repeated at each end of the backplane line.

	PINS	SIG NAME	COMMENTS
	1	GND	
	2	GND	COMMON POWER RETURN
	3		SPARE
	4	-12V	POWER SUPPLY
	5		SPARE
	6	+12V	POWER SUPPLY
	7		PIN LOST FOR CARD GUIDE
	8	-5V	POWER SUPPLY
	9		SPARE
	10	DB00/	BIT 0, BIDIRECTIONAL DATA BUSS
	11	DB01/	BIT 1 "
	12	DB02/	BIT 2 "
	13	DB03/	BIT 3 "
	14	DB04/	BIT 4 "
	15	DB05/	BIT 5 "
	16	DB06/	BIT 6 "
	17	DB07/	BIT 7 "
	18	DB08/	BIT 8, "
	19	DB09/	BIT 9 "
	20	DB10/	BIT 10 "
	21	DB11/	BIT 11 "
	22	DB12/	BIT 12 "
	23	DB13/	BIT 13 "
00088000	24	DB14/	BIT 14 "
00089000	25	DB15/	BIT 15 "
00090000	26	WRTHI/	CODED WITH WRTLO/ FOR OPERATIONS. SEE BELOW
00091000	27	WRTLO/	CODED WITH WRTHI/ FOR OPERATIONS. SEE BELOW
00092000	28	IO/	I/O VERSUS MEMORY OPERATION DESCRIPTOR
00093000	29	VALID/	PINS 10 THRU 53 ARE VALID
00094000	30	MAR00/	BIT 0, MEMORY/IO ADDRESS (BYTE ADDRESS)
00095000	31	MAR01/	BIT 1 "
00096000	32	MAR02/	BIT 2 "
00097000	33	MAR03/	BIT 3 "
00098000	34	MAR04/	BIT 4 "
00099000	35	MAR05/	BIT 5 "
00100000	36	MAR06/	BIT 6 "
00101000	37	MAR07/	BIT 7 "
00102000	38	MAR08/	BIT 8 "
00103000	39	MAR09/	BIT 9 "
00104000	40	MAR10/	BIT 10 "
00105000	41	MAR11/	BIT 11 "
00106000	42	MAR12/	BIT 12 "
00107000	43	MAR13/	BIT 13 "
00108000	44	MAR14/	BIT 14 "
00109000	45	MAR15/	BIT 15 "
00110000	46	MAR16/	BIT 16 "
00111000	47	MAR17/	BIT 17 "
00112000	48	MAR18/	BIT 18 "
00113000	49	MAR19/	BIT 19 "
00114000	50	MAR20/	BIT 20 "
00115000	51	MAR21/	BIT 21 "
00116000	52	MAR22/	BIT 22 "
00117000	53	MAR23/	BIT 23 "
00118000	54	PERR/	MEMORY ERROR SIGNAL
00119000	55	STRB/	READ, WRITE, AND SYNC STROBE. SEE BELOW
00120000	56	READY/	PERIPHERAL IS READY. SEE BELOW
00121000	57	DMARQ/	REQUEST FOR DMA ACCESS. I E BUSS BUSY
00122000	58	DMACK/	GRANT OF DMA REQUEST
00123000	59	DMAEN/	ENABLE BUSS ACCESS
00124000	60	NMI/	NON-MASKABLE (PANEL) INTERRUPT
00125000	61	INTEO/	INTERRUPT ENABLE OUTPUT TO CARDS IN CHAIN
00126000	62	INTEI/	INTERRUPT ENABLE INPUT FROM CHAIN
00127000	63	INTRQ/	INTERRUPT REQUEST
00128000	64	INACK/	INTERRUPT ACKNOWLEDGE FROM CPU
00129000	65		SPARE
00130000	66		SPARE
00131000	67		SPARE
	68		SPARE
	69		SPARE
	70	RUN	DEASSERTION INHIBITS READY, FOR SINGLE-STEP AND
	71	RESET/	MASTER SYSTEM RESET DE
	72		SPARE
	73	-20V	NOMINAL, FOR LOCALLY REGULATED ANALOG SUPPLIES
	74		SPARE, RECOMMENDED TO BE GND
	75	+20V	NOMINAL, FOR LOCALLY REGULATED ANALOG SUPPLIES
	76		SPARE, RECOMMENDED TO BE GND
	77	+5V	POWER SUPPLY
	78	+5V	

Note 1: WRTHI and WRTLO control byte versus word writes. If neither is asserted, a 16 bit read is performed from the peripheral, and the data placed on DB. If both are asserted, a 16 bit write from DB is performed. If WRTHI alone is asserted, the higher order of 8 bits of DB are written, and the low order bits are ignored, and vice-versa when WRTLO alone is asserted.

For byte reads, if the low order address bit (MAR00) is asserted, a peripheral will assume that the read is only being performed from the high-order byte. If MAR00 is deasserted, the peripheral cannot distinguish between a byte read and a word read. An 8 bit peripheral should assume all reads to be byte reads.

Note 2: IO/ can be considered as a 25th address bit. It normally separates I/O space from memory space.

Note 3: The following signals are intended to be driven by open collector (i.e."wire or D" on the buss) drives: RESET/, DMARQ/, RUN, INTRO/. This permits simutaneous energization by multiple sources.
In addition, during DMA operations (DMACK is true), the following may also be driven externally: WRTHI/, WRTLO/, STRB/, MAR00/, THRU MAR23/, AND IO/. and, if both WRTHI/ and WRTHLO/ are not high, DB00 thru DB15 may be driven.

Note 4: Buss access is gained by asserting DMARQ (only if DMARQ was false) and deasserting DMAEN. When DMACK signal is received, buss access is granted. Note that a device will deassert DMAEN at its output if DMACK is received false, or if it itself is asserting DMARQ. DMAEN is daisy chained to the next modules DMACK pin. This system can resolve buss contention in multiprocessor and/or DMA systems, but is only practical if buss use time percentage is small for each buss controller.

Note 5: (after buss access is gained) Peripheral communication commences with READY/ and VALID/ deasserted. The controlling device (e.g., CPU) sets the conditions on the mar buss (including IO/), WRTHI/ and WRTHLO/. If WRTHI or WRTLO are asserted the controller also sets the DB buss conditions. When all these signals are known valid, the VALID signal is asserted. Thus the leading edge of VALID/ can be used as a cycle initiator if needed. When the peripheral is able to perform, it asserts READY. The controller then pulses STRB/, and then deasserts VALID. The peripheral must then deassert READY within TFAULT, else a fault exists.

Note 6: (for simple systems) STRB is normally asserted, for communication, only when VALID is asserted. However, for synchronization purposes, STRB may be asserted while VALID is deasserted. This must be ignored by all normal peripherals. At this time the DB buss can provide information to systems units (such as front panel displays, etc.) and the MAR buss may provide a refresh address for dynamic memories.

Note 7: Assertion of VALID normally provokes the response READY. Use of non-existent peripherals or memory is detected by a timeout mechanism of at least 100 microsecs. This delay may be long since this is a fault condition, and must cause an abort. A different mechanism can be used for virtual memory segment absence traps.

DISCUSSION OF PHYSICAL IMPLEMENTATION ASPECTS

The above can be implemented on single sided cards, and on commercially available prototype boards. As an example only, Vero board part #12682 is 8 by 8 inches, providing 64 square inches of circuit space. Vero also provides a mating socket, usable for preliminary work, but not reliable for production. An excellent (and cheaper) alternative socket is AMP's part #5-583485-4 together with polarizing pin part #583714-1 which cost $3 to $5 in medium quantity. These sockets provide contact redundancy via the component side of the board, and have given minimal problems. At present the AMP sockets must be ordered in minimum quantities of 200, but common use of this buss will rapidly cause them to be available via distributors.

An 8 bit version of this buss, usable for most existing 8 bit processors, and built with the above hardware, has been in use at the Yale School of Medicine since 1975.

In our implementation, card orientation is controlled, on entry, by the backplane mounting chassis rails, and final card positioning by the polarizing pin in the socket.

Note that spare pins are assigned between power pins. This minimizes the risk of component damage resulting from card insertion/removal with power-on, which will always occur by accident.

In the above, buss access and interrupt priority is controlled by physical location in the card cage. Therefore it is essential that any modules not specifically using the "daisy chained" priority control signal pairs DMACK, DMAEN and INTEI, INTEO should jumper these signals together. These signal pairs are the only exception to direct parallel bussing of all signal pins.

MISCELLANEOUS RAMBLINGS ON VARIOUS SUBJECTS

On "On Card" Regulation. Consider a typical memory card, containing 32 chips each using 100 MA at 5V. Power drain is 3.2A, and on card dissipation is 16 watts.

If an on card regulator is used, the normal minimum input voltage will be about 7.5V at the trough of the input ripple. This ripple may well be 0.5V peak to peak, or more if the rectification system has small filter capacitors. This condition arises at low line voltage, say 100 V.A.C., for calculation convenience. If the line voltage rises to 130V and the system become lightly loaded, the unregulated voltage will rise to 8.0 X 1.3, or 10.4 volts. This must still supply 3.2A to the card, and on card dissipation has risen to 34 watts. The extra, and unnecessary, 17.8 watts must be unloaded somewhere, and the result is that the ambient temperature rises. Now the 0-70 degree chips may well have to operate at 90 degrees or more.

However, when the main power supply provides regulation, the total power consumption remains the same, but the extra 17.8 watts can now be dissipated at the convenient backpanel location, out in free air. The power supply wires to this card have suddenly become able to carry 18 odd watts of heat out to the world. This heat pump is also free.

Thus any supplies which must provide heavy currents should be centrally regulated. Bear in mind that TTL noise problems usually do not arise via the VCC line, but via the ground line.

On 8 Bit Computers on the 16 Bit Buss. Since all memory provides 16 bit words, and such devices as the 8080 require 8 bit words, provision has been made to read and write in 8 bit bytes.

The input to the 8080 will be provided by a 2 to 1 multiplexer, 8 bits wide. The low order address bit from the 8080 will control this multiplexer, and the MAR00 bit. The remaining 15 bits of the 8080 address will control MAR01 thru MAR15.

On write, the multiplexer direction is reversed. The low order address bit will control the use of WRTLO and WRTHI memory write control signals. No actual multiplexer is needed, since the 8080 data output can be duplicated on both the high and low order DB bytes.

Complications arise with the I/O instructions, since the 8080 normally places the low order port bit on the low order address bit. Reads and writes can then be performed using the addressing method described above. However, on some ports the act of reading should reset a status bit, DATA'READY, (or whatever name is assigned). This read will occur for either a word read or a low order byte read. If the peripheral returns, for example, a status byte on the high order address, and the data byte on the low order address, a word read will return both status and data, and reset the flag. A high order byte read will return status, without affecting the flag, and a low order byte read will return data and reset the flag. This implies that a data byte should always occupy the low-order byte. Sixteen bit CPU's and 8 bit CPU's can use appropriate software to control the peripheral I/O, the low order 8080 address bit is also gate out to MAR23 (an otherwise unused address bit). The peripheral being read can now tell which port is actually being accessed, and control its status flags accordingly.

This complication will also arise with memory-mapped I/O, whenever the action of being read must be noted.

The above treatment should, in essence, be capable of interfacing any existing 8-bit micro-computer to this buss.

On Card Dimensions. Cards for this system must be 8 inches wide for compatibility with the specified sockets. This allows the Vero board, or its equivalent, to be used at any time, and plugged into the system. This card is 8 inches long, so the system must allow for cards at least this length. This area is sufficient to implement most subsystems. However, a need may arise to interface with existing S100 cards, which are 6.5 by 10.5. The 6.5 inch dimension will piggyback onto the 8 inch width, but, allowing for the connector, a carrying card of at least 11 inches will be required. Therefore I suggest that the standard card be 8 by 11 inches.

Note that the 8 inch dimension allows systems to be implemented in an 8¾ inch standard card cage, with front card insertion. This avoids losing card access when racked with other instruments. At the same time, a 19 inch by 8¾ inch by 15 inch (width, height, depth) instrument is conveniently sized, and can house virtually any required capability. Of course, other housing schemes for this card size can be used.

Notice also that all cards will never require routing, since they can be cut square. This reduces manufacturing costs, and eases prototype construction.

On the Need for a Buss Specification Now. Several 16 bit chips and/or chip sets have already appeared, including the TI 9900, Data General Micronova, Fairchild's Microflame, DEC's LSI 11 set, National's Pace, GI's CIP1600, Zilog Z8000, Intel 8086, Motorola 6809 chips, plus others of which I am not yet aware.

I expect that the equivalent of the original Altair will soon appear, using one of these devices, (it may already be in the design stage) and again create a de-facto standard, with attendant problems. Hopefully, if this standard is published early and often, that design will be based on this buss. In addition, if firms are convinced that the buss is sound and will be used, peripherals and/or computers or other devices can be designed now, with assurance of eventual compatibility. Therefore I have pointed out sources for connectors, prototype cards, and so forth. My own gain will be the ability to buy other people's designs, thus enabling me to do more amusing things than re-inventing or re-packaging wheels.

On Reliable Memory Systems. Extremely reliable memory systems can be built using error correcting codes, with optional recording of memory error occurence. This requires extra bits, and is not economically practical for 8 bit systems. However, once the memory is sufficiently large (say 32K bytes) and 16 bits wide, the system becomes practical. This requires sufficient card area to mount the extra memory bits, and the error correcting logic. The 8 × 11 inch card should be sufficient using existing 4K memory chips, and will easily allow 65K byte modules using 16K chips.

For a 16 bit word, several error control options exist:

1. None. Most 8 bit systems use this.
2. Parity. 1 extra bit. Detects 1 bit errors, not 2.
3. Hamming corrector, 5 extra bits. Corrects 1 bit errors, does not detect all 2 bit errors.
4. Hamming corrector, 6 extra bits. Corrects 1 bit errors and detects all 2 bit errors.

Such systems will normally be able to correct all single bit errors, and detect all 2 bit errors, and most higher order errors. The perr signal has been provided as a means of creating extremely high priority interrupts, or such other action as is desired, on these uncorrectible errors. It may also be used to report parity errors from simple memories using single parity bits, or of course it can be ignored.

On the "Broadcast" Feature. Buss controllers will be able to "broadcast" information on the MAR and DB lines by outputting STRB without either a VALID or an INACK signal. This has been envisioned as a means of detecting such things as fetch cycles, controlling memory refresh cycles, etc. Forty bits are available for this purpose. Output may include the identity of the present buss controller. Use of this feature will require agreement on conventions, and is likely to become chaotic. However, excessive restrictions are likely to cause designers to violate the conventions.

I propose the following conventions:

1. Buss controller identity appears on MAR20 thru MAR23.
2. Refresh address appears on MAR00 thru MAR07 (8 bits).
3. Remaining 16 data bits and 12 address bits are free.

On Buss Speed. Firstly, buss speed should not be confused with CPU clock rate. Buss speed can be measured in bandwidth, i.e. bytes per second transferrable. This is, of course, limited by the peripherals used. The CPU clock rate controls the CPU internal micro instruction timing, and only some of these micro cycles, in any device, require buss access.

Computers in the "mini" class, including powerful machines such as the HP3000, DEC PDP 11's, NOVAS, VARIAN, etc., rarely obtain a possible buss rate of 10 megabytes/sec. In fact, buss rates of 2 megabytes are far more common. Notice that a 2 foot wire will normally have a down and back propagation time of 4 nanoseconds, and that several such cycles are necessary to damp ringing, etc. This purely physical fact sets an upper limit for non-pipelined systems, including this buss.

The following minimum/maximum specifications are proposed:

- TDV (Buss data valid to leading edge of VALID/ signal)
  20 ns. (min.) No maximum.
- TRS (READY/ leading edge to leading edge of STRB/)
  20 ns. (min.) To be specified for all buss controller. No maximum.
- TDR (DB data on read valid before leading edge of READY/) May be negative. This allows for "anticipation" systems where data is known to settle before the buss controller can actually read the lines. This value should be specified for all "readable" modules.
- TSS (STRB/ pulse width)
  100 ns. minimum. No maximum. To be specified for all bus controllers and writeable modules. This allows use as an actual write pulse to memory chips, yet module specifications permit compatibility to be evaluated.
- TSV (Trailing edge of STRB/ to trailing edge of VALID/)
  0 ns. minimum. 100 ns. maximum.
- TVD (Trailing edge of VALID/ to buss data becoming invalid) 20 ns. minimum, i.e. all data to be held 20 ns.
- TVR (Trailing edge of VALID/ to trailing edge of READY/) 100 ns. maximum. Specify for all modules.

Note that these specifications enable, but do not require, a buss transfer rate of 10 megabytes (better than 1 word per 200 nanosecs).

On Devices and Systems on the Buss. Devices for use on the buss include the "normal" modules, which already exist for 8 bit busses at least, simple CPU's, for central use only, memory module, and I/O devices.

However, several elegant devices can be built:

A. CPU's with resident memory, not requiring buss access to function. The memory may or may not be accessible over the buss, and if so may be remapped to new addresses. This CPU however can access all buss devices. Thus it may be set a task, and function entirely off-line. It may access as wide a database as is required, and signal readiness, idleness, completion, etc. For example, several such devices may be software programmed to perform arithmetic. A master device can partition an arithmetic statement in terms of precedence (i.e. multiplication before addition, parenthesized sub-expressions first, etc.) and pass calculations at the same precedence level to the slave CPU's. This will effectively allow evaluation time to depend on the number of precedence levels, rather than on the number of operations. Similarly, vector and matrix operations may be partitioned. The number-crunching possibilities are endless. Of course, this device can also be used as an I/O processor, etc.

B. Consider a memory device containing, say, thirty-two 64 KBIT CCD devices, and two 4 KBIT memory modules. The CCD's hold about 256 KBYTES (i.e. 1 floppy disc) and are addressable, on the buss, as a 128K word space. Two active segment I.D. registers exist, 1 for each 4 KBYTE memory segment. The registers identify the CCD segment held within the high speed RAM block. On a buss read/write, if the addressed segment is stored in the RAM, access is immediate. If not, a segment is stored (based on last used of the two, for simple segment management hardware) and the desired segment loaded from the CCD's. This transfer will probably require 1 to 2 millisecs. However, the probability of requiring a transfer should normally be small. Thus average access time can easily be in the order of 500 ns to 1 microsec. In use, this device would normally be loaded with the contents of a floppy disc, and operated on. The access time would drop by possibly 5 orders of magnitude.

Notice that implementing such a device requires a sufficiently large card, otherwise kluges such as double cards, or external buss extensions, etc. must appear.

A Z80 Micro-disassembler

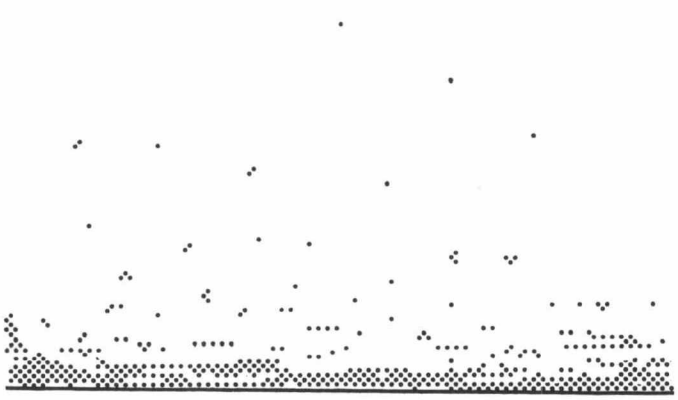
BY HOLGER PETERSEN
Samwerstr. 21
2300 KIEL 1
West Germany

Here is a small Z-80 program (less than 512 bytes) which will display the memory content 1 to 4 bytes at a time depending on the length of the OP-code. I call it micro-disassembler. I think it might be useful to some of your readers; if you do not think so, please publish a small note with my address and the remark that I am searching contacts to other German Z-80 users. Since September '78 I have owned an SDS Z-80 starter kit and have bought an Artec Memory Board which worked at once. I bought a video board from JADE (less than 3 weeks delivery to Germany!) but it required some modifications to select logic. I interfaced a Selectric Printer and a surplus keyboard, both imported from England since computer parts are still too expensive in continental Europe (the 4K-PET is 1895DM = $1000!).

My program works as a subroutine in the following manner: registers HL are to point to the beginning and DE to the end of the memory range to disassemble. The first action is a carriage return. Then there is a test if DE is still greater than HL. If not, HL is printed followed by 3 spaces. Then follows the main subroutine that determines the number of bytes in the OP-code and returns it in B. They are then printed with a space between them. After this, C is tested, if a special action is needed. If not, the loop is completed. Special actions are:

1) another carriage return after absolute jumps and returns,
2) print the destination of relative jumps.

By the way, of the 1276 possible instructions ((256 X 5) – 4) 698 are used by MOSTEK. Has anyone explored the others??



The main routine works like this: B is loaded with a table-address, C with the first byte; A is loaded with the table-byte, and it is tested if bit 7 is set. This is the case with the special OP-codes DD, ED, and FD which vary in length. If it is not set, the length must be 1, 2 or 3. So we put it in B, and the remaining bits 2-6 into C to be tested later. Bit 6 is used to produce another CR, and bit 5 to indicate a relative jump. Bit 2 is used to indicate the 4-byte ED-codes and special DD/FD-codes which are only 1 byte longer than their HL-depending counterparts. I plan to use the remaining bits 3 and 4 to indicate non-relative jumps and calls in a relocater-program.

An ED-code is found by testing for zero in C, where XD now is. 0 in bit 4, now in C; we get first the table-value for the byte following the DD, ED or FD into A and clear C. For ED, we put tentatively 2 into B, test bit 2 of the table value and return if not set. Else we return with a 4 in B. With DD and FD we test again bit 2 of the table-value and add 2 to its byte-number if not set, else only 1 is added. In the offset calculation, first 3 spaces are made, then we decrement HL and get the offset to A; and increment HL for proper return. We test if it is a forward or backward jump and calculate accordingly. After printing the destination we return to the main loop. After address 2693 hex you may shorten the code if you have appropriate monitor-routines:

2693	prints HL in hex
269C	prints A in hex
26AD	makes a lowercase"o" instead of "zero" to distinguish better from uppercase D (depends on my character-set)
26B7	outputs a space
26BB	shall utput a CR, for slowing down my memory-mapped ed TV or for printing comments, it waits for input
26C6	inputs the parameters to HL and DE.
26B3	is the only reference to your output-routine; it must not disturb any register
26BB	and
26E7	are the references for your input-routine; only AF may be used.

I would be glad to supply tapes to readers in Kansas City/ INTEL-Hex format or on TRS-80 level II or on NORTH STAR minifloppy in exchange for some programs you will send!

```
THE CODE produced by the nrogramm    MY COMMENTS

2600 27FF

2600   CD C6 26          CALL Parameter-input
2603   CD BB 26          CALL Cariage-return
2606   CD 8D 26          CALL Compare HL with DE
2609   F0                RET POS
260A   CD 93 26          CALL Print HL
260D   06 03             LD B,$03
260F   CD B7 26          CALL Space
2612   10 FB    260F     DJNZ three times
2614   CD 28 26          CALL Number
2617   7E                LD A,(HL) get first byte of OP-Code
2618   CD 9C 26          CALL Print a
261B   CD B7 26          CALL Space
261E   23                INC HL
261F   10 F6    2617     DJNZ other bytes too
2621   AF                XOR A clear flags
2622   B1                OR C  get specials
2623   C4 5B 26          CALL NZ  if  "
2626   18 DB    2603     JR back to carriage-return

2628   06 27             LD B,$27   Table-base-adress
262A   4E                LD C,(HL)  OP-Code to C
262B   0A                LD A,(BC)  table-value to A
262C   CB 7F             TST 7,A
262E   20 09    2639     JR NZ  if specials
2630   F5                PUSH AF save A
2631   E6 03             AND $03 leave number
2633   47                LD B,A  store in B
2634   F1                POP AF  get old A
2635   E6 7C             AND $7C leave specials
2637   4F                LD C,A  store in C
2638   C9                RET

2639   CB 61             TST 4,C is it ED ?
263B   23                INC HL  point to second Byte
263C   4E                LD C,(HL)  put into C
263D   2B                DEC HL  restore HL
263E   0A                LD A,(BC)  Table-value to A
263F   0E 00             LD C,$00   Claer C
2641   20 08    264B     JR NZ jump if not ED
2643   06 02             LD B,$02 tentativly 2 bytes
2645   CB 57             TST 2,A  or more ?
2647   C8                RET Z    if two-byte code
2648   06 04             LD B,$04 it has 4 bytes
264A   C9                RET

264B   CB 57             TEST 2,A same test
264D   20 06    2655     JR NZ    jump if code without displacement
264F   E6 03             AND $03 leave number
2651   C6 02             ADD $02 but it has 2 more
2653   47                LD B,A
2654   C9                RET

2655   E6 03             AND $03 as above
2657   C6 01             ADD $01 but only one more
2659   47                LD B,A  save number in B
265A   C9                RET

265B   CB 69             TST 5,C relative jump ?
265D   C4 66 26          CALL NZ so print adresse
2660   CB 71             TST 6,C extra return?
2662   C4 BB 26          CALL NZ Carriage-return
2665   C9                RET

2666   F5                PUSH AF save A
2667   06 03             LD B,$03
2669   CD B7 26          CALL Space

266C   10 FB    2669     DJNZ three times
266E   E5                PUSH HL save HL
266F   CD 79 26          CALL relativ adress
2672   6F                LD L,A
2673   CD 93 26          CALL Print HL
2676   E1                POP HL
2677   F1                POP AF
2678   C9                RET

2679   2B                DEC HL point to rel.displacement
267A   7E                LD A,(HL) get into A
267B   23                INC HL restore HL
267C   17                RLA    get sign bit to cary
267D   38 05    2684     JR C   if back-jump
267F   1F                RRA    restore A
2680   85                ADD L  get low byte
2681   D0                RET NC
2682   24                INC H  if cary from above
2683   C9                RET

2684   1F                RRA    restore A
2685   ED 44             NEG    negate A
2687   47                LD B,A save in B
2688   7D                LD A,L
2689   90                SUB B  get low byte
268A   D0                RET NC
268B   25                DEC H  if cary
268C   C9                RET

268D   E5                PUSH HL save HL
268E   AF                XOR A   clear flags
268F   ED 52             SBC HL,DE  set flags
2691   E1                POP HL
2692   C9                RET

2693   7C                LD A,H
2694   CD 9C 26          CALL print A
2697   7D                LD A,L
2698   CD 9C 26          CALL Print A
269B   C9                RET

269C   F5                PUSH AF save A
269D   0F                RRCA
269E   0F                RRCA
269F   0F                RRCA    shift low to high
26A0   0F                RRCA    nibble
26A1   CD A5 26          CALL    Print nibble
26A4   F1                POP AF  restore A
26A5   E6 0F             AND $0F mask nibble
26A7   C6 90             ADD $90 the
26A9   27                DAA     decimal adjust
26AA   CE 40             ADC $40 trick to get
26AC   27                DAA     ASCII from HEX
26AD   FE 30             CP "0"  is it zero
26AF   20 02    26B3     JR NZ   jump if not
26B1   3E 6F             LD A,"o" load lower case o
26B3   CD 35 23          CALL external output routine of User
26B6   C9                RET

26B7   3E 20             LD A," "  space
26B9   18 F8    26B3     JR     to output

26BB   CD F8 25          CALL external input of User
26BE   CD B3 26          CALL output
26C1   FE 0D             CP "CR"  was it RETURN
26C3   20 F6    26BB     JR NZ    loop back if not
26C5   C9                RET      else return

26C6   CD CC 26          CALL input to DE
26C9   CD B7 26          CALL Space
```

```
26CC   CD D1 26          CALL input to HL
26CF   EB                EX DE,HL
26D0   C9                RET

26D1   CD DA 26          CALL input to A
26D4   67                LD H,A save in H
26D5   CD DA 26          CALL input to A
26D8   6F                LD L,A save in L
26D9   C9                RET

26DA   CD E7 26          CALL    nibble-input
26DD   07                RLCA
26DE   07                RLCA
26DF   07                RLCA shift nibble
26E0   07                RLCA
26E1   47                RLCA LD B,A
26E2   CD E7 26          CALL    nibble-input
26E5   B0                OR B get full byte
26E6   C9                RET

26E7   CD F8 25          CALL user-input
26EA   D6 30             SUB $30
26EC   FE 0A             CP $0A bigger then 9 ?
26EE   D8                RET C  return if not
26EF   D6 07             SUB $07 it was a-f
26F1   C9                RET

26F2   00                NOP
26F3   00                NOP  free space for you !!!
26F4   00
26F5   00
26F6   00
26F7   00
26F8   00
26F9   00
26FA   00
26FB   00
26FC   00
26FD   00
26FE   00
26FF   00                NOP


2700   01 03 01          This is the table
2703   01 01 01          most entries are 01
2706   02                so they disassemble
2707   01 01 05          as the three-byte code
270A   01 01 01          LD BC immediate
270D   01 02 01
2710   22 03 01
2713   01 01 01
2716   02
2717   01 62 05
271A   01 01 01
271D   01 02 01
2720   22 07 07
2723   05
2724   01 01 02
2727   01 22 05
272A   07
272B   05
272C   01 01 02
272F   01 22 03
2732   03
2733   01 01 01
2736   02
2737   01 22 05
273A   03
273B   01 01 01
```

```
273E   02
273F   01 01 01
2742   01 05 01
2745   01 01 01
2748   01 01 01
274B   01 01 01
274E   01 01 01
2751   01 01 05
2754   01 01 01
2757   01 01 01
275A   01 05 01
275D   01 01 01
2760   01 01 01
2763   01 01 01
2766   01 01 01
2769   01 01 01
276C   01 01 01
276F   01 01 01
2772   01 05 01
2775   01 41 01
2778   01 01 01
277B   05
277C   01 01 01
277F   01 01 01
2782   01 01 01
2785   01 01 01
2788   01 01 01
278B   01 01 01
278E   01 01 01
2791   01 01 01
2794   01 01 01
2797   01 01 01
279A   01 01 01
279D   01 01 01
27A0   01 01 01
27A3   01 01 01
27A6   01 01 01
27A9   01 01 01
27AC   01 01 01
27AF   01 01 01
27B2   01 01 01
27B5   01 01 01
       27B8   01 01 01
27BB   01 01 01
27BE   01 01 01
27C1   01 03 43
27C4   03
27C5   01 02 01
27C8   01 41 03
27CB   02
27CC   03
27CD   03
27CE   02
27CF   01 01 01
27D2   03
27D3   02
27D4   03
27D5   01 02 01
27D8   01 01 03
27DB   02
27DC   03
27DD   82
27DE   02
27DF   01 01 05
27E2   03
27E3   05
27E4   03
27E5   05
27E6   02
27E7   01 01 45
```

```
27EA   03
27EB   01 03 82
27EE   02
27EF   01 01 01
27F2   03
27F3   01 03 01
27F6   02
27F7   01 01 05
27FA   03
27FB   01 03 82
27FE   02
27FF 01 00 00        end of table is 27FF !
```

The End

MODULAR PROGRAMMING with the APPLE II

BY R. F. ZANT
Dept. of Accounting and Information Systems
North Texas State University
Denton, TX 76203

Both the modular and structured programming approaches encourage the use of subroutines within a program. The modular approach also encourages the use of common routines in different programs. For example, the same sort or error handling routine can often be used in several programs.

The utilization of common routines is practical only if external subroutines are allowed in the language being used or if there is a way to easily insert common code into different programs. Neither of these capabilities are available in the BASIC language on the Apple II. External subroutines can be written in machine language, but not in BASIC.

Several other BASIC language systems solve this problem with a system command that loads and attaches a second program to the current program in main storage. Common routines are written with high (or low) statement numbers and saved on tape or disk. Then, using the command, they are added to different programs as needed.

The two assembly language programs reproduced below provide this capability on the Apple II. The first program is for integer BASIC; the second is for APPLESOFT. Both programs are used according to the following procedure:

1. Load assembly language program
2. Load program A
3. Call 777
4. Load program B
5. Call 777

With integer BASIC, program A must be the routine with the highest numbered statements. With APPLESOFT the exact opposite is true; program A must be the routine with the lowest numbered statements. The following listing demonstrates the use of the APPLESOFT version of the routine with a disk system.

```
]NEW
]30  PRINT A,B
]40  END
]SAVE TEST
]BLOAD APPEND FP
]NEW
]10  A=10
]20  B=A*2
]CALL 777
]LOAD TEST
]CALL 777
]LIST
```

```
1000 *  APPEND INT
1010 *
1020 *  ROUTINE TO APPEND
1030 *  INTEGER BASIC PROGRAMS
1040 *  ON THE APPLE II
1050 *
1060 *  WRITTEN BY:
1070 *  ROBERT F. ZANT
1080 *
1090 *  01-07-79
1100 *
1110 *  ASSEMBLED USING
1120 *  S-C ASSEMBLER II
1130 *
1140 HIMEML .EQ $4C
1150 HIMEMH .EQ $4D
1160 PENDL  .EQ $CA
1170 PENDH  .EQ $CB
1180 SAVEL  .EQ $306
1190 SAVEH  .EQ $307
1200        .OR $300
1210 FLAG   .DA #*-*
1220        LDA FLAG
1230        BNE SECOND
1240        INC FLAG    ENTERED FOR
1250        LDA HIMEML  FIRST ROUTINE
1260        STA SAVEL
1270        LDA HIMEMH
1280        STA SAVEH
1290        LDA PENDL   MAKE END OF
1300        STA HIMEML  FIRST ROUTINE
1310        LDA PENDH    HIMEM
1320        STA HIMEMH
1330        RTS
1340 SECOND DEC FLAG    SECOND ROUTINE
1350        LDA SAVEL   NOW LOADED
1360        STA HIMEML
1370        LDA SAVEH   RESTORE HIMEM
1380        STA HIMEMH
1390        RTS
1400         EN
```

Figure 1. Integer BASIC Version

```
1000 *  APPEND FP
1010 *
1020 *  ROUTINE TO APPEND
1030 *  APPLESOFT PROGRAMS
1040 *  ON THE APPLE II
1050 *
1060 *  WRITTEN BY:
1070 *  ROBERT F. ZANT
1080 *
1090 *  01-07-79
1100 *
1110 *  ASSEMBLED USING
1120 *  S-C ASSEMBLER II
1130 *
1140 BEGINL .EQ $67
1150 BEGINH .EQ $68
1160 SAVEL  .EQ $304
1170 SAVEH  .EQ $305
1180 DISPL  .EQ $306
1190 DISPH  .EQ $307
1200        .OR $300
1210 FLAG   .DA #*-*
1220        LDA FLAG
1230        BNE SECOND
1240        INC FLAG
1250        LDA BEGINL
1260        STA SAVEL
1270        LDA BEGINH
1280        STA SAVEH
1290 MAIN   LDY #$00
1300        LDA (BEGINL),Y
1310        TAX
1320        BNE CONT
1330        INY
1340        LDA (BEGINL),Y
1350        BNE CONT1
1360        RTS
1370 CONT   INY
1380        LDA (BEGINL),Y
1390 CONT1  STA BEGINH
1400        STX BEGINL
1410        JMP MAIN
1420 SECOND DEC FLAG
1430        SEC
1440        LDA BEGINL
1450        SBC SAVEL
1460        STA DISPL
1470        LDA BEGINH
1480        SBC SAVEH
1490        STA DISPH
1500 LOOP   LDY #$00
1510        LDA (BEGINL),Y
1520        BNE NOTEND
1530        INY
1540        LDA (BEGINL),Y
1550        BEQ RETURN
1560        LDY #$00
1570        LDA (BEGINL),Y
1580 NOTEND CLC
1590        ADC DISPL
1600        TAX
1610        STA (BEGINL),Y
1620        INY
1630        LDA (BEGINL),Y
1640        ADC DISPH
1650        STA (BEGINL),Y
1660        STA BEGINH
1670        STX BEGINL
1680        JMP LOOP
1690 RETURN LDA SAVEL
1700        STA BEGINL
1710        LDA SAVEH
1720        STA BEGINH
1730        RTS
1740        .EN
```

Figure 2. APPLESOFT Version

Letters

OCR

Dear Editor:

I feel that the idea of machine-readable printed programs as a medium for software exchange has great possibilities. The major requirement for a standard is that (1) Reasonably-priced and compatible equipment must be readily available (from more than one manufacturer); (2) The OCR reader algorithm/software needs to be available cheaply with the reader (or be in the public domain, free from copyrights); (3) For an OCR format to be a universal medium of software exchange, as well as being able to read the OCR format with an inexpensive wand, it should be possible for almost anyone with a printer to list software in the format.

I feel that though the "Paperbytes" concept is basically an excellent idea, it falls short of these requirements. (1) At present, the only supplier of wands "designed for the Paperbytes format" seems to be Microscan—which doesn't advertise in *Byte*, and doesn't appear to answer letters. Wand options do not seem to be available with major brands of hobby computers from computer stores. (2) Despite the foreword to *Byte's Bar Code Loader Book* – stating that they do not want to restrict use of the format in any way—there was a paper with a Microscan copyright notice inserted in the front of the book. (3) The Paperbytes format is proportionally spaced (light spaces between characters are equal width, black 1 and 0 characters differ in width), but the majority of hobbyist printers do not print with proportional spacing—much less print with special OCR fonts.

Already there are several commercial OCR wand systems; the simplest will read credit card numbers (with check digits), more advanced systems will read names in a variety of type fonts from visiting cards. A simple system to read (source or object) hex dumps printed in elite (12 pitch) or pica (10 pitch) would be a good start for a hobby standard. *Does anyone know a source of reasonably priced wands that could be used for such a job?*

On the subject of cheap software exchange media: although several hobby microcomputer systems connect to the user's audio cassette recorder, Casio seems to be the first manufacturer to provide a calculator that does. The Casio FX502P is a 256 step 22-memory programmable calculator that allows 9 levels of subroutine nesting and about 10 labels for program units (includes subroutine and program labels; jump address labels within a program are additional). Program units may be independently loaded, stored or renumbered. It shouldn't be too long before someone incorporates one of the logically-controlled audio cassette decks into a calculator or microcomputer system, and provides a type operating system which allows programs to be chained automatically. Some of the audio tape decks have three heads for read-and-verify, and some can find the "nth" item (tune or program) by searching the tape and counting the pauses between items. Being able to use the cassette deck for audio as well as computing would justify the financial outlay to many hobbyists.

Yours faithfully,
K.S.Wilkinson

CPO Box 1748
Tokyo, JAPAN 100-91

FURTHER PRUNING TO tiny-c

Dear Suzanne:

A correction and some addenda to my article on tiny-c topiary (*DDJ* #36). In the docfile entry for *globals* I wrote: "Unlike big C, tiny-c can't dynamically allocate space for global arrays." Well, big C can't do it either. First mistake I ever made.

As to the addenda, when I saw my article in print I realized that there were no references to books and papers on C. Let me list some here.

1. Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. New Jersey: Prentice-Hall, 1978.
   This is the C bible, so of course it comes in two parts. The first and longer section is a thorough introduction to the language, the second a reference manual that defines it.
2. C.T. Zahn. *C Notes: A Guide to the C Programming Language.* New York: Yourdon Press, 1979.
   Kernighan and Ritchie assume some programming knowledge on the part of the reader; Zahn assumes more. His intention, he says, is to fill the gap between the two halves of K & R's book. They provide an introduction and a definition; he offers terse discussions of each feature, with helpful hints for the working programmer.
3. Brian W. Kernighan. *Programming in C—A Tutorial*. Murray Hill, New Jersey: Bell Laboratories, undated.
   Much the same material as the first part of (1), condensed into a shorter course. Kernighan is a wizard writer, and this is probably the best introduction to C for casual users or interested nonusers.
4. Dennis M. Richie. *C Reference Manual*. Murray Hill, New Jersey: Bell Laboratories, undated (1974).
   Very close to the second part of (1), which apparently supersedes it and which certainly got more careful proofreading. (This paper has two sections numbered 2.3, for example.)
5. D.M. Ritchie, S.C. Johnson, M.E. Lesk, B.W. Kernighan. "The C Programming Language." *The Bell System Technical Journal*, Vol. 57, No. 6, Part 2, July/August 1978, p. 1991.
   The best source I know for the history and philosophy behind C. A good supplement to (1).
6. S.C. Johnson and D. M. Ritchie. "Portability of C Programs and the UNIX System." *The Bell System Technical Journal*, Vol. 57, No. 6, Part 2, July/August 1978, p. 2021.
   An account of how the UNIX operating system, written in C, was moved from its PDP-11 birthplace to an Interdata 8/32.

7. M.E. Lesk. "The Portable C Library." Murray Hill, New Jersey: Bell Laboratories, undated.
Runtime routines, mostly I/O, for Bell's C compiler.
8. D.M. Ritchie. "A New Input-Output Package." No publication data, but obviously a Bell Labs memorandum.
The package described here is meant to replace the earlier portable library outlined in (6).

Those are the C references currently on my desk. Perhaps someone out there can add to the pile?

Sincerely,
Les Hancock

305 Lexington Avenue
New York, NY 10016

INFLATION HEDGE!

Dear Suzanne:

My husband, a systems engineer, is a computer nut. My formerly attractive guest room, now with additional objects like a microcomputer, printer, floppy discs, etc., is going full speed at all hours. I swear my husband creates his own problems, as he always has debugging to conquer. Occasionally his programs are debugged by a fellow nut, who then gets paid off by being taken out to a nice restaurant for dinner (and don't forget the wife or girlfriend). Anyway you look at it, we are minus $50.

My husband recently became a subscriber to *DDJ* and found a better-appreciated method of paying off a debt. He gives his debugger friend a year's subscription to *DDJ* at $15 and everyone is happy.

Hey, we saved another $35 this week.

Sincerely,
NCW

A GIFT IN THE MORNING MAIL

Dear Madam:

Today I am posting an index to *Dr. Dobb's Journal*, Vol. 1. I am sending it in two parts, *i.e.*, pp 1-14 and 15-28, in order to fool the Post Office: they may lose one, but surely not both! If they fail to arrive in a reasonable time, please let me know, and I'll send spare copies. Assuming I have some copyright in it, I make it over to you.

I should mention that my thanks are due to John Fowle, who helped with the proofreading.

I am, Madam,
Your obedient servant,
R. Larkin

49 Courthope Rd.
London NW3 2LE
United Kingdom

This is the kind of thing that can make an editor's day . . .
–SR

SAVING/RESTORING MACHINE STATUS

Dear *Dr. Dobb's*:

I am sending the enclosed code as a further example of saving machine status. The routines submitted were written about one year ago by one of my high school students, Alex Caemmerer. The routines are different from the others in that this set saves the stack pointer as well as the registers. This addition is useful if the main program doesn't have enough stack levels, i.e., most basics only have a limited stack range. I feel they are optimized and quite efficient.

I completely enjoy your magazine and am glad to be able to contribute a small program for your consideration.

Sincerely,
John Bockelmann

174 Jefferson Ave.
Tenafly, New Jersey 07670

```
A:RESTR.PRN

CP/M MACRO ASSEM 2.0   #001    RESTORE TOTAL MACHINE STATUS

                       TITLE   'RESTORE TOTAL MACHINE STATUS'
                       ; THIS CALLED ROUTINE WILL COMPLETELY RESTORE
                       ; ALL REGISTERS, PC, SP TO ALLOW FOR A GRACEFUL RETURN
                       ; TO A MAIN PROGRAM AFTER PROCESSING A SUBROUTINE.
                       ; THIS ROUTINE IS USED TO RETURN FROM A SUBROUTINE
                       ; WHICH USED 'SAVE' TO SAVE THE MACHINE
                       ; STATUS. THIS ROUTINE WOULD BE USED IN CONJUNCTION
                       ; WITH BASIC OR OTHER HIGH LEVEL LANGUAGE TO LINK
                       ; MACHINE LANGUAGE SUBROUTINES.
                       ;
                       ; ROUTINE IS CALLED WHEN READY TO LEAVE THE MACHINE
                       ; LANGUAGE SUBROUTINE.
                       ; ROUTINE HAS NO EXTERNAL REFERENCES
                       ; ROUTINE USES THE SAME DATA SPACE (RETADR) AS DOES
                       ; SAVE
                       ;
                       ;
0000 E1        RESTR:  POP     H       ; PUT RETURN ADDRESS IN H&L
0001 220E00            SHLD    RETADR  ; SAVE RETURN
0004 F1                POP     PSW     ; RESTORE ALL REGISTERS
0005 C1                POP     B       ; ASSUMES STACK IS 'STRAIGHT' AND AT
0006 D1                POP     D       ; THE SAME LEVEL AS IT WAS WHEN
                                       ; 'SAVE ' WAS CALLED
0007 E1                POP     H       ; REALLY THE OLD STACK POINTER
0008 F9                SPHL            ; RESTORE OLD SP
0009 2A0E00            LHLD    RETADR  ; GET RETURN ADDR
000C E3                XTHL            ; RESTORE H&L FROM OLD STACK
000D C9                RET             ; RETURN TO CALLING PROGRAM
                                       ; NEXT INSTRUCTION WOULD RETURN TO
                                       ; MAIN PROGRAM
000E           RETADR: DS      2       ; DUMMY FOR THIS ASSEMBLY
                                       ; DELETE IN REAL PROG, ALREADY SET UP
                                       ; IN SAVE
0010                   END
```

```
A:SAVE.PRN

CP/M MACRO ASSEM 2.0   #001    SAVE TOTAL MACHINE STATUS

                       TITLE   'SAVE TOTAL MACHINE STATUS'
                       ;
                       ; THIS CALLED ROUTINE WILL COMPLETELY SAVE ALL
                       ; REGISTERS, PC, SP AND ALLOW FOR A
                       ; GRACEFUL RETURN FROM WHERE YOU WERE WHEN
                       ; USED IN CONJUNCTION WITH RESTR WHICH WILL
                       ; COMPLETELY RESTORE MACHINE STATUS.
                       ; THIS ROUTINE IS USEFUL FOR ENTERING SUBROUTINES FROM
                       ; BASIC OR OTHER HIGH LEVEL LANGUAGE
                       ;
                       ; SUBROUTINE HAS NO EXTERNAL REFERENCES
                       ; SUBROUTINE USES ONE DATA SPACE
                       ;
                       ;
9FFF =         STACK:  EQU     9FFFH   ; ARBITRARY LOCATION OF LOCAL STACK
0000 E3        SAVE:   XTHL            ; PUT TOP OF STACK IN H&L
                                       ; H&L NOW ON TOP OF OLD STACK
0001 221300            SHLD    RETADR  ; SAVE TOP OF STACK FOR RETURN
0004 210000            LXI     H,0
0007 39                DAD     SP      ; PUT OLD SP IN H&L
0008 31FF9F            LXI     SP,STACK        ; NEW STACK MUST NE EQU'ED
000B E5                PUSH    H       ; SAVE OLD SP ON NEW STACK (OLD H&L
                                       ; ON TOP OF OLD STACK)
000C D5                PUSH    D
000D C5                PUSH    B
000E F5                PUSH    PSW
000F 2A1300            LHLD    RETADR  ; PUT RETURN ADDR IN H&L
0012 E9                PCHL            ; USED IN PLACE OF 'RET' TO GO
                                       ; BACK TO PROGRAM WHICH CALLED
                                       ; THIS ROUTINE. THIS SHOULD BE THE FIRST
                                       ; CALL OF THE SUBROUTINE
                                       ; ROUTINE ASSUMES OLD STACK HAS 2 LEVELS
                                       ; FOR STORING H&L
0013           RETADR: DS      2       ; SPACE FOR RETURN ADDR FROM STACK
                       ;
                       ;
0015                   END
```

8080 PUSH AND PULL

Dear *Dr. Dobb's*,

In *Dr. Dobb's* #33, page 45, William Moss listed a set of routines to efficiently PUSH and PULL all registers for the 8080 without destroying their contents. Below is listed a slightly improved version.

	# Bytes	# States
William's version	20	172
New version	18	156

Sincerely,
Bruce Smith

519 N. Logan
Lansing, MI 48915

```
0000 E3             0010 PUSHR  XTHL
0001 22 0A 00       0020        SHLD  JP+1      RETURN ADDRESS
0004 E1             0030        POP   H
0005 E5             0040        PUSH  H
0006 F5             0050        PUSH  PSW
0007 D5             0060        PUSH  D
0008 C5             0070        PUSH  B
0009 C3 00 00       0080 JP     JMP   0         RETURN
000C E1             0090 PULLR  POP   H
000D C1             0100        POP   B
000E D1             0110        POP   D
000F F1             0120        POP   PSW
0010 E3             0130        XTHL
0011 C9             0140        RET

SYMBOL TABLE

JP    0009    PULLR 000C    PUSHR 0000
```

WHAT'S AN 8080 COMPATIBLE PROGRAM?

Dear *Dr. Dobb's*:

I enjoyed your April 1979 article on Unspecified 8085 Op Codes. Using these additional instructions should pose no problem, as long as their use is noted in the program documentation. Such a program is, of course, no longer "8080 compatible."

This raises an interesting question though; what constitutes an 8080 compatible program? And what are the software differences between the "8080 compatible" chips?

Each compatible chip (8080A, 8085, Z80) is available in several clock speeds. And the cycles-per-instruction is different for each type. This indicates that any form of software timing loops will not be compatible between different systems.

Intel's 8080A treats its unassigned op codes as no-ops. These codes are (hex): 08, 10, 18, 20, 28, 30, 38, CB, D9, DD, ED, and FD.

The flash register bits 5, 3, and 1 are unassigned.

The DAA op works properly only after addition.

The Parity flag is valid after any logical or arithmetic op using the A register.

The IN and OUT ops places the port address on both Address 15-8 and Address 7-0.

Software differences between Intel's 8080A and "compatible" chips:

There are some 8080A chips, by other manufacturers, with a flag bit for subtract. This allows DAA to work properly for either addition or subtraction.

The 8085 uses the unassigned op codes for special instructions. Flag register bits 5 and 1 are also used. Special interrupt restarts are at addresses (hex) 0024, 2C, 34, and 3C; these are within the Restart area.

The Z80 uses all of the unassigned op codes for its expanded instruction set. It also uses Flag register bit 1. The Parity flag serves a dual role; arithmetic ops use it for 2s complement overflow, and logical ops use it for parity. The NMI Restart is at address (hex) 0066. The IN and OUT ops place the port address on Address 7-0, and A register on Address 15-8.

These differences can prevent an "8080 compatible" program from running on your system, and you may need to know which chip the program was written for.

If you wish to write software for general distribution, that will run on the 8080A, 8085, and Z80, adhere to the following rules:

- Never use the unassigned op codes (listed above).
- Never use Flag register bits 5, 3, or 1.
- Use the Parity flag only after the logical ops AND, OR, XOR, and DAA.
- Use only 4 bytes (instead of 8) for Restarts 4, 5, 6, and 7.
- Do not use memory locations 0066-70 (hex).
- Do not use software timing loops.
- Use the DAA op only after addition.

When writing software for general distribution, I would also recommend the following:

- The instructions HLT, DI, and EI should not be used; it should be presumed the program runs in EI mode.
- The Restart instructions should not be used, since these may be used by the system for interrupts.
- The SP register should always point to a valid, available stack area. In an interrupt-driven system, an interrupt may occur at any time, which will store registers on the stack.
- The IN and OUT ops should only be used by the I/O driver routines, since I/O devices are system dependent.
- All interfacing with system dependent functions (such as I/O drivers, etc.) should be done with a single communications routine. This allows a simple change for differing interface requirements.
- The program code should not modify itself. This allows implementing the program in ROM.
- The program should be written so that it is relocatable at run-time, using a relocating loader. Once the program is loaded and started, it is no longer relocatable.

Following these guidelines will ensure that "8080 compatible" programs are easily transferrable from one system to another.

Yours truly,
William Detwiler

13703 Facade Ave.
Paramount, CA 90723

DOCUMENT ASSEMBLER CODE

Gentlebeings:

Ever since I bought the May issue of *DDJ* about a week ago, I've meant to enter the format program on my machine. But I kept putting it off, because there was so *#$!! Much of it! It sure did look nice, though.

Well, yesterday afternoon I began an orgy of keystroking that has now spanned a period of just over 24 hours, about twelve of which were spent entering the code, modifying it to assemble, adopting the patches for file input, and generally de-bugging it. But, as you can see from this letter, it now works! So I'd like to share my experiences and opinions with you and your readers.

First of all, Mike (I trust he won't mind my familiarity) is to be commended for a very nicely executed translation from the Ratfor source. It is, like the original, clean and wonderfully modularized. And, like most of the code from software tools, it works.

Well, it almost works. The 'HANG' command doesn't work quite right. It's nothing serious, though. All that's needed is to call BRK before setting TIVAL, and adding PUSH/POP D to BRK so that it will save that register pair. Then it works just fine.

So one mandatory logic error has been taken care of. Now on to my experiences getting format to run under CP/M.

First of all, Mike has used some symbols that are reserved by the assembler specifically, the variable NE and the label set produce nasty errors. I changed them to NNE and PSET, but this is somewhat a matter of taste.

Secondly, CP/M is organized around the concept of using a pair of characters to denote a 'NEWLINE' in text files. Not that the OP system itself knows about this, but it seems to be part of the specifications and everything written for CP/M faithfully follows the convention.

Therefore, my printer is strapped so as not to produce a line feed automatically when a line is printed. So I had ?PUTC (A. K. A. $PUTC, but $/S in labels are ignored) check the character it was sending out. If it's a new? line,? PUTC adds a trailing line feed. This works just fine.

A complimentary problem exists on the input side: here I have to ignore the line feeds. Fortunately, I had encountered all these problems previously, and so had an approach worked out for dealing with them.

Enough about the program itself, except to say that I am looking forward to extending and modifying it. It's opinion time!

And there is really just one thing I want to say:

Assembler Code Must Be Documented!!

I don't care how cleanly it's written, or how much you make use of long (as possible), meaningful names, or how well documented a similar program in another language is, or how extensively cross-referenced it is. I feel that at least one comment to every three or four lines is the bare minimum needed to make assembler coding accessible to someone other than the original programmer and his close friends. That's for clear, well designed code. For poorly written, unclear code, I must agree whole-heartedly with Messrs. Kernigan and Plauger: no amount of commenting can save bad code, in any language.

I don't mean this as a serious criticism of Mr. Gabrielson or his code. I like both of them. I just need to take this opportunity to sound off on a sore point.

In closing, and to make sure nobody takes this the wrong way, I wish to reiterate my overall appreciation of this progam: it is both well-written and useful, something that I see far too little of.

Sincerely,
M.J. Maney

630 Wren
Palatine, IL 60067

XS FLAG IN 8085 OP CODE

Dear *Dr. Dobb's*:

With reference to the article "Unspecified 8085 Op Codes Enhance Programming" (*DDJ* #34), the authors were not able to fully describe the function of the flag bit labeled X5 by them. After checking their logic equation on my 8085, I have divined its purpose in arithmetic operations; X5 is set if the true sign of the result of signed arithmetic is minus. One use for this test might be in sign extension.

I also ask any readers who have tested these op codes on other versions of the 8085, i.e., Intel's newer 8085A, and the second source parts NEC uPD8085 and AMD8085, to report whether these are compatible or not.

Sincerely
Darrel J. Van Buer

1522 Brockton Ave.
Los Angeles, CA 90025

8080 REGISTER/STORAGE

Dear *DDJ*:

In what is hopefully the last echo of the 8080 register storage/recovery question, I would like to submit an optimized solution to the problem. It may be noted that, although I prefer a different stacking order to that submitted by Mr. Moss (*DDJ*, March '79), there is no functional difference between the two PULLER routines.

We differ, however, on the PUSHER routines used. I am surprised that anyone would claim reentrancy for a procedure which uses an absolute memory location for even temporary storage. Suppose the processor were interruped between the SHLD and LHLD? Also, absolute addressing is a code- and time-inefficient technique on the 8080.

I propose the following, both of which are short enough (5 bytes for PUSHER, 6 bytes for PULLER) to be RST type subroutines (which is how I use them).

```
PUSHER  XTHL            ; Stack HL, HL ← Return address
        PUSH    D
        PUSH    B
        PUSH    PSW
        PCHL            ; PC ← Return address

PULLER  POP     H       ; HL ← Return address
        POP     PSW
        POP     B
        POP     D
        XTHL            ; Stack return address again
        RET
```

Total execution time for PUSHER is 67 cycles, and PULLER uses 80, including an RST-type call to each.

Sincerely,
D.C. Sessions

A TRICK ADDICTION

Dear *Dr. Dobb's*,

To lovers of the "programming trick," to which I am hopelessly addicted, I offer the following two constructions, which each save one instruction from the corresponding classical way of doing things. This code is for the 8080, but I suspect that equivalent things can be done on the 6800 and 6502. Since I've never seen them in other listings, I assume they're original.

The first macro is used to add a single byte quantity to a double byte quantity, both being stored in registers. Assume some offset in the accumulator is to be added to HL, and the overflow from L to H must be taken into account. This will do it:

```
ADD   L       ;ACC =  ACC + L
MOV   L,A     ;L = NEW VALUE
ADC   H       ;ACC = NEW L + H + CARRY
SUB   L       ;ACC = H + CARRY
MOV   H,A     ;HL = HL + ACC
```

The crux of the method is in the third instruction where the carry from the addition of L and A is added to H. This is done without first clearing the accumulator, resulting in a superficially incorrect result; the result is corrected by the fourth instruction. Note that this fourth instruction does not, in general, leave the carry flag in the same state as the classical code would, i.e., high —if an overflow occurred. If this is important, don't use the trick.

The principles illuminated in the previous example can be applied also to negate a double byte quantity, as shown here using the HL registers as an example:

```
XRA           ;ACC = 00
SUB   L       ;ACC = -(OLD L)
MOV   L,A     ;NEW L = -(OLD L)
SBB   H       ;ACC = (NEW L) - H - BORROW
SUB   L       ;ACC = - H - BORROW
MOV   H,A     ;HL NEGATED
```

The two's complement of the original HL is left in HL.

Anone wishing to further investigate such arithmetic tricks would do well to read Tom Pittman's article, "Byte Saving Programming Tricks for the 8080," in the June/July 1976 issue of *Dr. Dobb's Journal*, Volume 1, Issue 6.

Thank you,
Bruce Komusin

RFD #1 Box 60
Middle Island, N.Y. 11953

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 39 | October 1979 | Volume 4, Issue 9

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

tiny BASIC for the F8 4
Jerry D. Fox

Page List for the Apple 19
Robert D. Diaz

Just Poking Around with my PET 22
Gary Ratliff

Producing Pictures on Your Computer with a Diablo Printer 26
Jay C. Bowden and Anna K. Scharschmidt

Quick and Dirty Routines for the Sweet-16 24
Steve Wheeler

Main Software Teletype Terminal Interface 34
Donald Faulkner

tiny BASIC for the F8

BY JERRY D. FOX
3 High Ridge Road
Granby, CT 06035

Herewith is documentation and the source for a tiny BASIC Interpreter written for the F8 microcomputer. The structure of the program is copied from Palo Alto tiny BASIC written by Dr. Li-Chen Wang. It is to Dr. Wang's credit that he wrote such an elegant program that it was fairly easy to convert to F8 from 8080 assembly language.

The program contains software stack so the nesting of GOSUB's and FOR-NEXT loops can be accommodated. All I/O uses a monitor called FAIRBUG written by Fairchild. -Jerry D. Fox

TINY BASIC FOR THE F8

Numbers. All numbers are integers and must be less than 32767.

Variables. There are 26 variables noted by letters A through Z. There is also a single array @ (I). The single array @ (I) is equivalenced to A-Z in the following manner. @(−I) = A, @(−26) = Z. In addition to A-Z and @(I) being variables, they do double duty as string variables. A$-Z$, @$(I) refer to the same physical locations but permit string operations.

Functions. There are 4 functions:

ABS(X) gives the absolute value of X.

RND(X) gives a random number between 1 and X (inclusive).

PEEK(X) retrieves the byte at addr X.

LEN(X) if X=0, gives the length of the last referenced string if X = @$(I), or A$ to Z$, it gives the length of the referenced string.

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, except strings, can be used as direct commands except the following three, they can only be used as direct command 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.

LIST 120, N will print N lines (up to a maximum of 255) starting at 120.

NEW will delete all statements.

Abbreviation and Blanks. You may use blanks freely, except that numbers, command key words, and function names cannot 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	D. = DATA	C. = CALL	F. = FOR	GOS. = GOSUB
G. = GOTO	IF = IF	IN. = INPUT	L. = LIST	LEN = LEN
M. = MON	N. = NEW	N. = NEXT	P. = PRINT	PE. = PEEK
PO. = POKE	REM = REMARK	R. = RETURN	R. = RND	R. = RUN
RES. = RESTORE	READ = READ	S. = STEP	S. = STOP	TO = TO
T. = THEN	IMPLIED = LET			

Statements. A statement consists of a statement number of between 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.

- REMor 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 etiher 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 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 6 spaces, the values of C, D, and 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" is overprinted on "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$ = 'NO' THEN @$(I) = 'RIGHT'

  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 optional. It may or may not be 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.

- CALL Command: Call (X) calls the routine at address X.

- POKE Command: POKE X,Y at address X place the 1 byte value represented by Y.

- STRINGS

```
A$ = "ABC$1234XYZ"
X = LEN (0)
INPUT  @$(I)
IF A$ = @$(I)  GOTO 100
M$ = A$
```

String variables are physically the same as simple variables but logically different. In the above examples the physical locations A will contain AB, B will contain C$, and F will contain ZZ (odd length repeats the last character for compare purposes). X would be set to 11, the length of the last referenced string.

The maximum string length is 72 characters. The compare of A$ and @$(I) is not a full string length compare, only the single physical byte A and @(I) are compared. By using the LEN function FOR-NEXT and the fact that @(–1)=A one can easily compare a whole string. It is obvious that the user must be careful or strings could smash variables that could cause errors. Strings cannot be executed in the direct mode.

- READ and DATA Commands

```
100 DATA 1,0,  'YES', – 10
200 DATA (A + B) * C,  1000/33,  'HELP'
300 READ I,  B,  C$
400 READ E,  F,  @(0),  @$(I)
```

A read statement is used to assign to the listed variables values obtained from a DATA statement. The variables are read in a one-to-one correspondence. In the example: I will = 1 and @$(I) will equal HELP.

Each time a READ statement is encountered in the program, the READ resumes at the location in the DATA statements where reading previously stopped. Location of DATA statements are unimportant, but they must be in the correct sequential order.

- RESTORE Command: The RESTORE command returns the READ-DATA pointer to its original position so data may be reread.

- MON Command: A user option to return control to a monitor.

- 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 semi-colon and other commands.

Error Report. There are only three error conditions in tiny BASIC. The statement with the error is printed out with a question 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 GOTO 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 the deleted character 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, 1".

Symbol Table

```
STAT  0003   ST1   0009   ST2   000B   ST3   000D   FLIE  005C
INST  0067   NLIF  0075   ST4   007F   ST5   0087   TxCK  008F
TxC1  00A6   GETN  00A9   GET1  00E1   GET2  00B4   GET3  00C1
GET4  00CC   GET5  00D7   FNDN  00ED   FNDL  00F5   FND1  00F1
FND2  00FD   FND3  00FE   FNDT  010F   FND4  0110   PRTN  0117
PRTG  0130   PRT1  0134   PRT2  0142   PRTM  0145   XCV1  0156
XCV2  0169   XCV3  0171   XCV4  0174   XCV5  017D   XCV6  0183
PBLK  0186   QST6  018C   QST1  0195   QST2  019A   QST3  01A5
QST4  01AA   QST5  01AE   QST6  01B8   TESL  01BD   TVS   01DB
TVE   01F5   TVT   01FF   QSCY  0211   ASOY  0214   TV0   021A
TV1   0223   TV2   0225   TV3   022A   TVR   022E   TSTM  022F
TS1   0236   TS2   0241   TS3   0242   QHOW  025B   AHOW  025E
DIRT  0269   EXEC  0267   EX1   026C   EX2   027E   EX3   0285
EX4   0289   INPR  0290   INPT  029F   IP1   02A2   IP2   02B1
IP3   02D0   IPX   02FA   IPS   0304   IP4   030C   FINI  0316
FIN   0319   DFFT  031F   LET   032A   PRIT  0335   FR1   0344
PR2   034E   PR3   035B   PR4   0363   PR5   036E   PR6   0374
PR7   0389   PR8   038C   GOSB  0391   GOS1  03A2   RETN  03B4
RET1  03C1   WHAT  03D6   HOW   03DB   SORY  03DF   LIST  0400
LIS1  041A   LIS2  041E   LIS3  0421   LIS4  042C   LIS5  042D
LIS6  0436   LIS7  0445   NEW   044B   NEWT  044E   STOP  045B
IF    045E   REM   0469   IF1   0471   FOR   0474   FR1   048F
FR2   049A   FR3   049F   FR4   04A3   FR5   04B5   FR6   04BF
FR8   04D9   NEXT  04EE   NX0   04EE   NX1   04F8   NX2   050D
NX3   053D   NX4   0540   NX5   0553   NX6   055B   READ  055C
REA1  0567   REA2  0580   DATA  059B   DAT1  05A6   RESE  05AD
GOTO  05B5   GO1   05CB   RUN   05CB   RNXL  05D1   RTSL  05DD
RSML  05E7   FINH  05F0   EXPR  05FD   EXP1  0607   XP11  060F
XP12  0615   XP13  061C   XP14  0625   XP15  062E   XP16  0635
FALE  063A   TRUE  063B   XP17  063E   XP1R  0644   XP19  0660
EXP2  0669   XP21  067A   XP22  0681   XP23  0684   XP24  0693
XP25  06A9   XP26  06BD   EXP3  06BE   XP33  06C5   XP31  06C6
XP32  06EB   XP33  06F9   XP34  06FC   XP35  0729   EXP4  073A
XP40  0745   XP41  0763   XP42  0775   XP43  07B6   XP44  07D1
XP45  0796   PARN  0799   PAR1  07D0   GWHT  07H2   ALHT  07E5
ERRR  07B8   ENDR  07F3   RDAA  07FC   SETL  0802   SETH  0815
SET1  0830   SET2  0834   SET3  0841   SET4  0844   SETR  0849
CHKN  084C   CHGN  0852   CHG1  085E   CHG2  0862   SUBD  0863
NCS   0871   MTPY  0875   ADDD  0879   MULT  087B   MUL1  08B6
DIVE  08BB   DV1   08A5   DV2   08A9   DV3   08AC   COMP  08BD
COMX  08BE   COMT  08BF   COM1  08C6   SKIP  08CE   CHAR  08CF
IGNK  08D8   IGN1  08D9   EXCH  08E0   MV21  08F1   MVUP  08FA
MV1   0904   MV2   0908   MVDN  090C   MVD1  0914   BUFV  0922
BU0   092E   BU1   0938   BU2   0946   BU5   095F   BU6   0962
BU4   096E   BU7   0971   BU8   0980   SLEN  0984   SAVE  098E
SAV1  09A1   SAV2  09B1   SAV3  09BB   RESR  09BE   REFR  09C9
RES1  09CD   RES2  09D2   PUSC  09D5   PULC  09E1   PUS0  09E9
PUSR  09EA   PUL0  09F6   PULR  09F7   PUST  0A00   PULT  0A0C
POPT  0A15   USR   0A1E   POKE  0A2F   POK1  0A4A   APOT  0A4D
QUOE  0A51   AQ1   0A53   AQ2   0A5B   AQ3   0A66   AQ4   0A75
LEN   0AB2   ABS   0A92   RND   0AA0   RN1   0AC4   PEEK  0AE4
TAB   0AF5   TA1   0AFF   UMFL  0B0F   CHR   0B3B   CH1   0B10
HB00  B000   TAB1  0B1D   TAB2  0B32   TAB4  0B9C   TAB5  0BC3
TAB6  0BC9   TABB  0BD1   PROT  0BE7   TTY0  0BED   TTY1  0BF2
TTYI  0BF5   TTCR  0BF8   MON   0BFB   RANT  0BFE   TxTB  0C00
TxTE  1300   VARN  1300   STRH  1336   BUFF  1337   SKED  1381
STAK  1400
```

LISTING

```
 1  0000 29 04 4F    >          JMP   NEWT          GO SETUP TXTU
 2                   > *                           R3=FORMAT NUMBER
 3                   > *                           R4=CURRENT CHARACTER BEING PROCESSED
 4                   > *                           R5 IS A FLAG FOR STRINGS
 5                   > *                           R9 IS THE STATUS SAVE REG
 6                   > *                           SCRATCH STORAGE
 7                   > *                           26-27 TxTU       UNFILLED TEXT ADDR
 8                   > *                           30-31 CURRNT     CURRENT TBF
 9                   > *                           32-33 SKINP      SAVE INPUT STACK
10                   > *                           34-35 SKGOS      SAVE GOSUE STACK
11                   > *                           SAVE ORDER TOP-DOWN
12                   > *                           36-37 LOPVAR     LOOP VARIABLE
13                   > *                           40-41 LOPPT      TEXT POINTER
14                   > *                           42-43 LOPLN      LINE NUMBER
15                   > *                           44-45 LOPINC     INCREMENT
16                   > *                           46-47 LOPLMT     LIMIT
17                   > *                           52-53  TEMP POINTER FOR READ-DATA
18                   > *                           50-51  RESTORE POINTER FOR READ-DAT
19  0003 2A 14 00    > START    DCI   STACK          SETUP
20  0006 0E          >          LR    Q,DC          STACK REG Q
21  0007 70          >          CLR                 ZERO
22  0008 6F          >          LISL  7             SCRATCH
23  0009 63          > S11      LISU  3             AREA
24  000A 5C          >          LR    S,A           30-37
25  000B 65          >          LISU  5             AND
26  000C 5E          >          LR    D,A           50-57
27  000D 8F FB       >          BR7   ST1           30-37
28  000F 55          >          LR    5,A           CLEAR STRING FLAG FOR DIRECT
29  0010 62          >          LISU  2             RESET ISAR
30  0011 28 0B FB    >          PI    TTCR          CR LF
31  0014 58          >          LR    8,A           SET FOR PRTSTG
32  0015 2A 0B E7    >          DCI   PROMPT        OUTPUT
33  0018 28 01 30    >          PI    PRTSTG        PROMPT
34  001B 20 3E       > ST2      LI    *>*           LOAD
35  001D 51          >          LR    1,A           PROMPT CHARACTER
36  001E 28 C0 A9    >          PI    GETLN         GET A LINE
37  0021 11          >          LR    H,DC          SAVE EOL
38  0022 4B          >          LR    A,11          LOW ORDER
39  0023 56          >          LR    6,A           BYTE IN R6
40  0024 2A 13 37    >          DCI   BUFF          START OF LINE
41  0027 28 02 2F    >          PI    TSTNUM        SEE IF A NUMBER
42  002A 31          >          DS    1             NUMBER ?
43  002B 94 05       >          BNZ   ST3           B IF A NUMBER
44  002D 11          >          LR    H,DC          SAVE TBF
45  002E 29 02 64    >          JMP   DIRECT        A COMMAND
46  0031 20 FE       > ST3      LI    -2            BACKUP DC
47  0033 8E          >          ADC                 TO HEX LINE #
48  0034 28 09 D5    >          PI    PUSHDC        SAVE BOL
49  0037 4D          >          LR    A,I           STORE
50  0038 17          >          ST                  HEX
51  0039 4E          >          LR    A,D           LINE
52  003A 17          >          ST                  NUMBER
53  003B 4B          >          LR    A,11          LOW ORDER BYTE
54  003C 18          >          COM                 OF BEGIN
55  003D 1F          >          INC                 MAKE IT -
56  003E C6          >          AS    6             SAVE #
57  003F 56          >          LR    6,A           OF CHARS
58  0040 28 00 ED    >          PI    FNDLN         FIND LINE #
59  0043 28 09 D5    >          PI    PUSHDC        SAVE ADDR
60  0046 2C          >          XDC                 IN DC1 ALSO
61  0047 10          >          LR    DC,H          SAVE IN DC0 ALSO
62  0048 6E          >          LISL  6             PUT TXTU
63  0049 28 09 EA    >          PI    PUSHSR        ON THE STACK TOP
64                   > *                           AT THIS POINT DC0=DC1=FOUND LINE ADD
65  004C 84 05       >          BZ    FLINE         B IF FOUND LINE
66  004E 92 26       >          BNC   NLINE         B IF PAST TXTU
67  0050 90 16       >          BR    INSERT        B IF BETWEEN 2 LINES
68                   > *                           DELETE LINE POINTED TO BY DC1
69                   > *                           MOVE H THRU TxTU UP
70                   > *                           DC0=LINE FOLLOWING FOUND LINE
71                   > *                           DC1=FOUND LINE
72  0052 16          > FLINE    LM                  GET PAST
73  0053 16          >          LM                  LINE #
74  0054 28 01 0F    >          PI    FNDNXT        FIND NEXT LINE *FROM* IN DC0
75  0057 28 08 FA    >          PI    MVUP          DELETE THE LINE
76                   > *                           DC1 HAS THE UPDATED TXTU ADDR
77  005A 2C          >          XDC                 HAS UPDATED TxTU
78  005B 11          >          LR    H,DC          INTO H
79  005C 6E          >          LISL  6             SET ISAR TO TxTU
80  005D 4A          >          LR    A,10          NEW
81  005E 5D          >          LR    I,A           TXTU
82  005F 4B          >          LR    A,11          ADDR
83  0060 5E          >          LR    D,A           IN TXTU
84  0061 28 0A 15    >          PI    POPRT         CLEAR OLD TXTU
85  0064 28 09 EA    >          PI    PUSHSR        NEW TXTU ON THE STACK TOP
86                   > *                           INSERT BETWEEN 2 LINES
87  0067 46          > INSERT   LR    A,6           LOAD LINE LENGTH
88  0068 25 03       >          CI    3             ANY TEXT ?
89  006A 84 98       >          BZ    START         NO JUST DELETE
90                   > *                           MOVE TXTU(DC0) THRU FOUND LINE(STACK
91                   > *                           TOP) TO TXTU+R4 (DC1)
92  006C 28 00 89    >          PI    TXCK          UPDATE TXTU
93  006F 2C          >          XDC                 SETUP THE MOVE
94  0070 28 09 0C    >          PI    MVDOWN        MOVE DOWN
95  0073 90 04       >          BR    ST4           MOVE IN NEW LINE
96                   > *                           NEW LINE
97  0075 28 00 89    > NLINE    PI    TXCK          UPDATE TXTU
98                   > *                           MOVE IN NEW LINE
99                   > *                           SETUP DC0 AND DC1 FOR NEW LINE MOVE
100 0078 28 09 E1    > ST4      PI    FULLDC        *TO* FOUND LINE
101 007B 2C          >          XDC                 IN DC1
102 007C 28 09 E1    >          PI    FULLDC        *FROM* NEW LINE
103 007F 16          > ST5      LM                  LOAD A BYTE
104 0080 2C          >          XDC                 SWITCH DC
105 0081 17          >          ST                  STORE IT
106 0082 2C          >          XDC                 RESET DC
107 0083 36          >          DS    6             DEC BYTE COUNT
108 0084 94 FA       >          BNZ   ST5           B IF MORE
109 0086 29 00 1B    >          JMP   ST2           NEXT RECORD
110                  > *                           THIS ROUTINE EXITS WITH
111                  > *                           DC0=NEW TXTU, DC1=OLD TXTU
112                  > *                           SEE IF TXT AREA IS LEFT
113                  > *                           IF ROOM UPDATE TXTU BY R4
114 0089 08          > TXCK     LR    K,P           SAVE RETURN
115 008A 28 09 E1    >          PI    PULLDC        GET TXTU
116 008D 2C          >          XDC                 SAVE TXTU IN DC1
117 008E 2A 13 00    >          DCI   TXTE          TEXT END ADDR
118 0091 28 09 D5    >          PI    PUSHDC        ON THE STACK
119 0094 2C          >          XDC                 PUT
120 0095 11          >          LR    H,DC          TXTU
121 0096 2C          >          XDC                 IN
122 0097 10          >          LR    DC,H          BOTH DC0 AND DC1
123 0098 46          >          LR    A,6           UPDATE TXTU
124 0099 8E          >          ADC                 WITH NEW LINE LENGTH
125 009A 11          >          LR    H,DC          NEW TXTU IN 10-11
126 009B 6E          >          LISL  6             TXTU ISAR ADDR
127 009C 4A          >          LR    A,10          STORE
128 009D 5D          >          LR    I,A           NEW
129 009E 4B          >          LR    A,11          TXTU
130 009F 5E          >          LR    D,A           IN 26-27
131 00A0 28 08 BF    >          PI    COMT          COMPARE TXTE-TXTU
132 00A3 82 04       >          BC    TXC1          B IF MORE ROOM
133 00A5 29 02 14    >          JMP   ASORRY        NO MORE ROOM
134 00A8 0C          > TXC1     PK                  RETURN
135                  > *
136                  > *                           OUTPUT CONTENTS OF ACCUM
137                  > *                           INPUT A LINE
138 00A9 08          > GETLN    LR    K,P           SAVE RETURN
139 00AA 28 0A 00    >          PI    PUSHRT        PUSH IT
140 00AD 20 48       >          LI    72            BUFFER
141 00AF 58          >          LR    8,A           LENGTH
142 00B0 41          >          LR    A,1           LOAD LEAD CHARACTER
143 00B1 2A 13 37    > GET1     DCI   BUFF          BUFFER ADDR
144 00B4 51          > GET2     LR    1,A           OUTPUT
145 00B5 28 0B ED    >          PI    TTYO          THE BYTE
146 00B8 28 0B F5    >          PI    TTYI          GET A CHARACTER
147 00BB 21 7F       >          NI    $7F           TURN PARITY OFF
148 00BD 25 7F       >          CI    $7F           BACKSPACE ? RUBOUT
149 00BF 94 0C       >          BNZ   GET4          B IF NOT
150                  > *                           BACKUP DC0 TO DELETE A CHAR
151 00C1 20 FF       > GET3     LI    -1            BACKUP
152 00C3 8E          >          ADC                 DC0
153 00C4 48          >          LR    A,8           ADJUST
154 00C5 1F          >          INC                 THE
155 00C6 58          >          LR    8,A           COUNTER
156 00C7 11          >          LR    H,DC          ECHO
157 00C8 16          >          LM                  THE LAST
158 00C9 10          >          LR    DC,H          CHARACTER
159 00CA 90 E9       >          BR    GET2          DONT STORE
160 00CC 25 7D       > GET4     CI    $7D           DELETE LINE ?
161 00CE 94 08       >          BNZ   GET5          B IF NOT ALT-MODE
162 00D0 28 0B F8    >          PI    TTCR          OUTPUT CR LF
163 00D3 20 5E       >          LI    $5E           AND UP ARROW
```

```
164  00D5 90 DB      >            BR    GET1      AND START OVER
165  00D7 25 0A      > GET5       CI    $0A       LR ?
166  00D9 84 DA      >            BZ    GET2      IGNORE IT
167  00DB 25 00      >            CI    0         NULL ?
168  00DD 84 D6      >            BZ    GET2      IGNORE IT
169  00DF 17         >            ST              STORE IN BUFF
170  00E0 38         >            DS    R         CHECK BUFF ROOM
171  00E1 84 DF      >            BZ    GET3      B IF NO MORE ROMM
172  00E3 25 0D      >            CI    $0D       CR ?
173  00E5 94 CE      >            BNZ   GET2      B IF NOT CR
174  00E7 28 0B FB   >            PI    TTCR      OUTPUT CR LF
175  00EA 29 0A 0C   >            JMP   PULLRT    RETURN
176                  > *
177                  > *                           IF TXTU=PRESENT DC THEN STOP LOOKING
178                  > *                           THIS ROUTINE LOOKS FOR A LINE NUMBER
179                  > *                           THE LINE NUMBER IS IN SCRATCH 20-21
180                  > *                           CC=0 FOUND LINE,CC=+ PAST A NUMPER,
181                  > *                           CC=- PAST END OF TEXT
182                  > *                           FOR A FNDL ENTRY DC POINTS TO LINE NUMBER
183  00ED 2A 0C 00   > FNDLN      DCI   TXTB      LOAD BEGIN OF TEXT ADDR
184  00F0 08         > FNDL       LR    K,P       SAVE RETURN
185  00F1 6E         > FND1       LISL  6         SET ISAR TO TXTU
186  00F2 28 09 D5   >            PI    PUSHDC    PUT PRESENT TBP ON THE STACK
187  00F5 28 0B BF   >            PI    COMT      ADDR-TXTU SKIP LR H,DC AND LISL 0
188  00F8 92 05      >            BNC   FND2      B IF NOT THE END
189  00FA 70         >            CLR             SET
190  00FB 59         >            LR    9,A       STATUS
191  00FC 1D         >            LR    W,J       TO NO CARRY
192  00FD 0C         > FNDR       PK              RETURN
193  00FE 6C         > FND2       LISL  4         PUT THE
194  00FF 16         >            LM              TEXT
195  0100 5D         >            LR    I,A       NUMBER
196  0101 16         >            LM              ON
197  0102 5E         >            LR    D,A       THE
198  0103 28 09 EB   >            PI    PUSHSR+1  STACK
199  0106 28 0B BE   >            PI    COMX      TEXT-INPUT
200  0109 82 F3      >            BC    FNDR      B IF PAST OR EQUAL
201  010B 16         >            LM              GET PAST
202  010C 16         >            LM              LINE NUMBER
203  010D 90 02      >            BR    FND4      LOOK FOR NEXT LINE
204                  > *                           DC MUST BE SET PAST LINE # FOR THIS ENTR
205  010F 08         > FNDNXT     LR    K,P       SAVE RETURN
206  0110 16         > FND4       LM              LOAD NEXT CHAR
207  0111 25 0D      >            CI    $0D       CR ?
208  0113 94 FC      >            BNZ   FND4      B IF NOT
209  0115 90 DB      >            BR    FND1      KEEP LOOKING
210                  > *
211                  > *                           DC POINTS TO LINE TO PRINT
212  0117 08         > PRTLN      LR    K,P       SAVE
213  0118 28 0A 00   >            PI    PUSHRT    RETURN
214  011B 68         >            LISL  0         SET ISAR
215  011C 16         >            LM              LOAD
216  011D 5D         >            LR    I,A       THE
217  011E 16         >            LM              NUMBER
218  011F 5D         >            LR    I,A       INTO 20-21
219  0120 74         >            LIS   4         SET NUMBER
220  0121 53         >            LR    3,A       OF CHARS TO PRINT
221  0122 28 01 45   >            PI    PRTNUM    CONVERT AND PRINT
222  0125 28 01 86   >            PI    PBLK      PRINT A BLANK
223  0128 70         >            CLR             SET END CHAR
224  0129 58         >            LR    8,A       TO ZERO
225  012A 28 01 30   >            PI    PRTSTG    PRINT A STRING
226  012D 29 0A 0C   >            JMP   PULLRT    RETURN
227                  > *
228                  > *                           PRINT UNTIL MATCH OF R7
229                  > *                           OR A CR IS FOUND
230  0130 08         > PRTSTG     LR    K,P       SAVE
231  0131 28 0A 00   >            PI    PUSHRT    RETURN
232  0134 16         > PRT1       LM              LOAD A CHAR
233  0135 54         >            LR    4,A       SAVE CHAR
234  0136 51         >            LR    1,A       OUTPUT REG
235  0137 E8         >            XS    8         SEE IF A MATCH
236  0138 84 09      >            BZ    PRT2      RETURN IF A MATCH
237  013A 28 0B ED   >            PI    TTYO      OUTPUT A CHAR
238  013D 44         >            LR    A,4       DID WE OUTPUT
239  013E 25 0D      >            CI    $0D       A CR ?
240  0140 94 F3      >            BNZ   PRT1      B IF NOT
241  0142 29 0A 0C   > PRT2       JMP   PULLRT    RETURN
242                  > *
243                  > *                           PRINT NUMBER IN R20-21
244  0145 08         > PRTNUM     LR    K,P       SAVE RETURN
245  0146 28 0A 00   >            PI    PUSHRT    ADDR ON THE STACK
246                  > *                           ROUTINE TO CONVERT HEX TO DECIMAL
247                  > *                           HEX NUMBER MUST BE IN SCRATCH 20-21
248                  > *                           CHANGES SCRATCH 22-23 AND 24-25
249  0149 28 0B 4C   >            PI    CHKSGN    CHECK SIGN
250  014C 6A         >            LISL  2         SET SCRATCH
251  014D 70         >            CLR             22-23
252  014E 5D         >            LR    I,A       TO
253  014F 7A         >            LIS   10        DECIMAL
254  0150 5E         >            LR    D,A       10
255  0151 28 09 EA   >            PI    PUSHSR    PUT A 10 ON THE STACK
256  0154 43         >            LR    A,3       SAVE
257  0155 54         >            LR    4,A       FORMAT #
258  0156 28 0B BC   > XCV1       PI    DIVIDE    DIVIDE BY 10
259  0159 28 09 E5   >            PI    PUSH20    SAVE DIGIT(REMAINDER)
260  015C 6C         >            LISL  4         MOVE THE
261  015D 28 09 EA   >            PI    PUSHSR    RESULT
262  0160 28 09 F6   >            PI    PULL20    TO 20-21
263  0163 34         >            DS    4         DEC DIGIT COUNTER
264  0164 4D         >            LR    A,I       SEE IF
265  0165 EC         >            XS    S         EXCLUSIVE OR AND
266  0166 CE         >            AS    D         ADD TO CHECK FOR ZERO
267  0167 94 EE      >            BNZ   XCV1      B IF MORE
268  0169 34         > XCV2       DS    4         NEED TO PAD BLANKS ?
269  016A 91 06      >            BM    XCV3      B IF WE DONT
270  016C 28 01 86   >            PI    PBLK      PRINT A BLANK
271  016F 90 F9      >            BR    XCV2      SEE IF WE NEED MORE BLANKS
272  0171 48         > XCV3       LR    A,8       OUTPUT
273  0172 90 0A      >            BR    XCV5      THE SIGN
274  0174 28 09 F7   > XCV4       PI    PULLSR    GET A DIGIT
275  0177 25 0A      >            CI    10        LAST ONE ?
276  0179 84 09      >            BZ    XCV6      B IF LAST
277  017B 22 30      >            OI    $30       ASCII
278  017D 51         > XCV5       LR    1,A       OUTPUT REG
279  017E 28 0B ED   >            PI    TTYO      OUTPUT
280  0181 90 F2      >            BR    XCV4      ANOTHER
281  0183 29 0A 0C   > XCV6       JMP   PULLRT    RETURN
282                  > *
283                  > *                           OUTPUT A BLANK
284  0186 20 20      > PBLK       LI    " "       LOAD A BLANK
285  0188 51         >            LR    1,A       OUTPUT REG
286  0189 29 0B ED   >            JMP   TTYO      GO PRINT
287                  > *
288                  > *                           CHECK FOR ' OR " TYPE STRING
289                  > *                           R8 HAS THE PRESENT BYTE
290                  > *                           CC=0 FOR A DROP THRU RETURN
291  018C 08         > QSTRING    LR    K,P       SAVE
292  018D 28 0A 00   >            PI    PUSHRT    RETURN
293  0190 44         >            LR    A,4       GET CHAR
294  0191 25 27      >            CI    "'"       STRING ?
295  0193 94 16      >            BNZ   QST4      NOT ' MAYBE "
296  0195 58         > QST1       LR    8,A       LOAD END CHAR
297  0196 16         >            LM              GET PAST IT
298  0197 28 01 30   >            PI    PRTSTG    PRINT STRING
299  019A 28 0A 15   > QST2       PI    POPRT     PULL RETURN INTO 12-13
300  019D 44         >            LR    A,4       WAS THE LAST A CR
301  019E 25 0D      >            CI    $0D       CR ?
302  01A0 94 04      >            BNZ   QST3      B IF NOT CR
303  01A2 29 05 D1   >            JMP   RNXL      RUN NEXT LINE
304  01A5 28 08 CF   > QST3       PI    CHAR      GET NEXT CHAR
305  01A8 E4         >            XS    4         SET CC=0 FOR DROP THRU RETURN
306  01A9 0C         >            PK              RETURN
307  01AA 25 22      > QST4       CI    '"'       STRING ?
308  01AC 84 E8      >            BZ    QST1      GO LOAD END CHAR
309  01AE 25 5F      > QST5       CI    $5F       A BACK ARROW
310  01B0 94 09      >            BNZ   QST6      B IF NOT
311  01B2 7D         >            LIS   13        OUTPUT JUST A CR
312  01B3 51         >            LR    1,A       OUTPUT REG
313  01B4 28 0B F2   >            PI    TTY1      OUTPUT THE CR
314  01B7 16         >            LM              GET PAST <
315  01B8 90 E1      >            BR    QST2      GO DROP THRU RETURN
316  01BA 29 0A 0C   > QST6       JMP   PULLRT    RETURN AND BRANCH
317                  > *
318                  > *                           SEE IF VARIABLE OR ARRAY
319                  > *                           IF SO PUT ADDR IN DC1 ,TBP IN DC0
320  01BD 44         > TESTVL     LR    A,4       LOAD CHARACTER
321  01BE 24 C0      >            AI    $C0       SUBTRACT 6
322  01C0 91 6D      >            BM    TVR       B IF NOT A VARIABLE
323  01C2 1E         >            LR    J,W       SAVE STATUS
324  01C3 52         >            LR    2,A       SAVE VARIABLE
325  01C4 25 1A      >            CI    26        A-Z ?
326  01C6 91 67      >            BM    TVR       B IF NOT
327  01C8 08         >            LR    K,P       SAVE
328  01C9 28 0A 00   >            PI    PUSHRT    RETURN
329  01CC 28 08 CE   >            PI    SKIP      INC DC AND GET NEXT CHAR
330  01CF 25 24      >            CI    "$"       A STRING ?
331  01D1 94 06      >            BNZ   TVS       b NOT A STRING
332  01D3 49         >            LR    A,9       SAVE STATUS
333  01D4 55         >            LR    5,A       SET AS STRING
334  01D5 28 08 CE   >            PI    SKIP      GET NEXT CHARACTER
335  01D8 1D         > TVS        LR    W,J       RESTORE STATUS
336  01D9 94 49      >            BNZ   TV1       B IF A VARIABLE
337  01DB 28 07 99   >            PI    PARN      ( SHOULD BE NEXT
```

```
0338  01DE 28 0B F1    >            PI    MV2021      MOVE 20-21 TO 22-23
0339  01E1 28 0B 79    >            PI    ADDD        DOUBLE INDEX
0340  01E4 81 13       >            BP    TVT         B IF NOT TOO BIG
0341                   > *                           MAY BE @(-INDEX)
0342  01E6 45          >            LR    A,5         CHANGE STATUS
0343  01E7 18          >            COM               OF THE
0344  01E8 1F          >            INC               PREVIOUS
0345  01E9 55          >            LR    5,A         STRING JUST IN CASE
0346  01EA 4D          >            LR    A,I         SEE IF
0347  01EB 1F          >            INC               LT -255
0348  01EC 94 0B       >            BNZ   TVE         B IF NOT
0349  01EE 4E          >            LR    A,D         MAKE
0350  01EF 18          >            COM               LOW
0351  01F0 1F          >            INC               POSITIVE
0352  01F1 25 34       >            CI    52          @(-INDEX) PAST Z ?
0353  01F3 81 31       >            BP    TV2         B IF IT ISNT
0354  01F5 29 02 5B    > TVE        JMP   GHOW        ERROR
0355  01F8 2C          > TVT        XDC               SAVE TBP
0356  01F9 2A 13 C0    >            DCI   VAREGN      BEGIN OF ARRAY
0357  01FC 11          >            LR    H,DC        INTO 10-11
0358  01FD 2B 0B F1    >            PI    MV2021      MOVE 2*INDEX INTO R22-23
0359  0200 4A          >            LR    A,10        SO WE
0360  0201 5D          >            LR    I,A         CAN
0361  0202 4B          >            LR    A,11        PUT BEGIN
0362  0203 5E          >            LR    D,A         IN 20-21
0363  0204 2B 0B 63    >            PI    SUBD        VARBGN-INDEX
0364  0207 6E          >            LISL  6           PUT TXTU
0365  0208 2B 09 EA    >            PI    PUSHSR      ON THE STACK
0366  020B 2B 0B BD    >            PI    COMP        TXTU-@(INDEX)
0367  020E 92 0B       >            BNC   TVO         B IF ROOM LEFT
0368  0210 2C          >            XDC               GET TBP
0369  0211 2B 09 D5    > GSORRY     PI    PUSHDC      SAVE TBP
0370  0214 2A 03 DF    >.ASORRY     DCI   SORRY
0371  0217 29 07 B8    >            JMP   ERROR       PROCESS ERROR
0372  021A 12          > TVO        SR    1           MAKE STATUS + OR 0
0373  021B 2B 09 EA    >            PI    PUSHSR      MOVE VAR ADDR
0374  021E 2B 09 E1    >            PI    PULLDC      TO DC
0375  0221 90 08       >            BR    TV3         GO RETURN
0376                   > *                           A-Z VARIABLE
0377  0223 42          > TV1        LR    A,2         LOAD VARIABLE
0378  0224 13          >            SL    1           VAR INDEX*2
0379  0225 2C          > TV2        XDC               SAVE TBP
0380  0226 2A 13 00    >            DCI   VARBGN      GET
0381  0229 8E          >            ADC               VARIABLE ADDR
0382  022A 2C          > TV3        XDC               DC0=TBP, DC1=VAR ADDR
0383  022B 29 0A 0C    >            JMP   PULLRT      RETURN
0384  022E 1C          > TVR        POP               FAST RETURN
0385                   > *
0386                   > *                           TEST AN ITEM POINTED TO BY DC
0387                   > *                           THE NUMBER AND @20-21) WILL CONTAIN
0388                   > *                           THE HEX CONVERSION OF IT
0389                   > *                           IF R1=0 NOT A NUMBER
0390  022F 08          > TSTNUM     LR    K,P         SAVE RETURN
0391  0230 68          >            LISL  0           SET ISAR
0392  0231 70          >            CLR               ZERO
0393  0232 5D          >            LR    I,A         LEAD
0394  0233 5E          >            LR    D,A         BYTES
0395  0234 1F          >            INC               SET R1=1 FOR NO NUMBER
0396  0235 51          >            LR    1,A         AND DIGIT COUNTER
0397  0236 28 0B CF    > TS1        PI    CHAR        GET NEXT CHAR
0398  0239 25 2F       >            CI    $2F         SEE
0399  023B 81 05       >            BP    TS2         IF
0400  023D 25 39       >            CI    $39         A
0401  023F 81 02       >            BP    TS3         DECIMAL
0402  0241 0C          > TS2        PK                RETURN
0403  0242 15          > TS3        SL    4           STRIP
0404  0243 14          >            SR    4           ASCII
0405  0244 52          >            LR    2,A         SAVE THE DIGIT
0406  0245 31          >            DS    1           SET R1 FOR A NUMBER FOUND
0407  0246 28 0B F1    >            PI    MV2021      MOVE 20-21 TO 22-23
0408  0249 70          >            CLR               ZERO R20
0409  024A 5D          >            LR    I,A         AND
0410  024B 5C          >            LR    S,A         R21
0411  024C 7A          >            LIS   10          MULTIPLY
0412  024D 57          >            LR    7,A         EXISTING DIGITS
0413  024E 28 0B 7B    >            PI    MULT        BY 10
0414  0251 69          >            LISL  1           NOW
0415  0252 4C          >            LR    A,S         ADD
0416  0253 C2          >            AS    2           THE
0417  0254 5E          >            LR    D,A         NEW
0418  0255 4C          >            LR    A,S         DIGIT
0419  0256 19          >            LNK               TO THE
0420  0257 5C          >            LR    S,A         ACCUMULATED RESULT
0421  0258 16          >            LM                SKIP THIS BYTE
0422  0259 81 DC       >            BP    TS1         B IF NO OVERFLOW
0423  025B 28 09 D5    > GHOW       PI    PUSHDC      SAVE TBP
0424  025E 2A 03 DB    > AHOW       DCI   HOW
0425  0261 29 07 B8    >            JMP   ERROR

0426                   > *                           DC1 MUST POINT TO TABLE
0427                   > *                           DC0 POINTS TO WORD
0428                   > *                           REG H POINTS TO DC0
0429  0264 2A 0B 1D    > DIRECT     DCI   TAB1        COMMAND TABLE
0430  0267 2C          > EXEC       XDC               IN DC1
0431  0268 10          >            LR    DC,H        DC0=TINY BASIC POINTER(TBP)
0432  0269 20 FF       >            LI    -1          SET R1
0433  026B 51          >            LR    1,A         TO -1
0434  026C 16          > EX1        LM                LOAD FROM TB LINE
0435  026D 2C          >            XDC               GET TABLE ADDR
0436  026E 25 2E       >            CI    '.'         PERIOD ?
0437  0270 84 14       >            BZ    EX3         B IF YES
0438  0272 8D          >            CM                COMPARE
0439  0273 2C          >            XDC               PUT TBP BACK IN DC0
0440  0274 84 F7       >            BZ    EX1         B IF A MATCH
0441                   > *                           HERE NO MATCH
0442  0276 41          >            LR    A,1         BACKUP
0443  0277 8E          >            ADC               TBP
0444  0278 2C          >            XDC               AND GET THE
0445  0279 8E          >            ADC               LAST ACCESSED
0446  027A 16          >            LM                TABLE BYTE
0447  027B F1          >            NS    1           AN ADDR?
0448  027C 91 0C       >            BM    EX4         B IF IT WAS
0449                   > *                           LOOK FOR ADDR
0450  027E 16          > EX2        LM                LOAD
0451  027F F1          >            NS    1           AN ADDR?
0452  0280 81 FD       >            BP    EX2         B IF NOT
0453  0282 16          >            LM                GET LOW BYTE
0454  0283 90 E3       >            BR    EXEC        RESTORE WORD ADDR
0455                   > *                           FOUND A PERIOD
0456  0285 16          > EX3        LM                LOAD NEXT CHAR
0457  0286 F1          >            NS    1           AN ADDR?
0458  0287 81 FD       >            BP    EX3         B IF NOT
0459  0289 13          > EX4        SL    1           TURN SIGN
0460  028A 12          >            SR    1           BIT OFF
0461  028B 04          >            LR    KU,A        SAVE HI ORDER
0462  028C 16          >            LM                LOAD
0463  028D 05          >            LR    KL,A        LOW ORDER
0464  028E 2C          >            XDC               GET TP POINTER IN DC1
0465  028F 0C          >            PK                CALL ROUTINE
0466                   > *
0467                   > *                           SKINP IN 32-33
0468  0290 6A          > INPERR     LISL  2           SETUP SKINP ISAR
0469  0291 4D          >            LR    A,I         RESTORE
0470  0292 06          >            LR    QU,A        THE
0471  0293 4E          >            LR    A,D         OLD
0472  0294 07          >            LR    QL,A        STACK
0473  0295 2B 09 F6    >            PI    PULL2D      RESTORE CURRNT
0474  0298 62          >            LISU  2           RESET ISAR
0475  0299 2B 09 E1    >            PI    PULLDC      CLEAR STACK
0476  029C 2B 09 E1    >            PI    PULLDC      GET ORIGINAL TBP
0477                   > *
0478  029F 2B 09 D5    > INPUT      PI    PUSHDC      SAVE TBP IN CASE OF ERROR
0479  02A2 2B 0B CF    > IP1        PI    CHAR        GET NEXT CHAR
0480  02A5 2B 01 BC    >            PI    QSTRING     SEE IF A STRING
0481  02A8 94 0B       >            BNZ   IP2         B NOT A STRING
0482  02AA 2B 01 BD    >            PI    TESTVL      VARIABLE ?
0483  02AD 91 5E       >            BM    IP4         B IF NOT
0484  02AF 90 20       >            BR    IP3         B IF A VAR
0485                   > *
0486  02B1 2B 09 D5    > IP2        PI    PUSHDC      SAVE TBP FOR PRTSTG
0487  02B4 2B 01 BD    >            PI    TESTVL      VARIABLE ?
0488  02B7 81 04       >            BP    **5         B IF A VAR
0489  02B9 29 07 B2    >            JMP   QWHAT       ERROR
0490  02BC 11          >            LR    H,DC        SAVE TBP
0491  02BD 16          >            LM                SAVE
0492  02BE 57          >            LR    7,A         THIS BYTE
0493  02BF 70          >            CLR               AND
0494  02C0 10          >            LR    DC,H        STORE
0495  02C1 17          >            ST                A ZERO
0496  02C2 58          >            LR    B,A         FOR END OF STRING
0497  02C3 2B 09 E1    >            PI    PULLDC      GET TBP
0498  02C6 2B 01 30    >            PI    PRTSTG      PRINT STRING
0499  02C9 20 FF       >            LI    -1          BACKUP
0500  02CB 8F          >            ADC               DC
0501  02CC 11          >            LR    H,DC        SAVE TBP
0502  02CD 47          >            LR    A,7         AND RESTORE
0503  02CE 17          >            ST                CHAR
0504  02CF 10          >            LR    DC,H        RESTORE TBP
0505                   > *                           HERE AN INPUT VARIABLE
0506  02D0 2B 09 D5    > IP3        PI    PUSHDC      SAVE TBP
0507  02D3 63          >            LISU  3           SAVE
0508  02D4 2B 09 E9    >            PI    PUSH2D      CURRNT
0509  02D7 20 FF       >            LI    -1          SET CURRNT
0510  02D9 5C          >            LR    S,A         TO MINUS
0511  02DA 6A          >            LISL  2           ISAR FOR SKINP
0512  02DB 02          >            LR    A,QU        SAVE
0513  02DC 5D          >            LR    I,A         THE
```

```
514  02DD 03          >          LR    A,GL          STACK
515  02DE 5D          >          LR    I,A           POINTER
516  02DF 62          >          LISU  2             RESET ISAR
517  02E0 2C          >          XDC                 SAVE
518  02E1 28 09 D5    >          PI    PUSHDC        VARIABLE ADDR
519                   > *                           PROMPT FOR INPUT
520  02E4 20 3A       >          LI    ':'           PROMPT
521  02E6 51          >          LR    1,A           CHAR
522  02E7 28 00 A9    >          PI    GETLN         GET A LINE
523  02EA 2A 13 37    >          DCI   BUFF          INPUT ADDR
524  02ED 45          >          LR    A,5           SETUP
525  02EE 59          >          LR    9,A           STRING
526  02EF 1D          >          LR    W,J           STATUS
527  02F0 91 09       >          BM    IPX           B IF NOT A STRING
528  02F2 28 09 E1    >          PI    PULLDC        GET VAR ADDR
529  02F5 28 09 22    >          PI    BUFTOV        MOVE BUFF TO VAR
530  02F8 90 0B       >          BR    IPS           CONTINUE
531  02FA 28 05 FD    > IPX      PI    EXPR          EVALUATE EXPR
532  02FD 28 09 E1    >          PI    PULLDC        GET VAR ADDR
533  0300 4D          >          LR    A,I           STORE THE VALUE
534  0301 17          >          ST                  INTO
535  0302 4F          >          LR    A,D           THE
536  0303 17          >          ST                  VARIABLE
537  0304 63          > IPS      LISU  3             RESTORE
538  0305 28 09 F7    >          PI    PULLSR        CURRNT
539  0308 62          >          LISU  2             RESTORE
540  0309 28 09 E1    >          PI    PULLDC        TBP
541  030C 28 0A 15    > IP4      PI    POPRT         CLEAR STACK
542  030F 28 0B D8    >          PI    IGNEK         GET NEXT CHAR
543  0312 25 2C       >          CI    ',',          COMMA ?
544  0314 94 8A       >          BZ    INPUT         B IF MORE
545                   > *                           SEE IF PROPER END
546  0316 20 FF       > FINI     LI    -1            BACKUP
547  0318 8E          >          ADC                 TBP
548  0319 28 05 F0    > FIN      PI    FINISH        FINISH
549  031C 29 07 B2    >          JMP   GWHAT         RETURN HERE IS AN ERROR
550                   > *
551                   > *                           HERE IF NO LET
552  031F 28 0B D8    > DEFLT    PI    IGNEK         GET NEXT CHAR
553  0322 25 0D       >          CI    $0D           AN EMPTY LINE
554  0324 94 04       >          BNZ   *+5           B NOT EMPTY
555  0326 29 05 D1    >          JMP   RNXL          ITS OK GET NEXT LINE
556  0329 10          >          LR    DC,H          RESTORE TBP
557                   > *
558  032A 28 0B C2    > LET      PI    SETVAL        GET VARIABLE
559  032D 44          >          LR    A,4           RESTORE CHAR
560  032E 25 2C       >          CI    ',',          COMMA ?
561  0330 16          >          LM                  GET PAST THIS CHAR
562  0331 84 F8       >          BZ    LET           DO IT AGAIN
563  0333 90 E2       >          BR    FINI          FINISH UP
564                   > *
565                   > *                           PRINT A STRING OR NUMBER
566                   > *                           R3 IS THE FORMAT NUMBER
567  0335 76          > PRINT    LIS   6             SET
568  0336 53          >          LR    3,A           DIGIT COUNTER
569  0337 28 0B D8    >          PI    IGNBK         GET NEXT CHAR
570  033A 25 3B       >          CI    ';'           MULTIPLE RECCR ?
571  033C 94 07       >          BNZ   PR1           B IF NOT
572  033E 28 0B FF    >          PI    TTCR          JUST A CR LF
573  0341 29 05 E7    >          JMP   RSML          RUN SAME LINE
574  0344 25 0D       > PR1      CI    $0D           CR ?
575  0346 94 07       >          BNZ   PR2           B NOT CR
576  0348 28 0B FF    >          PI    TTCR          JUST CR LF
577  034B 29 05 D1    >          JMP   RNXL          RUN NEXT LINE
578  034E 25 23       > PR2      CI    '#'           FORMAT CHANGE ?
579  0350 94 0A       >          BNZ   PR3           B IF NOT
580  0352 28 05 FD    >          PI    EXPR          EVALUATE FORMAT
581  0355 69          >          LISL  1             GET FORMAT OUT
582  0356 4D          >          LR    A,I           OF 20-21
583  0357 53          >          LR    3,A           INTO R3
584  0358 33          >          DS    3             ADJUST FORMAT
585  0359 90 09       >          BR    PR4           CHECK FOR COMMA
586  035B 20 FF       > PR3      LI    -1            BACKUP
587  035D 8E          >          ADC                 TBP
588  035E 28 01 FC    >          PI    GSTRING       SEE IF A STRING
589  0361 94 12       >          BNZ   PR6           B IF NOT A STRING
590                   > *                           DROPS THRU IF STRING OR BACK___
591  0363 44          > PR4      LR    A,4           LOAD CHAR
592  0364 25 2C       >          CI    ',',          COMMA ?
593  0366 94 07       >          BNZ   PR5           B IF NOT COMMA
594  0368 16          >          LM                  GET PAST COMMA
595  0369 28 05 F0    >          PI    FINISH        GO FINISH UP LINE
596  036C 90 E1       >          BR    PR2           CONTINUE
597  036E 28 0B F8    > PR5      PI    TTCR          LIST END SO CR LF
598  0371 29 03 19    >          JMP   FIN           FINISH UP
599  0374 28 05 FD    > PR6      PI    EXPR          EVALUATE EXPRESSION
600                   > *                           SEE IF A STRING
601  0377 45          >          LR    A,5           GET STRING FLAG
602  0378 59          >          LR    9,A           INTO
603  0379 1D          >          LR    W,J           STATUS REG
604  037A 91 0E       >          BM    PR7           B IF NOT A STRING
605  037C 35          >          DS    5             CLEAR STRING FLAG
606  037D 2C          >          XDC                 SAVE TBP
607  037E 2A 13 37    >          DCI   BUFF          ADDR OF STRING
608  0381 70          >          CLR                 GET STRING
609  0382 58          >          LR    8,A           TERMINATOR
610  0383 28 01 30    >          PI    PRTSTG        GO PRINT STRING
611  0386 2C          >          XDC                 RESTORE TBP
612  0387 90 04       >          BR    PR8           CONTINUE
613  0389 28 01 45    > PR7      PI    PRTNUM        PRINT THE NUMBER
614  038C 28 0B CF    > PR8      PI    CHAR          GET NEXT CHAR
615  038F 90 D3       >          BR    PR4           LOOK FOR COMMA
616                   > *
617  0391 28 09 BE    > GOSUB    PI    SAVE          SAVE FOR PARAMETERS
618  0394 28 05 FD    >          PI    EXPR          EVALUATE EXPR
619  0397 28 09 D5    >          PI    PUSHDC        SAVE TBP
620  039A 28 00 ED    >          PI    FNDLN         FIND TARGET LINE
621  039D 84 04       >          BZ    GOS1          B IF FOUND
622  039F 29 02 5E    >          JMP   AHOW          ERROR
623  03A2 63          > GOS1     LISU  3             SET
624  03A3 28 09 E9    >          PI    PUSH20        SAVE CURRNT
625  03A6 6C          >          LISL  4             SAVE
626  03A7 28 09 EA    >          PI    PUSHSR        SKGOS
627  03AA 02          >          LR    A,QU          PUT STACK
628  03AB 5D          >          LR    I,A           POINTER
629  03AC 03          >          LR    A,QL          INTO
630  03AD 5D          >          LR    I,A           SKGOS
631  03AE 70          >          CLR                 ZERO
632  03AF 5D          >          LR    I,A           LOPVAR
633  03B0 5D          >          LR    I,A           IN SCRATCH
634  03B1 29 05 DD    >          JMP   RTSL          RUN THE LINE
635                   > *
636  03B4 28 07 F3    > RETURN   PI    ENDCR         LOOK FOR CR
637  03B7 63          >          LISU  3             SET ISAR
638  03B8 6C          >          LISL  4             TO SKGOS
639  03B9 4D          >          LR    A,I           LOAD HI
640  03BA EC          >          XS    S             EXCLUSIVE OR AND
641  03BB CE          >          AS    D             AND ADD TO CHECK FOR 0
642  03BC 94 04       >          BNZ   RET1          B IF NOT
643  03BE 29 07 B2    >          JMP   GWHAT         DIDNT EXIST
644  03C1 4D          > RET1     LR    A,I           LOAD
645  03C2 06          >          LR    QU,A          STACK POINTER
646  03C3 4E          >          LR    A,D           FROM
647  03C4 07          >          LR    QL,A          SKGOS
648  03C5 28 09 F7    >          PI    PULLSR        LOAD OLD SKGOS
649  03C8 28 09 F6    >          PI    PULL20        LOAD CURRNT
650  03CB 28 09 E1    >          PI    PULLDC        RESTORE TBP
651  03CE 2C          >          XDC                 SAVE DC
652  03CF 28 09 BE    >          PI    RESTOR        GO RESTORE
653  03D2 2C          >          XDC                 RESTORE TBP
654  03D3 28 03 19    >          PI    FIN           FINISH UP
655  03D6 57 48 41 54 > WHAT     DC    'WHAT'
656  03DA 0D          >          DC    $0D
657  03DB 48 4F 57    > HOW      DC    'HOW'
658  03DE 0D          >          DC    $0D
659                   > *                           3E6-3FF IS USED BY FAIRBUG
660  03DF 53 4F 52 52 > SORRY    DC    'SORRY'
661  03E3 59          >
662  03E4 0D          >          DC    $0D
663  03E5 20 46 41 49 >          DC    ' FAIRBUG USES'
664  03E9 52 42 55 47 >
665  03ED 20 55 53 45 >
666  03F1 53          >
667  03F2 33 45 36 2D >          DC    '3E6-3FF'
668  03F6 33 46 46    >
669  03F9 39 41 42 43 >          DC    '9ABCDEF'
670  03FD 44 45 46    >
671                   > *
672                   > *                           LIST (CR) LISTS ALL SAVED LINES
673                   > *                           LIST N (CR) FROM N DOWN
674                   > *                           LIST N,# (CR) WILL LIST
675                   > *                           # LINES FROM N DOWN
676  0400 28 02 2F    > LIST     PI    TSTNUM        SEE IF A NUMBER
677  0403 31          >          DS    1             WAS IT ?
678  0404 84 27       >          BZ    LIS4          B IF IT WAS NOT
679  0406 44          >          LR    A,4           SEE IF THE
680  0407 25 2C       >          CI    ',',          NEXT ONE IS A COMMA
681  0409 94 22       >          BNZ   LIS4          B IF IT ISNT
682  040B 28 09 EA    >          PI    PUSHSR        SAVE THE LINE #
683  040E 28 08 CE    >          PI    SKIP          GET THE NEXT CHAR
684  0411 28 02 2F    >          PI    TSTNUM        AND THE NEXT NUMBER
685  0414 31          >          DS    1             WAS IT A NUMBER
686  0415 94 04       >          BNZ   LIS1          B IF IT WAS
687  0417 29 07 B2    >          JMP   GWHAT         ELSE ERROR
```

```
688  041A 4C          > LIS1   LR   A,S      IS THE
689  041B CD          >        AS   I        NUM OF LINES GT 255
690  041C 84 04       >        BZ   LIS3     B IF IT ISNT
691  041E 29 02 5B    > LIS2   JMP  GHOW     ELSE ERROR
692  0421 4C          > LIS3   LR   A,S      LOAD THE
693  0422 56          >        LR   6,A      NUMBER OF LINES TO PRINT
694  0423 CE          >        AS   D        SEE IF ZERO
695  0424 84 F9       >        BZ   LIS2     ERROR IF IT IS
696  0426 28 09 F7    >        PI   PULLSR   RESTORE BEGIN LINE #
697  0429 71          >        LIS  1        SET LIST FLAG FOR N,#
698  042A 90 02       >        BR   LIS5     TO LIST #,N
699  042C 70          > LIS4   CLR           SET LIST FLAG
700  042D 55          > LIS5   LR   5,A      TO NO N
701  042E 28 07 F3    >        PI   FNDCR    GET PAST CR
702  0431 28 00 ED    >        PI   FNDLN    FIND A LINE
703  0434 92 10       >        BNC  LIS7     B IF PAST TXTU
704  0436 28 01 17    > LIS6   PI   PRTLN    PRINT A LINE
705  0439 28 00 FC    >        PI   FNDL     GET NEXT LINE
706  043C 92 08       >        BNC  LIS7     B IF PAST TXTU
707  043E 45          >        LR   A,5      SEE IF LOOPING
708  043F C5          >        AS   5        ON N
709  0440 84 F5       >        BZ   LIS6     B IF NOT
710  0442 36          >        DS   6        DEC N
711  0443 94 F2       >        BNZ  LIS6     AND LOOP
712  0445 29 00 03    > LIS7   JMP  START    GO PROMPT
713                   > *
714                   > *                    CLEAR TEXT AREA
715  0448 28 07 F3    > NEW    PI   FNDCR    CLEAR TEXT LINE
716  044B 2A 0C 00    > NEWT   DCI  TXTB     BEGIN ADDR
717  044E 11          >        LR   H,DC     IN 10-11
718  044F 62          >        LISU 2        SET ISAR FOR INITIAL ENTRY
719  0450 6E          >        LISL 6        SET ISAR TO TXTU
720  0451 4A          >        LR   A,10     RESET
721  0452 5D          >        LR   I,A      TO
722  0453 4B          >        LR   A,11     THE BEGINNING
723  0454 5E          >        LR   D,A      OF TEXT AREA
724  0455 29 00 03    >        JMP  START    RESTART
725                   > *
726  0458 28 07 F3    > STOP   PI   ENDCR    FIND CR
727  045B 29 00 03    >        JMP  START    RESTART
728                   > *
729                   > *                    IF TRUE RUN SAME LINE
730                   > *                    IF NOT TRUE RUN NEXT LINE
731  045E 28 05 FD    > IF     PI   EXPR     EVALUATE THE EXPRESSION
732  0461 4D          >        LR   A,I      SEE IF
733  0462 EC          >        XS   S        EXCLUSIVE OR AND
734  0463 CE          >        AS   D        ADD TO CHECK FOR ZERO
735  0464 84 04       >        BZ   REM      B IF IT IS
736  0466 29 05 E7    >        JMP  RSML     RUN SAME LINE
737                   > *                    REMARK IS A FALSE IF
738  0469 28 01 0F    > REM    PI   FNDNXT   FIND THE NEXT LINE
739  046C 92 04       >        BNC  IF1      B PAST THE END
740  046E 29 05 DD    >        JMP  RTSL     RUN NEXT LINE
741  0471 29 00 03    > IF1    JMP  START    NO MORE TEXT
742                   > *
743                   > *                    FOR VAR=EXPR TO EXPR SKIP EXPR
744  0474 28 09 8E    > FOR    PI   SAVE     SAVE FOR VARIABLE
745  0477 28 0B 02    >        PI   SETVAL   GET VARIABLE
746  047A 20 FE       >        LI   -2       BACKUP
747  047C 2C          >        XDC           DC1
748  047D 8E          >        ADC           TO GET VAR ADDR
749  047E 28 09 D5    >        PI   PUSHDC   ON THE STACK
750  0481 2C          >        XDC           RESTORE TBP
751  0482 63          >        LISU 3        PUT VARIABLE
752  0483 6E          >        LISL 6        ADDR INTO
753  0484 28 09 F7    >        PI   PULLSR   LOPVAR
754  0487 62          >        LISU 2        RESET ISAR
755  0488 2A 0B C3    >        DCI  TAB5     GO LOOK
756  048B 29 02 67    >        JMP  EXEC     FOR "TO"
757  048E 28 05 FD    > FR1    PI   EXPR     EVALUATE LIMIT
758  0491 28 09 E9    >        PI   PUSH20   LOPLMT(46-47) ON STACK
759  0494 2A 0B C9    >        DCI  TAB6     GO LOOK
760  0497 29 02 67    >        JMP  EXEC     FOR "STEP"
761  049A 28 05 FD    > FR2    PI   EXPR     EVALUATE INCREMENT
762  049D 90 05       >        BR   FR4      B AROUND DEFAULT OF 1
763  049F 70          > FR3    CLR           NO "SKIP", SET
764  04A0 5D          >        LR   I,A      INCREMENT
765  04A1 1F          >        INC           TO
766  04A2 5E          >        LR   D,A      ONE
767  04A3 28 09 E9    > FR4    PI   PUSH20   LOPINC(44-45)
768  04A6 63          >        LISU 3        USE CURRNT
769  04A7 28 09 E9    >        PI   PUSH20   AS LOFLN(42-43)
770  04AA 28 09 D5    >        PI   PUSHDC   AND TBP AS LOPPT(40-41)
771  04AD 0F          >        LR   DC,Q     STACK ADDR IN DC
772  04AE 28 09 C9    >        PI   REFOR    PUT THEM ALL IN SCRATCH
773  04B1 63          >        LISU 3        SET ISAR FOR LOPVAR
774  04B2 0F          >        LR   DC,Q     SAVE
775  04B3 2C          >        XDC           ORIGINAL STACK ADDR
776  04B4 0F          >        LR   DC,Q     RESET DC0
777  04B5 11          > FR5    LR   H,DC     TEMP SAVE
778  04B6 0E          >        LR   Q,DC     NEW STACK ADDR
779  04B7 16          >        LM            SEE IF
780  04B8 8B          >        OM            THE END
781  04B9 94 04       >        BNZ  FR6      B IF NOT 0
782  04BB 2C          >        XDC           RESTORE ORIGINAL
783  04BC 90 1C       >        BR   FRF      STACK ADDR
784  04BE 6E          > FR6    LISL 6        SET ISAR FOR LOPVAR
785  04BF 28 0A BF    >        PI   COMT     COMPARE TO LOPVAR
786  04C2 7A          >        LIS  10       GO DOWN TO
787  04C3 8F          >        ADC           THE NEXT LEVEL
788  04C4 94 F0       >        BNZ  FR5      B IF FOUND
789  04C6 20 FF       >        LI   -1       GET
790  04C8 8E          >        ADC           FROM ADDR
791  04C9 2C          >        XDC           RESTORE ORIGINAL STACK ADDR
792  04CA 0E          >        LR   Q,DC     PUT
793  04CB 28 09 D5    >        PI   PUSHDC   ON THE STACK TOP
794  04CE 2C          >        XDC           PUT
795  04CF 11          >        LR   H,DC     FROM
796  04D0 2C          >        XDC           IN BOTH
797  04D1 10          >        LR   DC,H     DC0 AND DC1
798  04D2 7A          >        LIS  10       TO ADDR IS
799  04D3 8E          >        ADC           FROM+10
800  04D4 2C          >        XDC           IN DC1
801  04D5 28 09 0C    >        PI   MVDOWN   MOVE THE STACK DOWN 10 BYTES WORTH
802  04D8 10          >        LR   DC,H     RESTORE
803  04D9 0E          > FR8    LR   Q,DC     STACK ADDR
804  04DA 64          >        LISU 4        RESTORE
805  04DB 28 09 E9    >        PI   PUSH20   TBP BY USING
806  04DE 28 09 E1    >        PI   PULLDC   LOPPT
807  04E1 62          >        LISU 2        RESET ISAR
808  04E2 29 03 19    >        JMP  FIN      FINISH UP
809                   > *
810                   > *                    NEXT VAR
811  04E5 28 0B CF    > NEXT   PI   CHAR     GET NEXT CHAR
812  04E8 28 01 BD    >        PI   TESTVL   TEST VARIABLE
813  04EB 2C          >        XDC           VAR ADDR IN DC0
814  04EC 81 05       >        BF   *+6      B AROUND ERROR JUMP
815  04EE 2C          > NX0    XDC           RESTORE TBP
816  04EF 29 07 B2    >        JMP  GWHAT    ERROR
817  04F2 28 09 DF    >        PI   PUSHDC   PUT VAR ADDR(LOPVAR)
818  04F5 28 09 F6    >        PI   PULL20   INTO 20-21
819  04F8 63          > NX1    LISU 3        SET ISAR
820  04F9 6E          >        LISL 6        TO LOPVAR
821  04FA 4D          >        LR   A,I      SEE
822  04FB CC          >        AS   S        IF
823  04FC EE          >        XS   D        ITS ZERO
824  04FD 84 FC       >        BZ   NX0      B IF ZERO AN ERROR
825  04FF 28 09 EA    >        PI   PUSHSR   PUT LOPVAR
826  0502 62          >        LISU 2        ON THE STACK TOP
827  0503 28 0B BD    >        PI   COMP     COMPARE NEXT VARIABLE
828  0506 84 06       >        BZ   NX2      TO LOPVAR & B IF EQ
829  0508 28 09 FE    >        PI   RESTOR   RESTORE NEXT LEVEL
830  050B 90 EC       >        BR   NX1      KEEP LOOKIN
831                   > *                    NEED TO ADD LOPINC TO THE
832                   > *                    VARIABLE LOPVAR POINTS TO.
833  050D 28 09 BF    > NX2    PI   SAVE+1   PUT EVERYTHING BACK SKIP XDC
834  0510 2C          >        XDC           SAVE TBP
835  0511 28 09 E9    >        PI   PUSH20   GET ADDR OF
836  0514 28 09 E1    >        PI   PULLDC   LOPVAR IN DC
837  0517 11          >        LR   H,DC     SAVE LOPVAR ADDR
838  0518 16          >        LM            PUT
839  0519 5D          >        LR   I,A      VALUE
840  051A 16          >        LM            OF LOPVAR
841  051B 5D          >        LR   I,A      IN 20-21
842  051C 0F          >        LR   DC,Q     SAVE
843  051D 76          >        LIS  6        SETUP
844  051E 8E          >        ADC           TEMP STACK
845  051F 0E          >        LR   Q,DC     TO GET LOPINC(44-45)
846  0520 28 09 F8    >        PI   PULLSR+1 INTO 22-23 (SKIP LR H,DC)
847  0523 28 0B 79    >        PI   ADDD     INDEX=INDEX+LOPINC
848  0526 4D          >        LR   A,I      STORE
849  0527 17          >        ST            THE
850  0528 4E          >        LR   A,D      INCREMENTED
851  0529 17          >        ST            LOPVAR
852  052A 28 0B BE    >        PI   COMX     COMPARE(LIMIT-INDEX)
853  052D 0F          >        LR   DC,Q     SAVE THE
854  052E 11          >        LR   H,DC     ORIGINAL STACK ADDR
855  052F 84 10       >        BZ   NX4      B IF NOT DONE
856                   > *                    HERE ASSUMING LOPINC IS +, IT COULD
857                   > *                    BE - SO BOTH GT & LT MUST BE CHECKED
858                   > *                    BACKUP TEMP Q BY 4 BYTES TO LOPINC
859  0531 20 FC       >        LI   -4       RESET TEMP STACK
860  0533 8E          >        ADC           TO LOPINC
861  0534 70          >        CLR           SEE
```

```
 862  0535 8B          >          OM              IF MINUS
 863  0536 91 06       >          BM    NX3       B IF NEGATIVE INC
 864  0538 1D          >          LR    W,J       RESTORE STATUS
 865  0539 91 19       >          BM    NX5       + INC, BR IF DONE(-)
 866  053B 90 04       >          BR    NX4       NOT DONE
 867  053D 1D          > NX3      LR    W,J       RESTORE STATUS
 868  053E 81 14       >          BP    NX5       - INC, BR IF DONE(+)
 869                   > *                        STILL LOOPING, SET CURRNT=LOPVAR
 870                   > *                        AND DC0=LOPPT THE NEW TBP=
 871  0540 0F          > NX4      LR    DC,Q      RESET
 872  0541 20 F8       >          LI    -8        STACK
 873  0543 8E          >          ADC             TO POINT
 874  0544 0E          >          LR    Q,DC      TO LOPPT(40-41)
 875  0545 10          >          LR    DC,H      SAVE
 876  0546 2C          >          XDC             ORIGINAL STACK IN DC1
 877  0547 28 09 E1    >          PI    PULLDC    GET TBP=LOPPT
 878  054A 63          >          LISU  3         SET
 879  054B 28 09 F6    >          PI    PULL20    CURRNT=LOPLN
 880  054E 62          >          LISU  2         RESET ISAR
 881  054F 2C          >          XDC             RESET
 882  0550 0E          >          LR    Q,DC      STACK POINTER
 883  0551 90 06       >          BR    NX6       GO FINISH LINE
 884  0553 10          > NX5      LR    DC,H      RESET ORIGINAL
 885  0554 0E          >          LR    Q,DC      STACK
 886  0555 28 09 BE    >          PI    RESTOR    GET NEXT FOR-NEXT LEVEL
 887  0558 2C          > NX6      XDC             RESTORE TBP
 888  0559 29 03 19    >          JMP   FIN       FINISH UP
 889                   > *
 890                   > *                        READ A,B$,@(I),@$(I)
 891  055C 28 08 CF    > READ     PI    CHAR      GET NEXT CHAR
 892  055F 28 01 BD    >          PI    TESTVL    A VARIABLE ?
 893  0562 28 09 D5    >          PI    PUSHDC    SAVE TBP
 894  0565 81 04       >          BP    *+5
 895  0567 29 07 B5    > REA1     JMP   AWHAT     ERROR
 896  056A 65          >          LISU  5         SET ISAR
 897  056B 6A          >          LISL  2         TO TEMP POINTER
 898  056C 4D          >          LR    A,I       GET PRESENT
 899  056D 5A          >          LR    10,A      DATA
 900  056E 4E          >          LR    A,D       ADDR
 901  056F 5B          >          LR    11,A      INTO
 902  0570 10          >          LR    DC,H      DC0
 903  0571 62          >          LISU  2         RESET ISAR
 904  0572 28 07 FC    >          PI    RDATA     PROCESS DATA ITEM
 905  0575 28 08 CF    >          PI    CHAR      GET NEXT CHAR
 906  0578 25 2C       >          CI    *,*       SEE IF A COMMA
 907  057A 84 05       >          BZ    REA2      B IF IT IS
 908  057C 25 0D       >          CI    $0D       CR ?
 909  057E 94 E8       >          BNZ   REA1      ERROR IF NOT
 910  0580 28 08 CE    > REA2     PI    SKIP      GET PAST , OR CR
 911  0583 65          >          LISL  5         SETUP
 912  0584 6A          >          LISL  2         TEMP POINTER
 913  0585 11          >          LR    H,DC      STORE
 914  0586 4A          >          LR    A,10      THE
 915  0587 5D          >          LR    I,A       PRESENT
 916  0588 4B          >          LR    A,11      DATA
 917  0589 5E          >          LR    D,A       POINTER
 918  058A 62          >          LISU  2         RESET ISAR
 919  058B 28 09 E1    >          PI    PULLDC    RESTORE TBP
 920  058E 28 08 D8    >          PI    IGNBK     WAS LAST CHAR
 921  0591 25 2C       >          CI    *,*       A COMMA ?
 922  0593 84 C8       >          BZ    READ      B IF MORE
 923  0595 29 03 16    >          JMP   FINI      FINISH UP
 924                   > *
 925                   > *
 926                   > *                        DATA N,*STRING*,EXPR
 927  0598 11          > DATA     LR    H,DC      SAVE TBP
 928  0599 65          >          LISU  5         SET ISAR
 929  059A 68          >          LISL  0         TO DATA POINTER
 930  059B 4D          >          LR    A,I       SEE IF
 931  059C EC          >          XS    S         ZERO BY
 932  059D CE          >          AS    D         XOR AND ADDING
 933  059E 94 09       >          BNZ   DAT1      B IF NOT ZERO
 934  05A0 4A          >          LR    A,10      SETUP
 935  05A1 5D          >          LR    I,A       PERMANENT
 936  05A2 4B          >          LR    A,11      DAT
 937  05A3 5D          >          LR    I,A       POINTER
 938  05A4 4A          >          LR    A,10      AND THE
 939  05A5 5D          >          LR    I,A       TEMP
 940  05A6 4B          >          LR    A,11      POINTER
 941  05A7 5D          >          LR    I,A       AS WELL
 942  05A8 62          > DAT1     LISU  2         RESET ISAR
 943  05A9 10          >          LR    DC,H      RESTORE TBP
 944  05AA 29 04 69    >          JMP   REM       RUN THE NEXT LINE
 945                   > *
 946                   > *                        RESET DATA POINTER
 947  05AD 65          > RESTORE  LISU  5         SET ISAR
 948  05AE 28 08 F1    >          PI    MV2021    RESET THE POINTER

 949  05B1 62          >          LISU  2         RESET ISAR
 950  05B2 29 03 19    >          JMP   FIN       FINISH UP
 951                   > *
 952                   > *                        FIND THE LINE NUMBER AND GOTO I
 953  05B5 28 05 FD    > GOTO     PI    EXPR      EVALUATE THE EXPRESSION
 954  05B8 28 09 D5    >          PI    PUSHDC    SAVE DC IN CASE OF ERROR
 955  05BB 28 07 F3    >          PI    ENDCR     FIND CR
 956  05BE 28 00 ED    >          PI    FNDLN     FIND THE LINE
 957  05C1 94 06       >          BNZ   GO1       B IF NOT FOUND
 958  05C3 28 0A 15    >          PI    POPRT     CLEAR THE STACK
 959  05C6 90 16       >          BR    RTSL      RUN THE LINE
 960  05C8 29 02 5F    > GO1      JMP   AHOW      PROCESS ERROR
 961                   > *
 962  05CB 28 07 F3    > RUN      PI    ENDCR     FIND CR
 963  05CE 2A 0C 00    >          DCI   TXTB      BEGIN OF TEXT ADDR
 964                   > *                        RUN THE NEXT LINE
 965  05D1 68          > RNXL     LISL  0         SET SCRATCH
 966  05D2 70          >          CLR             20-21
 967  05D3 5D          >          LR    I,A       THE LINE NUMBER
 968  05D4 5E          >          LR    D,A       TO ZERO
 969  05D5 28 00 F0    >          PI    FNDL      FIND NEXT LINE
 970  05D8 82 04       >          BC    RTSL      B IF NOT PAST TXTU
 971  05DA 29 00 03    >          JMP   START     RESTART
 972                   > *                        SAVE CURRENT LINE ADDR
 973  05DD 63          > RTSL     LISU  3         GET
 974  05DE 68          >          LISL  0         VALUE OF
 975  05DF 4A          >          LR    A,10      CURRNT
 976  05E0 5D          >          LR    I,A       STORE
 977  05E1 4B          >          LR    A,11      NEW
 978  05E2 5E          >          LR    D,A       LINE NUMBER
 979  05E3 62          >          LISU  2         RESET ISAR
 980  05E4 10          >          LR    DC,H      GET PAST
 981  05E5 16          >          LM              LINE NUMBER
 982  05E6 16          >          LM              LINE NUMBER
 983                   > *                        RUN THE SAME LINE
 984  05E7 11          > RSML     LR    H,DC      PUT LINE ADDR IN H
 985  05E8 70          >          CLR             CLEAR
 986  05E9 55          >          LR    5,A       STRING FLAG
 987  05EA 2A 0B 32    >          DCI   TAB2      TB COMMANDS TABLE
 988  05ED 29 02 67    >          JMP   EXEC      GO PROCESS
 989                   > *
 990                   > *                        FINISH UP THE LINE
 991  05F0 08          > FINISH   LR    K,P       SAVE RETURN
 992  05F1 28 0B DB    >          PI    IGNBK     GET NEXT CHAR
 993  05F4 25 3B       >          CI    *;*       MULTIPLE STATEMENT ?
 994  05F6 84 F0       >          BZ    RSML      YES RUN SAME LINE
 995  05F8 25 0D       >          CI    $0D       CR ?
 996  05FA 84 D6       >          BZ    RNXL      YES RUN NEXT LINE
 997  05FC 0C          >          PK              RETURN
 998                   > *
 999                   > *                        EVALUATE AN EXPRESSION
1000  05FD 08          > EXPR     LR    K,P       SAVE
1001  05FE 28 0A 00    >          PI    PUSHRT    RETURN
1002  0601 28 06 69    >          PI    EXPR2     GET 1ST EXPRESSION
1003  0604 28 09 E9    >          PI    PUSH20    SAVE 1ST EXPR
1004  0607 11          > EXPR1    LR    H,DC      SAVE TBP
1005  0608 2A 0B D1    >          DCI   TAB8      RELATIONAL OPERATOR TAPLE
1006  060B 29 02 67    >          JMP   EXEC      GO SEE IF WE HAVE OP
1007                   > *
1008  060E 28 06 44    > XP11     PI    XP18      ">="
1009  0611 82 29       >          BC    TRUE      B TRUE
1010  0613 90 26       >          BR    FALSE     SET FALSE=0
1011  0615 28 06 44    > XP12     PI    XP18      "#"
1012  0618 94 22       >          BNZ   TRUE      B TRUE
1013  061A 90 1F       >          BR    FALSE     SET FALSE=0
1014  061C 28 06 44    > XP13     PI    XP18      ">"
1015  061F 84 1A       >          BZ    FALSE     SET FALSE=0
1016  0621 82 19       >          BC    TRUE      B TRUE
1017  0623 90 16       >          BR    FALSE     SET FALSE=0
1018  0625 28 06 44    > XP14     PI    XP18      "<="
1019  0628 92 12       >          BNC   TRUE      B TRUE
1020  062A 84 10       >          BZ    TRUE      B TRUE
1021  062C 90 0D       >          BR    FALSE     SET FALSE=0
1022  062E 28 06 44    > XP15     PI    XP18      "="
1023  0631 84 09       >          BZ    TRUE      B TRUE
1024  0633 90 06       >          BR    FALSE     SET FALSE=0
1025  0635 28 06 44    > XP16     PI    XP18      "<"
1026  063A 92 02       >          BNC   TRUE      B TRUE
1027  063A 3E          > FALSE    DS    D         SET R21=0
1028  063B 29 0A 0C    > TRUE     JMP   PULLRT    RETURN
1029                   > *                        NOT A RELATIONAL OPERATOR
1030  063E 28 09 F7    > XP17     PI    PULLSR    GET 1ST EXPRESSION
1031  0641 29 0A 0C    >          JMP   PULLRT    RETURN
1032                   > *                        GET 2ND EXPR AND COMPARE
1033  0644 08          > XP18     LR    K,P       SAVE
1034  0645 28 0A 00    >          PI    PUSHRT    RETURN
1035  0648 70          >          CLR             CLEAR
```

```
1036  0649 55          >          LR    5,A       STRING FLAG
1037  064A 28 06 69    >          PI    EXPR2     GET 2ND EXPR
1038  064D 28 0A 15    >          PI    POPRT     POP RETURN INTO K
1039                   > *                       IF THE 2 ITEMS HAVE THE SAME SIGN
1040                   > *                       LEAVE AS IS, BUT IF THE SIGN IS
1041                   > *                       DIFFERENT EXCHANGE THE STACK AND ISAR
1042  0650 0F          >          LR    DC,Q      STACK ADDR
1043  0651 4C          >          LR    A,S       LOAD HI BYTE
1044  0652 8C          >          XM              XOR THEM
1045  0653 10          >          LR    DC,H      RESTORE TBP
1046  0654 81 0B       >          BP    XP19      B IF THE SAME SIGN
1047  0656 28 08 F1    >          PI    MV2021    MOVE 2ND EXPR TO 22-23
1048  0659 28 09 F7    >          PI    PULLSR    STACK(1ST EXPR) INTO 20-21
1049  065C 6A          >          LISL  2         2ND EXPR
1050  065D 28 09 EA    >          PI    PUSHSR    ONTO STACK TOP
1051  0660 28 0B BD    > XP19     PI    COMP      COMPARE 1ST AND 2ND EXPR
1052  0663 70          >          CLR             SET
1053  0664 5D          >          LR    I,A       20-21=1
1054  0665 1F          >          INC             FOR
1055  0666 5C          >          LR    S,A       TRUE AND SET LISL 1
1056  0667 1D          >          LR    W,J       RESTORE STATUS
1057  0668 0C          >          PK              RETURN
1058                   > *                       DC MUST CONTAIN TBP
1059  0669 08          > EXPR2    LR    K,P       SAVE
1060  066A 28 0A 00    >          PI    PUSHRT    RETURN
1061  066D 68          >          LISL  0         SET ISAR
1062  066E 28 0B CF    >          PI    CHAR      GET NEXT CHARACTER
1063  0671 25 2D       >          CI    "-"       MINUS ?
1064  0673 94 06       >          BNZ   XP21      B IF NOT
1065  0675 70          >          CLR             SET
1066  0676 5D          >          LR    I,A       FIRST
1067  0677 5E          >          LR    D,A       EXPR=0
1068  0678 90 37       >          BR    XP26      TREAT LIKE SUBTRACT
1069  067A 25 2B       > XP21     CI    "+"       PLUS ?
1070  067C 94 04       >          BNZ   XP22      B IF NOT
1071  067E 28 0B CE    >          PI    SKIP      INC DC AND GET NEXT CHAR
1072  0681 28 06 BE    > XP22     PI    EXPR3     PROCESS FIRST EXPR
1073  0684 44          > XP23     LR    A,4       LOAD THE CHAR
1074  0685 68          >          LISL  0         RESET TO 20-21
1075  0686 25 2B       >          CI    "+"       ADD ?
1076  0688 94 20       >          BNZ   XP25      B IF NOT
1077  068A 28 09 EA    >          PI    PUSHSR    SAVE FIRST EXPR
1078  068D 28 0B CE    >          PI    SKIP      INC DC AND GET NEXT CHAR
1079  0690 28 06 BE    >          PI    EXPR3     PROCESS 2ND EXPR
1080  0693 6A          > XP24     LISL  2         GET 1ST
1081  0694 28 09 F7    >          PI    PULLSR    EXPR INTO 22-23
1082  0697 4C          >          LR    A,S       LOAD HI
1083  0698 68          >          LISL  0         SET TO OTHER HI
1084  0699 EC          >          XS    S         EXCLUSIVE OR
1085  069A 1E          >          LR    J,W       SAVE STATUS OF SIGNS
1086  069B 28 0B 79    >          PI    ADDD      ADD
1087  069E 1D          >          LR    W,J       RESTORE STATUS
1088  069F 91 E4       >          BM    XP23      B IF SIGNS DIFFER
1089  06A1 4C          >          LR    A,S       SIGNS THE SAME
1090  06A2 6A          >          LISL  2         SO MUST BE RESULT
1091  06A3 EC          >          XS    S         EQUAL ?
1092  06A4 81 DF       >          BP    XP23      B IF THEY ARE
1093  06A6 29 02 5B    >          JMP   QHOW      PROCESS ERROR
1094  06A9 25 2D       > XP25     CI    "-"       MINUS ?
1095  06AB 84 04       >          BZ    XP26      B IF MINUS
1096  06AD 29 07 96    >          JMP   XP45      RETURN
1097  06B0 28 09 EA    > XP26     PI    PUSHSR    SAVE FIRST EXPR
1098  06B3 28 0B CE    >          PI    SKIP      INC DC AND GET NEXT CHAR
1099  06B6 28 06 BF    >          PI    EXPR3     GET 2ND EXPR
1100  06B9 28 0B 52    >          PI    CHGSGN    CHANGE SIGN
1101  06BC 90 D6       >          BR    XP24      GO ADD
1102                   > *
1103  06BE 08          > EXPR3    LR    K,P       SAVE
1104  06BF 28 0A 00    >          PI    PUSHRT    RETURN
1105  06C2 28 07 3A    >          PI    EXPR4     GET FIRST EXPR
1106  06C5 44          > XP30     LR    A,4       LOAD THE CHAR
1107  06C6 25 2A       > XP31     CI    "*"       MULTIPLY ?
1108  06C8 94 33       >          BNZ   XP34      B IF NOT
1109  06CA 28 09 E9    >          PI    PUSH2D    SAVE 1ST EXPR
1110  06CD 28 0B CE    >          PI    SKIP      INC DC AND GET NEXT CHAR
1111  06D0 28 07 3A    >          PI    EXPR4     GET 2ND EXPR
1112  06D3 28 0B 4C    >          PI    CHKSGN    CHECK SIGN
1113  06D6 48          >          LR    A,B       SAVE
1114  06D7 52          >          LR    2,A       THE SIGN
1115  06D8 28 0B E0    >          PI    EXCH      EXCHANGE STACK AND 20-21
1116  06DB 28 0B 4C    >          PI    CHKSGN    CHECK THE SIGN
1117  06DE 20 FF       >          LI    $FF       SEE IF
1118  06E0 F6          >          NS    6         HI GT 255
1119  06E1 84 09       >          BZ    XP32      B LE 255
1120                   > *                       NUMBER IN 20-21 GT 255
1121  06E3 20 FF       >          LI    $FF       SEE IF
1122  06E5 FC          >          NS    S         THIS GT 255
1123  06E6 94 12       >          BNZ   XP33      B IF GT 255(OVERFLOW WILL OCCUR)
1124  06E8 28 0B EC    >          PI    EXCH      SWITCH SO SMALL IN R7-8
1125  06EB 28 08 F1    > XP32     PI    MV2021    MOVE 20-21 TO 22-23
1126  06EE 70          >          CLR             ZERO
1127  06EF 5D          >          LR    I,A       20-21
1128  06F0 5E          >          LR    D,A       FOR A MULT
1129  06F1 28 0B 75    >          PI    MTPLY     GO MULTIFLY
1130  06F4 28 0A 15    >          PI    POPRT     CLEAR STACK
1131  06F7 81 31       >          BP    XP35      TAKE CARE OF SIGNS
1132  06F9 29 02 5B    > XP33     JMP   QHOW      PROCESS ERROR
1133  06FC 25 2F       > XP34     CI    "/"       DIVIDE ?
1134  06FE 84 04       >          BZ    *+5       B IF /
1135  0700 29 07 96    >          JMP   XP45      GO RETURN
1136  0703 28 09 E9    >          PI    PUSH2D    SAVE 1ST EXPR
1137  0706 28 0B CE    >          PI    SKIP      INC DC AND GET NEXT CHAR
1138  0709 28 07 3A    >          PI    EXPR4     GET 2ND EXPR
1139  070C 28 0B 4C    >          PI    CHKSGN    CHECK SIGN OF 2ND
1140  070F 48          >          LR    A,B       SAVE
1141  0710 52          >          LR    2,A       SIGN
1142  0711 28 0B E0    >          PI    EXCH      SWITCH FIRST INTO SCRATCH
1143  0714 28 0B 4C    >          PI    CHKSGN    CHECK SIGN OF 1ST EXPR
1144  0717 6A          >          LISL  2         PUT 2ND
1145  0718 28 09 F7    >          PI    PULLSR    EXPRESSION IN 22-23
1146  071B 4D          >          LR    A,I       SEE IF
1147  071C EC          >          XS    S         EXCLUSIVE OR AND
1148  071D CE          >          AS    D         ADD TO CHECK FOR ZERO
1149  071E 84 DA       >          BZ    XP33      B IF WE ARE
1150  0720 28 0B FB    >          PI    DIVIDE    GO DIVIDE
1151  0723 28 09 EA    >          PI    PUSHSR    MOVE
1152  0726 28 09 F6    >          PI    PULL2D    RESULT IN 20-21
1153                   > *                       ADJUST SIGN OF RESULT
1154  0729 70          > XP35     CLR             SEE IF
1155  072A CC          >          AS    S         RESULT IS +
1156  072B 91 CD       >          BM    XP33      B IF IT ISNT
1157  072D 42          >          LR    A,2       SEE IF
1158  072E CB          >          AS    B         SIGNS DIFFER
1159  072F 84 95       >          BZ    XP30      B IF 2 +'S
1160  0731 25 2D       >          CI    "-"       IS JUST ONE A -
1161  0733 94 91       >          BNZ   XP30      B IF 2 -'S
1162  0735 28 0B 52    >          PI    CHGSGN    CHANGE THE SIGN
1163  0738 90 8C       >          BR    XP30      CONTINUE
1164                   > *                       EVALUATE THE INPUT
1165  073A 08          > EXPR4    LR    K,P       SAVE
1166  073B 28 0A 00    >          PI    PUSHRT    RETURN
1167  073E 11          >          LR    H,DC      SAVE TBP
1168  073F 2A 0B 9C    >          DCI   TAB4      FUNCTION TABLE
1169  0742 29 02 67    >          JMP   EXEC      GO LOOK
1170  0745 28 01 BD    > XP40     PI    TESTVL    TEST THE VALUE
1171  0748 91 47       >          BM    XP44      B IF NOT A VAR
1172                   > *                       CHECK FOR A STRING
1173  074A 68          >          LISL  0         SET ISAR
1174  074B 45          >          LR    A,5       GET
1175  074C 59          >          LR    S,A       STRING
1176  074D 1D          >          LR    W,J       FLAG
1177  074E 91 39       >          BM    XP43      B IF NOT A STRING
1178                   > *                       PROCESS STRING
1179  0750 28 09 D5    >          PI    PUSHDC    SAVE TBP
1180  0753 2A 13 37    >          DCI   BUFF      TO ADDR
1181  0756 2C          >          XDC             RESTORE VAR ADDR
1182  0757 11          >          LR    H,DC      SAVE VAR ADDR
1183  0758 70          >          CLR             ZERO ACCUM
1184  0759 1D          >          LR    W,J       GET STATUS
1185  075A 94 04       >          BNZ   *+5       B IF A-Z
1186  075C 16          >          LM              ADJUST ARRAY LOC
1187  075D 20 FE       >          LI    -2        GET VAR
1188  075F 51          >          LR    1,A       INTO R1
1189  0760 20 48       >          LI    72        SETUP MAX
1190  0762 52          >          LR    2,A       DIGIT COUNTER
1191  0763 32          > XP41     DS    2         CHECK LENGTH
1192  0764 94 04       >          BNZ   *+5       B IF NOT TOO LONG
1193  0766 29 02 5E    >          JMP   AHOW      ERROR
1194  0769 16          >          LM              LOAD A CHAR
1195  076A 2C          >          XDC             GET BUFF ADDR
1196  076B 17          >          ST              STORE CHAR
1197  076C 21 80       >          NI    $80       LAST CHAR ?
1198  076E 94 06       >          BNZ   XP42      B IF LAST
1199  0770 2C          >          XDC             RESET VAR ADDR
1200  0771 41          >          LR    A,1       ADJUST
1201  0772 8E          >          ADC             VAR ADDR
1202  0773 90 EF       >          BR    XP41      CONTINUE
1203  0775 70          > XP42     CLR             MARK END OF STRING
1204  0776 17          >          ST              OF THE STRING
1205  0777 28 09 84    >          PI    SLEN      STORE LENGTH
1206  077A 10          >          LR    DC,H      RESTORE
1207  077B 2C          >          XDC             VAR ADDR
1208  077C 10          >          LR    DC,H      VAR ADDR
1209  077D 16          >          LM              LOAD
```

```
1210  077E 5D          >          LR    I,A       THE
1211  077F 16          >          LM              1ST AND 2ND CHARS
1212  0780 21 7F       >          NI    $7F       INSURE HIGH
1213  0782 5D          >          LR    I,A       BIT IS OFF
1214  0783 28 09 E1    >          PI    PULLDC    RESTORE TBP
1215  0786 90 0F       >          BR    XP45      GO RETURN
1216  0788 2C          > XP43     XDC             GET VAR ADDR
1217  0789 16          >          LM              LOAD THE
1218  078A 5D          >          LR    I,A       VALUE OF
1219  078B 16          >          LM              THE VARIABLE
1220  078C 5E          >          LR    D,A       INTO 20-21 LISL 3
1221  078D 2C          >          XDC             RESTORE TBP
1222  078E 90 07       >          BR    XP45      RETURN
1223  0790 28 02 2F    > XP44     PI    TSTNUM    SEE IF A NUMBER
1224  0793 31          >          DS    1         NUMBER ?
1225  0794 84 0B       >          BZ    PAR1      B IF NOT A NUMBER
1226  0796 29 0A 0C    > XP45     JMP   PULLRT    RETURN
1227                   > *
1228  0799 08          > PARN     LR    K,P       SAVE
1229  079A 28 0A 00    >          PI    PUSHRT    RETURN
1230  079D 28 08 DB    > PAR1     PI    IGNBK     GET NEXT CHARACTER
1231  07A0 25 28       >          CI    "("       LEFT HAND PAREN ?
1232  07A2 94 0F       >          BNZ   QWHAT     B IF NOT
1233  07A4 28 05 FD    >          PI    EXPR      EVALUATE EXPRESSION
1234  07A7 44          >          LR    A,4       RESTORE CHARACTER
1235  07A8 25 29       >          CI    ")"       CLOSING PAREN ?
1236  07AA 94 07       >          BNZ   QWHAT     B IF NOT FOUND
1237  07AC 28 08 CE    >          PI    SKIP      INC DC AND GET NEXT CHAR
1238  07AF 29 0A 0C    >          JMP   PULLRT    RETURN
1239                   > *
1240  07B2 28 09 D5    > QWHAT    PI    PUSHDC    SAVE TBP
1241  07B5 2A 03 D6    > AWHAT    DCI   WHAT      WHAT MESSAGE
1242  07B8 28 01 30    > ERROR    PI    PRTSTG    PRINT "WHAT,HOW, OR SORRY
1243  07BB 28 09 E1    >          PI    PULLDC    GET TBP
1244  07BE 20 FF       >          LI    -1        GET
1245  07C0 8E          >          ADC             POINTER TO CHAR
1246  07C1 11          >          LR    H,DC      IN ERROR
1247  07C2 16          >          LM              SAVE
1248  07C3 55          >          LR    5,A       THE CHAR
1249  07C4 70          >          CLR             PUT
1250  07C5 10          >          LR    DC,H      A ZERO
1251  07C6 17          >          ST              THERE
1252  07C7 63          >          LISU  3         GET CURRNT
1253  07C8 68          >          LISL  0         IN SCRATCH 30
1254  07C9 4C          >          LR    A,S       SEE
1255  07CA CC          >          AS    S         IF A MINUS
1256  07CB 81 04       >          BF    **5       B IF GE 0
1257  07CD 29 02 90    >          JMP   INPERR    REDO INPUT
1258  07D0 4D          >          LR    A,I       SEE
1259  07D1 EC          >          XS    S         EXCLUSIVE OR AND
1260  07D2 CE          >          AS    D         ADD TO CHECK FOR ZERO
1261  07D3 94 04       >          BNZ   **5       B IF NOT 0
1262  07D5 29 00 03    >          JMP   START     RESTART
1263  07D8 28 09 EA    >          PI    PUSHSR    VALUE
1264  07DB 28 09 E1    >          PI    PULLDC    OF CURRNT IN DC
1265  07DE 62          >          LISU  2         RESET SCRATCH
1266  07DF 28 01 17    >          PI    PRTLN     PRINT THE LINE
1267  07E2 20 FF       >          LI    -1        UPTO THE 0
1268  07E4 8E          >          ADC             ADJUST DC
1269  07E5 45          >          LR    A,5       PUT THE
1270  07E6 17          >          ST              CHAR BACK
1271  07E7 20 3F       >          LI    "?"       OUTPUT
1272  07E9 51          >          LR    1,A       A
1273  07EA 28 0B ED    >          PI    TTYO      ?
1274  07ED 28 01 30    >          PI    PRTSTG    PRINT REST OF THE LINE
1275  07F0 29 00 03    >          JMP   START     RESTART
1276                   > *
1277                   > *                     FIND END OF LINE
1278  07F3 08          > ENDCR    LR    K,P       SAVE RETURN
1279  07F4 28 08 DB    >          PI    IGNBK     GET NEXT CHAR
1280  07F7 25 0D       >          CI    $0D       IS IT A CR ?
1281  07F9 94 B8       >          BNZ   QWHAT     B IF NOT
1282  07FB 0C          >          PK              RETURN
1283                   > *
1284                   > *                     PROCESS A READ
1285  07FC 08          > RDATA    LR    K,P       SAVE RETURN
1286  07FD 28 0A 00    >          PI    PUSHRT
1287  0800 90 14       >          BR    SETRH     PROCESS READ-DATA
1288                   > *                     STORES THE RIGHT HAND
1289                   > *                     EXPRESSION IN THE VARIABLE
1290  0802 08          > SETVAL   LR    K,P       SAVE
1291  0803 28 0A 00    >          PI    PUSHRT    RETURN
1292  0806 28 08 CF    >          PI    CHAR      GET NEXT CHAR
1293  0809 28 01 BD    >          PI    TESTVL    TEST THE VARIABLE
1294  080C 91 A5       >          BM    QWHAT     B IF NOT THERE
1295  080E 28 08 D8    >          PI    IGNBK     GET NEXT CHAR
1296  0811 25 3D       >          CI    "="       GET PAST EQUAL
```

```
1297  0813 94 9E       >          BNZ   QWHAT     B NOT THERE
1298  0815 45          > SETRH    LR    A,5       SAVE 1ST STRING FLAG
1299  0816 56          >          LR    6,A       IN R6
1300  0817 70          >          CLR             CLEAR
1301  0818 55          >          LR    5,A       STRING FLAG
1302  0819 2C          >          XDC             SAVE
1303  081A 28 09 D5    >          PI    PUSHDC    VARIABLE ADDR
1304  081D 2C          >          XDC             TBP BACK IN DC0
1305  081E 28 05 FD    >          PI    EXPR      EVALUATE EXPRESSION
1306  0821 2C          >          XDC             SAVE TBP AND GET RH VAR ADDR
1307  0822 28 09 E1    >          PI    PULLDC    GET LH VAR ADDR
1308                   > *                     STRING ?
1309  0825 45          >          LR    A,5       LOAD
1310  0826 59          >          LR    9,A       RH
1311  0827 1D          >          LR    W,J       STRING FLAG
1312  0828 46          >          LR    A,6       LOAD
1313  0829 59          >          LR    9,A       LH STRING FLAG
1314  082A 91 16       >          BM    SET3      B IF RH NOT A STRING
1315  082C 55          >          LR    5,A       SAVE LH STRING FLAG
1316  082D 1D          >          LR    W,J       AND CHECK ITS STATUS
1317  082E 81 05       >          BF    SET2      B IF A STRING
1318  0830 2C          > SET1     XDC             GET TBP
1319  0831 29 07 B2    >          JMP   QWHAT     ERROR
1320  0834 2C          > SET2     XDC             SAVE
1321  0835 28 09 D5    >          PI    PUSHDC    TBP
1322  0838 2C          >          XDC             GET VAR ADDR
1323  0839 28 09 22    >          PI    BUFTOV    MOVE THE STRING
1324  083C 28 09 E1    >          PI    PULLDC    RESTORE TBP
1325  083F 90 09       >          BR    SETR      RETURN
1326  0841 1D          > SET3     LR    W,J       GET LH STRING FLAG
1327  0842 81 ED       >          BP    SET1      ERROR IF A STRING
1328  0844 4D          > SET4     LR    A,I       STORE
1329  0845 17          >          ST              VALUE
1330  0846 4E          >          LR    A,D       IN
1331  0847 17          >          ST              VARIABLE
1332  0848 2C          >          XDC             RESTORE TBP
1333  0849 29 0A 0C    > SETR     JMP   PULLRT    RETURN
1334                   > *
1335                   > *                     CHECK SIGN OF NUMBER IN 20-21
1336                   > *                     R8=0 FOR + AND "-" FOR -
1337  084C 68          > CHKSGN   LISL  0         ISAR TO 20
1338  084D 70          >          CLR             ZERO
1339  084E 58          >          LR    8,A       ZERO FOR +
1340  084F CC          >          AS    S         ADD HI ORDER BYTE
1341  0850 81 11       >          BP    CHG2      B IF 0 OR -
1342  0852 69          > CHGSGN   LISL  1         LOAD LOW
1343  0853 4C          >          LR    A,S       ORDER
1344  0854 18          >          COM             TWO'S
1345  0855 1F          >          INC             COMPLEMENT
1346  0856 5E          >          LR    D,A       BACKIN 21
1347  0857 1E          >          LR    J,W       SAVE STATUS
1348  0858 4C          >          LR    A,S       LOAD HI
1349  0859 18          >          COM             COMPLEMENT
1350  085A 1D          >          LR    W,J       RESTORE STATUS
1351  085B 92 02       >          BNC   CHG1      B IF NO ADJUST
1352  085D 1F          >          INC             ADJUST IF NO CARRY
1353  085E 5C          > CHG1     LR    S,A       SAVE IN R20
1354  085F 20 2D       >          LI    "-"       LOAD A MINUS
1355  0861 58          >          LR    8,A       INTO R8
1356  0862 1C          > CHG2     POP             RETURN
1357                   > *
1358                   > *                     SCRATCH (20-21)=(20-21)-(22-23)
1359  0863 6B          > SUBD     LISL  3         SET ISAR
1360  0864 4C          >          LR    A,S       LOAD
1361  0865 18          >          COM             MAKE IT
1362  0866 1F          >          INC             A MINUS
1363  0867 69          >          LISL  1         GET OTHER LOW ORDER
1364  0868 CC          >          AS    S         ADD IT
1365  0869 1E          >          LR    J,W       SAVE STATUS
1366  086A 5D          >          LR    I,A       SAVE 21 MINUS 23
1367  086B 4E          >          LR    A,D       LOAD HI ORDER
1368  086C 18          >          COM             COMPLEMENT
1369  086D 1D          >          LR    W,J       RESTORE CARRY STATUS
1370  086E 92 02       >          BNC   NCS       B IF THERE WAS A CARRY
1371  0870 1F          >          INC             MAKE 2'S COMPLEMENT IF NO CARRY
1372  0871 68          > NCS      LISL  0         SET TO HI ORDER
1373  0872 CC          >          AS    S         ADD HI ORDER
1374  0873 5C          >          LR    S,A       SAVE IT AND SET LISL 0
1375  0874 1C          >          POP             RETURN
1376                   > *
1377                   > *                     SCRATCH (20-21)=(20-21)+(22-23)
1378                   > *                     MULT IS AN ADDITIONAL ENTRY
1379                   > *                     SCRATCH (20-21)=(22-23)*R3
1380                   > *                     20-21 SHOULD BE ZERO FOR A MULT ENTRY
1381  0875 70          > MTPLY    CLR             CHECK FOR
1382  0876 C7          >          AS    7         A ZERO
1383  0877 90 0E       >          BR    MULT      MULTIPLIER
```

```
1384  0879 71          > ADDC    LIS   1          FOR ADD
1385  087A 57          >         LR    7,A        FOR ADD ENTRY OVERRIDE MULT
1386  087B 6B          > MULT    LISL  3          SET ISAR
1387  087C 4C          >         LR    A,S        LOAD 23
1388  087D 69          >         LISL  1          ADD
1389  087E CC          >         AS    S          21 TO IT
1390  087F 5D          >         LR    I,A        SAVE
1391  0880 4E          >         LR    A,D        LOAD 22
1392  0881 19          >         LNK              ADD CARRY
1393  0882 6A          >         LISL  0          ADD
1394  0883 CC          >         AS    S          20
1395  0884 5C          >         LR    S,A        SAVE NEW 20
1396  0885 37          >         DS    7          DEC MULTIPLIER
1397  0886 94 FA       > MUL1    BNZ   MULT       B IF A MULTIPLY
1398  0888 70          >         CLR              RESET
1399  0889 CC          >         AS    S          STATUS
1400  088A 1C          >         POP              RETURN WITH LISL 0
1401                   > *
1402                   > *                        DIVIDE SCRATCH (20-21) BY (22-23)
1403                   > *                        RESULT IN SCRATCH (24-25)
1404                   > *                        REMAINDER IN SCRATCH (20-21)
1405  088B 08          > DIVIDE  LR    K,P        SAVE RETURN
1406  088C 28 0A 00    >         PI    PUSHRT     PUSH RETURN
1407  088F 28 09 E9    >         PI    PUSH20     SAVE 20-21
1408  0892 4D          >         LR    A,I        SET
1409  0893 5E          >         LR    D,A        21=20
1410  0894 70          >         CLR              SET
1411  0895 5C          >         LR    S,A        20=0
1412  0896 28 08 AF    >         PI    DV1        DO 1ST PASS
1413  0899 41          >         LR    A,1        SET
1414  089A 5C          >         LR    S,A        SCRATCH 24=1ST DIGIT
1415  089B 69          >         LISL  1          SAVE
1416  089C 4E          >         LR    A,D        21
1417  089D 51          >         LR    1,A        IN R1
1418  089E 28 09 F7    >         PI    PULLSR     RESTORE (20-21)
1419  08A1 41          >         LR    A,1        SETUP REMAINDER
1420  08A2 5C          >         LR    S,A        IN 20
1421  08A3 90 05       >         BR    DV2        B AROUND ENTRY
1422  08A5 08          > DV1     LR    K,P        SAVE RETURN
1423  08A6 28 0A 00    >         PI    PUSHRT     PUSH RETURN
1424  08A9 20 FF       > DV2     LI    -1         SET
1425  08AB 51          >         LR    1,A        DIGIT COUNTER-1
1426  08AC 41          > DV3     LR    A,1        LOAD AND
1427  08AD 1F          >         INC              INCREMENT
1428  08AE 51          >         LR    1,A        DIGIT
1429  08AF 28 0B 63    >         PI    SUBD       SUBTRACT
1430  08B2 81 F9       >         BP    DV3        B IF NOT LT 0
1431  08B4 28 08 79    >         PI    ADDD       ADJUST FOR PAST ZERO
1432  08B7 6D          >         LISL  5          LOAD LOW
1433  08B8 41          >         LR    A,1        ORDER RESULT INTO
1434  08B9 5E          >         LR    D,A        SCRATCH 25 AND LISL 24
1435  08BA 29 0A 0C    >         JMP   PULLRT     RETURN
1436                   > *
1437                   > *                        COMPARE STACK TOP TO SCRATCH 20-21
1438                   > *                        STACK TOP < 20-21 USE BNC(-)
1439                   > *                        STACK TOP = 20-21 USE BZ
1440                   > *                        STACK TOP > 20-21 USE BZ AND THEN BC(+
1441                   > *                        PULLS VALUE OFF OF STACK
1442  08BD 11          > COMP    LR    H,DC       SAVE DC
1443  08BE 68          > COMX    LISL  0          SET LOW ISAR
1444  08BF 0F          > COMT    LR    DC,Q       GET STACK POINTER
1445  08C0 4D          >         LR    A,I        LOAD HI ORDER BYTE
1446  08C1 8D          >         CM               STACK HI - 20
1447  08C2 94 03       >         BNZ   COM1       B IF NOT EQUAL
1448  08C4 4E          >         LR    A,D        LOAD LOW
1449  08C5 8D          >         CM               STACK LOW-21
1450  08C6 1E          > COM1    LR    J,W        SAVE STATUS
1451  08C7 0F          >         LR    DC,Q       LOAD STACK ADDR
1452  08C8 16          >         LM               TAKE LAST ITEM
1453  08C9 16          >         LM               OFF OF THE STACK
1454  08CA 0E          >         LR    Q,DC       SAVE NEW POINTER
1455  08CB 10          >         LR    DC,H       RESTORE DC
1456  08CC 68          >         LISL  0          RESET ISAR
1457  08CD 1C          >         POP              RETURN
1458                   > *
1459  08CE 16          > SKIP    LM               SKIP THIS BYTE
1460  08CF 11          > CHAR    LR    H,DC       SAVE DC
1461  08D0 16          >         LM               GET NEXT CHAR
1462  08D1 25 20       >         CI    ' '        BLANK ?
1463  08D3 84 FB       >         BZ    CHAR       B IF A BLANK
1464  08D5 54          >         LR    4,A        GET LAST DC
1465  08D6 10          >         LR    DC,H       GET DC0 OF LOADED BYTE
1466  08D7 1C          >         POP              RETURN
1467                   > *
1468                   > *                        FIND NEXT NON BLANK CHAR STARTING AT DC
1469  08D8 11          > IGNBK   LR    H,DC       SAVE TBP
1470  08D9 16          > IGN1    LM               LOAD A CHAR
1471  08DA 25 20       >         CI    ' '        SEE IF A BLANK
1472  08DC 84 FC       >         BZ    IGN1       B IF A BLANK
1473  08DE 54          >         LR    4,A        SAVE CHARACTER
1474  08DF 1C          >         POP              RETURN WITH CHAR IN ACCUM
1475                   > *                        EXCHANGE TOP OF STACK & ISAR
1476                   > *                        R7-8 ALSO SFT TO SCRATCH VALUE
1477  08E0 68          > EXCH    LISL  0          SET ISAR
1478  08E1 0B          >         LR    K,P        SAVE RETURN
1479  08E2 4D          >         LR    A,I        HI
1480  08E3 56          >         LR    6,A        IN R6
1481  08E4 4E          >         LR    A,D        LOW
1482  08E5 57          >         LR    7,A        IN R7
1483  08E6 28 09 F7    >         PI    PULLSR     PUT STACK IN SCRATCH
1484  08E9 60          >         LISU  0          NOW
1485  08EA 6E          >         LISL  6          PUT
1486  08EB 28 09 EA    >         PI    PUSHSR     SCRATCH ON THE STACK
1487  08EE 62          >         LISU  2          RESET
1488  08EF 68          >         LISL  0          ISAR
1489  08F0 0C          >         PK               RETURN
1490                   > *
1491                   > *                        MOVE 20-21 TO 22-23
1492  08F1 68          > MV2021  LISL  0          SET ISAR
1493  08F2 4D          >         LR    A,I        LOAD 20
1494  08F3 6A          >         LISL  2          SET TO 22
1495  08F4 5E          >         LR    D,A        INTO 22
1496  08F5 4D          >         LR    A,I        LOAD 21
1497  08F6 6B          >         LISL  3          SET TO 23
1498  08F7 5D          >         LR    I,A        INTO 23
1499  08F8 68          >         LISL  0          RESET ISAR
1500  08F9 1C          >         POP              RETURN
1501                   > *
1502                   > *                        MOVE DATA FROM HI TO LOW CORE
1503                   > *                        DC1=ADDR+1 OF LAST BYTE STORED
1504  08FA 11          > MVUP    LR    H,DC       SAVE FROM
1505  08FB 0F          >         LR    DC,Q       GET STACK ADDR
1506  08FC 4A          >         LR    A,10       COMPARE
1507  08FD 8D          >         CM               HI BYTE
1508  08FE 94 05       >         BNZ   MV1        B IF NOT EQ
1509  0900 4B          >         LR    A,11       COMPARE LOW
1510  0901 8D          >         CM               BYTE
1511  0902 84 08       >         BZ    MV2        B IF THE END
1512  0904 10          > MV1     LR    DC,H       RESTORE FROM
1513  0905 16          >         LM               LOAD A BYTE
1514  0906 2C          >         XDC              GET TO
1515  0907 17          >         ST               STORE IT
1516  0908 2C          >         XDC              RESTORE FROM
1517  0909 90 F0       >         BR    MVUP       GET NEXT BYTE
1518  090B 1C          > MV2     POP              RETURN
1519                   > *
1520                   > *                        DC0=FROM, DC1=TO, STACK END=TOP
1521                   > *                        THE TOP OF THE STACK IS COMPARED
1522                   > *                        TO THE 'FROM' ADDR AFTER EACH MOVE.
1523                   > *                        MOVE DATA FROM LOW TO HI CORE
1524                   > *                        H WILL HAVE THE ADDR OF THE LAST BYTE
1525                   > *                        THAT WAS MOVED
1526  090C 11          > MVDOWN  LR    H,DC       SAVE DC
1527  090D 0F          >         LR    DC,Q       STACK TOP ADDR
1528  090E 4A          >         LR    A,10       COMPARE
1529  090F 8D          >         CM               1ST BYTE
1530  0910 94 03       >         BNZ   MVD1       B IF NOT EQUAL
1531  0912 4B          >         LR    A,11       COMPARE
1532  0913 8D          >         CM               2ND BYTE
1533  0914 1E          > MVD1    LR    J,W        SAVE STATUS
1534  0915 10          >         LR    DC,H       RESTORE PRESENT FROM
1535  0916 16          >         LM               LOAD A BYTE
1536  0917 2C          >         XDC              GET TO
1537  0918 17          >         ST               STORE
1538  0919 20 FE       >         LI    -2         ADJUST
1539  091B 8E          >         ADC              TO
1540  091C 2C          >         XDC              ADJUST
1541  091D 8E          >         ADC              FROM
1542  091E 1D          >         LR    W,J        RESTORE STATUS
1543  091F 94 EC       >         BNZ   MVDOWN     B IF NOT THE END
1544  0921 1C          >         POP              RETURN
1545                   > *
1546                   > *                        MOVE A STRING FROM BUFF TO THE
1547                   > *                        VARIABLE ADDR. DC0=VAR ADDR
1548                   > *                        TBP IS ON THE STACK
1549  0922 08          > BUFTOV  LR    K,P        SAVE RETURN
1550  0923 70          >         CLR              ZERO ACCUM
1551  0924 1D          >         LR    W,J        GET STATUS
1552  0925 94 08       >         BNZ   BU0        B IF A-Z
1553  0927 16          >         LM               ADJUST ARRAY ADDR
1554  0926 6E          >         LISL  6          SET ISAR TO @ TYPE
1555  0929 28 09 EA    >         PI    PUSHSR     PUT TXTU ON THE STACK
1556  092C 20 FE       >         LI    -2         GET VAR
1557  092E 57          > BU0     LR    7,A        ADJUST
```

```
1558  092F 2C              >          XDC              SAVE VAR ADDR
1559  0930 84 07           >          BZ     BU1       B IF @
1560  0932 2A 13 36        >          DCI    STRLGH    PUT THE
1561  0935 28 09 D5        >          PI     PUSHDC    END ADDR OF
1562  0938 28 09 F6        > BU1      PI     FULL20    A-Z IN 20-21
1563  093B 7D              >          LIS    13        SET (CR) AS
1564  093C 51              >          LR     1,A       AS THE TERMINATOR
1565  093D 2A 13 37        >          DCI    BUFF      GET THE
1566  0940 28 08 CF        >          PI     CHAR      FIRST CHAR
1567  0943 20 48           >          LI     72        SETUP
1568  0945 52              >          LR     2,A       LENGTH COUNTER
1569  0946 16              > BU2      LM               LOAD A CHAR
1570  0947 56              >          LR     6,A       SAVE IT
1571  0948 E1              >          XS     1         (CR) ?
1572  0949 2C              >          XDC              GET VAR ADDR
1573  094A 84 26           >          BZ     BU7       B IF END OF STRING
1574  094C 70              >          CLR              SEE IF
1575  094D C6              >          AS     6         A NUL END
1576  094E 84 22           >          BZ     BU7       WAS USED
1577  0950 28 09 D5        >          PI     PUSHDC    PUT ADDR ON STACK
1578  0953 28 08 BF        >          PI     COMT      TATU-@ OR STRLGH-(A-Z)
1579  0956 45              >          LR     A,5       VAR TYPE
1580  0957 59              >          LR     9,A       INTO R9
1581  0958 92 06           >          BNC    BU5       B IF COMPARE WAS -
1582                       > *                         IF @ WE ARE OK
1583  095A 1D              >          LR     W,J       VAR TYPE
1584  095B 94 12           >          BNZ    BU4       B IF A-Z, AN ERROR
1585  095D 90 04           >          BR     BU6       CONTINUE
1586                       > *                         IF A-Z WE ARE OK
1587  095F 1D              > BU5      LR     W,J       VAR TYPE
1588  0960 84 0D           >          BZ     BU4       B IF @, AN ERROR
1589  0962 46              > BU6      LR     A,6       RESTORE CHAR
1590  0963 13              >          SL     1         MAKE SURE
1591  0964 12              >          SR     1         NO FALSE END
1592  0965 58              >          LR     8,A       SAVE THE CHAR
1593  0966 11              >          LR     H,DC      AND THE ADDR
1594  0967 17              >          ST               STORE CHAR
1595  0968 47              >          LR     A,7       ADJUST
1596  0969 8E              >          ADC              VAR ADDR
1597  096A 2C              >          XDC              RESTORE FROM
1598  096B 32              >          DS     2         CHECK STRING LENGTH
1599  096C 94 D9           >          BNZ    BU2       B IF MORE ROOM
1600  096E 29 02 5E        > BU4      JMP    AHOW      NO MORE ROOM
1601  0971 48              > BU7      LR     A,8       RESTORE LAST CHAR
1602  0972 10              >          LR     DC,H      AND ITS ADDR
1603  0973 22 80           >          OI     $80       OF STRING
1604  0975 17              >          ST               FLAG
1605  0976 42              >          LR     A,2       SEE IF
1606  0977 12              >          SR     1         THE STRING
1607  0978 13              >          SL     1         LENGTH IS
1608  0979 E2              >          XS     2         ODD OR EVEN
1609  097A 84 05           >          BZ     BU8       B IF EVEN
1610  097C 47              >          LR     A,7       ADJUST
1611  097D 8E              >          ADC              VAR ADDR
1612  097E 48              >          LR     A,8       LOAD THE LAST CHAR
1613  097F 17              >          ST               ODD,FILL THE WORD
1614  0980 28 09 84        > BU8      PI     SLEN      GET THE LENGTH
1615  0983 0C              >          PK               RETURN
1616                       > *
1617                       > *                         R2 HAS A DECREMENTED COUNTER
1618                       > *                         COMPLEMENT IT AND ADD START LE
1619  0984 42              > SLEN     LR     A,2       MAKE
1620  0985 18              >          COM              COUNTER
1621  0986 1F              >          INC              NEGATIVE
1622  0987 24 48           >          AI     72        ADD ORIGINAL LENGTH
1623  0989 2A 13 36        >          DCI    STRLGH    STORE
1624  098C 17              >          ST               LENGTH
1625  098D 1C              >          POP              RETURN
1626                       > *
1627                       > *                         SCRATCH 36-37 LOPVAR GOES ON
1628  098E 2C              > SAVE     XDC              SAVE TRP
1629  098F 08              >          LR     K,P       SAVE RETURN
1630  0990 2A 13 80        >          DCI    SKEND     STACK LIMIT ADDR
1631  0993 28 09 D5        >          PI     PUSHDC    SAVE END OF STACK
1632  0996 61              >          LISL   1         SET ISAR
1633  0997 6E              >          LISL   6         TO @ REGS
1634  0998 28 08 BF        >          PI     COMT      SKIP LISL 0
1635  099B 91 05           >          BM     SAV1      B IF ROOM LEFT
1636  099D 2C              >          XDC              GET TRP
1637  099E 29 02 11        >          JMP    GSORRY    PROCESS ERROR
1638  09A1 63              > SAV1     LISL   3         SETUP ISAR
1639  09A2 6E              >          LISL   6         SET TO LOPVR
1640  09A3 4D              >          LR     A,I       LOOP VARIABLE ADDR
1641  09A4 EC              >          XS     S         XOR AND ADD
1642  09A5 CE              >          AS     D         TO CHECK FOR ZERO
1643  09A6 84 11           >          BZ     SAV3      B IF ZERO
1644  09A8 0F              >          LR     DC,Q      GET STACK ADDR

1645  09A9 20 F8           >          LI     -8        MAKE ROOM
1646  09AB 8C              >          ADC              FOR 4 ITEMS ON THE STACK
1647  09AC 0E              >          LR     Q,DC      SAVE STACK
1648  09AD 64              >          LISU   4         SAVE THE REST
1649  09AE 68              >          LISL   8         OF THE VARIABLES
1650  09AF 78              >          LIS    8         SETUP
1651  09B0 51              >          LR     1,A       LOOP
1652  09B1 4D              > SAV2     LR     A,I       SAVE
1653  09B2 17              >          ST               FOR VARIABLES
1654  09B3 31              >          DS     1         DEC COUNTER
1655  09B4 94 FC           >          BNZ    SAV2      LOOP IF NOT DONE
1656  09B6 63              >          LISU   3         PUT
1657  09B7 6E              >          LISL   6         LOPVAR ON
1658  09B8 28 09 EA        > SAV3     PI     PUSHSR    ON THE STACK TOP
1659  09BB 2C              >          XDC              RESTORE TRP
1660  09BC 62              >          LISU   2         RESTORE ISAR
1661  09BD 0C              >          PK               RETURN
1662                       > *
1663                       > *                         RESTORE FOR VARIABLES
1664                       > *                         LOPVAR IS ON THE STACK TOP
1665  09BE 63              > RESTOR   LISU   3         SET
1666  09BF 6E              >          LISL   6         ISAR
1667  09C0 0F              >          LR     DC,Q      GET STACK POINTER
1668  09C1 16              >          LM               RESTORE
1669  09C2 5D              >          LR     I,A       LOPVAR
1670  09C3 16              >          LM               FROM
1671  09C4 5E              >          LR     D,A       STACK TOP
1672  09C5 EC              >          XS     S         XOR AND
1673  09C6 CC              >          AS     S         ADD TO SEE IF ZERO
1674  09C7 84 0A           >          BZ     RES2      B IF ZERO
1675  09C9 64              > REFOR    LISU   4         SET
1676  09CA 68              >          LISL   8         ISAR
1677  09CB 78              >          LIS    8         SETUP
1678  09CC 51              >          LR     1,A       LOOP
1679  09CD 16              > RES1     LM               LOAD
1680  09CE 5D              >          LR     I,A       LOOP VARIABLES
1681  09CF 31              >          DS     1         DEC COUNTER
1682  09D0 94 FC           >          BNZ    RES1      LOOP IF NOT DONE
1683  09D2 0E              > RES2     LR     Q,DC      RESTORE STACK
1684  09D3 62              >          LISU   2         RESET ISAR
1685  09D4 1C              >          POP              RETURN
1686                       > *
1687                       > *                         PUSH DC ON TO STACK
1688  09D5 11              > PUSHDC   LR     H,DC      SAVE CURRENT DC
1689  09D6 0F              >          LR     DC,Q      STACK POINTER
1690  09D7 20 FE           >          LI     -2        THE STACK POINTER
1691  09D9 8E              >          ADC              2 BYTES
1692  09DA 0E              >          LR     Q,DC      SAVE STACK POINTER
1693  09DB 4A              >          LR     A,10      SAVE
1694  09DC 17              >          ST               HI
1695  09DD 4B              >          LR     A,11      SAVE
1696  09DE 17              >          ST               LOW
1697  09DF 10              >          LR     DC,H      RESTORE DC
1698  09E0 1C              >          POP              RETURN
1699                       > *
1700                       > *                         PULL DC FROM STACK
1701  09E1 0F              > PULLDC   LR     DC,Q      BACK UP
1702  09E2 16              >          LM               LOAD
1703  09E3 5A              >          LR     10,A      HI
1704  09E4 16              >          LM               LOW
1705  09E5 5B              >          LR     11,A      LOAD
1706  09E6 0E              >          LR     Q,DC      SAVE NEW POINTER
1707  09E7 10              >          LR     DC,H      INTO DC
1708  09E8 1C              >          POP              RETURN
1709                       > *
1710                       > *                         SAVE 2 SCRATCH BYTES ON THE STACK
1711                       > *                         LISL REMAINS THE SAME
1712  09E9 68              > PUSH20   LISL   8         SPECIAL ENTRY FOR 20-21
1713  09EA 11              > PUSHSR   LR     H,DC      SAVE DC
1714  09EB 0F              >          LR     DC,Q      GET STACK POINTER
1715  09EC 20 FE           >          LI     -2        BACKUP TWO
1716  09EE 8F              >          ADC              BYTES
1717  09EF 0E              >          LR     Q,DC      SAVE STACK POINTER
1718  09F0 4D              >          LR     A,I       LOAD 1ST BYTE
1719  09F1 17              >          ST               SAVE IT
1720  09F2 4E              >          LR     A,D       LOAD 2ND BYTE
1721  09F3 17              >          ST               SAVE
1722  09F4 10              >          LR     DC,H      RESTORE DC
1723  09F5 1C              >          POP              RETURN
1724                       > *
1725                       > *                         PULL 2 BYTES FROM STACK INTO SCRATCH
1726                       > *                         LISL REMAINS THE SAME
1727  09F6 68              > PULL20   LISL   8         SPECIAL ENTRY FOR 20-21
1728  09F7 11              > PULLSR   LR     H,DC      SAVE DC
1729  09F8 0F              >          LR     DC,Q      LOAD STACK POINTER
1730  09F9 16              >          LM               LOAD
1731  09FA 5D              >          LR     I,A       INTO SCRATCH
```

```
1732  09FB 16          >          LM              LOAD 2ND
1733  09FC 5E          >          LR    D,A       INTO SCRATCH
1734  09FD 0E          >          LR    Q,DC      SAVE POINTER
1735  09FE 10          >          LR    DC,H      RESTORE DC
1736  09FF 1C          >          POP             RETURN
1737                   > *
1738                   > *                       SAVE RETURN (KU-KL) ON STACK
1739                   > *                       CALLING FORMAT IS
1740                   > *         LR    K,P
1741                   > *         PI    PUSHRT
1742  0A00 11          > PUSHRT    LR    H,DC    SAVE DC
1743  0A01 0F          >          LR    DC,Q      GET STACK POINTER
1744  0A02 20 FE       >          LI    -2        BACK UP 2
1745  0A04 8E          >          ADC             BYTES TO LAST ENTRY
1746  0A05 0E          >          LR    Q,DC      SAVE POINTER
1747  0A06 00          >          LR    A,KU      STORE
1748  0A07 17          >          ST              HIGH
1749  0A08 01          >          LR    A,KL      STORE
1750  0A09 17          >          ST              LOW
1751  0A0A 1C          >          LR    DC,H      RESTORE DC
1752  0A0B 1C          >          POP             RETURN
1753                   > *
1754                   > *                       PULL RETURN FROM STACK AND RETURN
1755                   > *                       TO THE ADDR THAT WAS PULLED FROM THE
1756                   > *                       STACK WITH A PK
1757  0A0C 11          > PULLRT    LR    H,DC    SAVE DC
1758  0A0D 0F          >          LR    DC,Q      GET STACK POINTER
1759  0A0E 16          >          LM              LOAD HI
1760  0A0F 04          >          LR    KU,A      ORDER INTO KU
1761  0A10 16          >          LM              LOAD LOW
1762  0A11 05          >          LR    KL,A      ORDER INTO KL
1763  0A12 0E          >          LR    Q,DC      SAVE UPDATED POINTER
1764  0A13 10          >          LR    DC,H      RESTORE DC
1765  0A14 0C          >          PK              RETURN TO PULLED RETURN ADDR
1766                   > *
1767                   > *                       PULLS RETURN OFF STACK INTO K
1768  0A15 11          > POPRT     LR    H,DC    SAVE DC
1769  0A16 0F          >          LR    DC,Q      GET STACK POINTER
1770  0A17 16          >          LM              LOAD
1771  0A18 04          >          LR    KU,A      THE
1772  0A19 16          >          LM              RETURN
1773  0A1A 05          >          LR    KL,A      ADDR
1774  0A1B 0E          >          LR    Q,DC      RESTORE IT
1775  0A1C 10          >          LR    DC,H      RESTORE DC
1776  0A1D 1C          >          POP             RETURN
1777                   > *
1778                   > *                       CALL A.L. ROUTINE
1779  0A1E 28 05 FD    > USR       PI    EXPR      GET ADDR
1780  0A21 28 09 D5    >          PI    PUSHDC    SAVE TBP
1781                   > *                       CALLED ROUTINE SHOULD SAVE RETURN
1782  0A24 4D          >          LR    A,I       LOAD HI CALL BYTE
1783  0A25 04          >          LR    KU,A      INTO KU
1784  0A26 4E          >          LR    A,D       LOW CALL BYTE
1785  0A27 05          >          LR    KL,A      INTO KL
1786  0A28 0C          >          PK              CALL THE ROUTINE
1787  0A29 28 09 E1    >          PI    PULLDC    RESTORE TBP
1788  0A2C 29 03 19    >          JMP   FIN       FINISH UP
1789                   > *
1790                   > *                       PUT A VALUE IN MEMORY
1791                   > *                       POKE ADDR,VALUE
1792  0A2F 28 05 FD    > POKE      PI    EXPR      GET POKE ADDR
1793  0A32 28 09 E9    >          PI    PUSH20    SAVE IT
1794  0A35 28 08 D8    >          PI    IGNBK     GET NEXT CHAR
1795  0A38 25 2C       >          CI    ',        COMMA ?
1796  0A3A 94 0F       >          BNZ   POK1      B IF NOT A COMMA
1797  0A3C 28 05 FD    >          PI    EXPR      GET NEXT EXPR
1798  0A3F 2C          >          XDC             SAVE TBP
1799  0A40 28 09 E1    >          PI    PULLDC    GET POKE ADDR
1800  0A43 69          >          LISL  1         STORE
1801  0A44 4E          >          LR    A,D       THE
1802  0A45 17          >          ST              VALUE (2ND BYTE OF THE WORD)
1803  0A46 2C          >          XDC             RESTORE TBP
1804  0A47 29 03 19    >          JMP   FIN       FINISH UP
1805  0A4A 29 07 B2    > POK1      JMP   GWHAT     ERROR
1806                   > *
1807                   > *                       PROCESS 'LITERAL'
1808                   > *                       PUT INTO BUFF AND 1ST 2 CHARS IN
1809  0A4D 20 27       > APOST     LI    '       SETUP
1810  0A4F 90 03       >          BR    AQ1       THE
1811  0A51 20 22       > QUOTE     LI    "       LITERAL
1812  0A53 51          > AQ1       LR    1,A       END CHAR
1813  0A54 2C          >          XDC             SAVE TBP
1814  0A55 2A 13 37    >          DCI   BUFF      GET TARGET
1815  0A58 20 48       >          LI    72        GET
1816  0A5A 52          >          LR    2,A       MAX LENGTH
1817  0A5B 2C          > AQ2       XDC             GET TBP
1818  0A5C 16          >          LM              LOAD A CHAR
1819  0A5D 2C          >          XDC             GET PRESENT BUFF ADDR
1820  0A5E 11          >          LR    H,DC      SAVE PRESENT BUFF ADDR
1821  0A5F 17          >          ST              STORE CHAR
1822  0A60 E1          >          XS    1         LAST ONE?
1823  0A61 84 04       >          BZ    AQ3       B IF IT IS
1824  0A63 32          >          DS    2         ROOM LEFT ?
1825  0A64 90 F6       >          BR    AQ2       CONTINUE
1826  0A66 10          > AQ3       LR    DC,H      A ZERO TERMINATOR
1827  0A67 17          >          ST              TO MARK END OF STRING
1828  0A68 2A 13 37    >          DCI   BUFF      PUT THE
1829  0A6B 68          >          LISL  0         FIRST
1830  0A6C 16          >          LM              AND
1831  0A6D 54          >          LR    4,A       SECOND
1832  0A6E 5D          >          LR    I,A       CHARS
1833  0A6F 16          >          LM              IN
1834  0A70 5C          > AQ4       LR    S,A       20-21
1835  0A71 E1          >          XS    1         WAS THERE JUST
1836  0A72 44          >          LR    A,4       ONE CHAR ?
1837  0A73 84 FC       >          BZ    AQ4       IF ONLY 1 REPEAT IT IN 21
1838  0A75 68          >          LISL  0         RESET ISAR
1839  0A76 28 09 84    >          PI    SLEN      SETUP STRING LENGTH
1840  0A79 71          >          LIS   1         SET STATUS
1841  0A7A 55          >          LR    5,A       TO STRING
1842  0A7B 2C          >          XDC             RESTORE TBP
1843  0A7C 28 08 CF    >          PI    CHAR      GET NEXT CHAR
1844  0A7F 29 0A 0C    >          JMP   PULLRT    RETURN
1845                   > *
1846                   > *                       GET STRING LENGTH OF VARIABLE
1847                   > *                       IF THE ARG=0 GET LAST LENGTH
1848  0A82 28 07 99    > LEN       PI    PARN      EVALUATE ARG
1849  0A85 2C          >          XDC             SAVE TBP, GET VAR ADDR
1850  0A86 70          >          CLR             ZERO HI
1851  0A87 55          >          LR    5,A       ZERO STRING FLAG
1852  0A88 5D          >          LR    I,A       BYTE
1853  0A89 2A 13 36    >          DCI   STRLGH    PUT THE
1854  0A8C 16          >          LM              STRING LENGTH
1855  0A8D 5E          >          LR    D,A       INTO 20-21
1856  0A8E 2C          >          XDC             RESTORE TBP
1857  0A8F 29 0A 0C    >          JMP   PULLRT    RETURN
1858                   > *
1859                   > *                       ABSOLUTE VALUE
1860  0A92 28 07 99    > ABS       PI    PARN      ABS(EXPR)
1861  0A95 28 08 4C    >          PI    CHKSGN    CHECK SIGN
1862  0A98 4C          >          LR    A,S       SEE IF
1863  0A99 25 80       >          CI    $80       -32768
1864  0A9B 84 6C       >          BZ    JMPHOW    B IF IT IS
1865  0A9D 29 0A 0C    >          JMP   PULLRT    RETURN
1866                   > *
1867                   > *                       RANDOM NUMBER
1868  0AA0 28 07 99    > RND       PI    PARN      RND(EXPR)
1869  0AA3 4C          >          LR    A,S       EXPR
1870  0AA4 CC          >          AS    S         MUST BE PLUS
1871  0AA5 91 62       >          BM    JMPHOW    B IF -
1872  0AA7 4D          >          LR    A,I       SEE IF ZERO
1873  0AA8 EC          >          XS    S         BY XORING
1874  0AA9 CE          >          AS    D         AND ADDING
1875  0AAA 84 5D       >          BZ    JMPHOW    B IF ZERO
1876  0AAC 28 09 EA    >          PI    PUSHSR    SAVE EXPR
1877  0AAF 2C          >          XDC             AND TBP
1878  0AB0 2A 0B FE    >          DCI   RANPT     GET LAST ADDR
1879  0AB3 28 09 D5    >          PI    PUSHDC    ON THE STACK
1880  0AB6 16          >          LM              LOAD CONTENTS
1881  0AB7 5A          >          LR    10,A      SAVE HI ADDR
1882  0AB8 5D          >          LR    I,A       OF
1883  0AB9 16          >          LM              RANDOM MEMORY
1884  0ABA 5B          >          LR    11,A      SAVE LOW ADDR
1885  0ABB 5E          >          LR    D,A       INTO 20-21
1886  0ABC 28 08 BE    >          PI    COMX      SKIP LR H,DC
1887  0ABF 82 04       >          BC    RN1       B IF NOT LAST ADDR
1888  0AC1 2A 00 03    >          DCI   START     WRAP AROUND ADDR
1889  0AC4 16          > RN1       LM              PUT
1890  0AC5 5D          >          LR    I,A       CONTENT OF
1891  0AC6 16          >          LM              ADDR IN RANPNT
1892  0AC7 5D          >          LR    I,A       INTO 20-21
1893  0AC8 11          >          LR    H,DC      NEW RANPNT ADDR ADDR
1894  0AC9 2A 0B FE    >          DCI   RANPNT    GET TARGET
1895  0ACC 4A          >          LR    A,10      STORE
1896  0ACD 17          >          ST              IT
1897  0ACE 4B          >          LR    A,11      IN
1898  0ACF 17          >          ST              RANPNT
1899  0AD0 28 09 F7    >          PI    PULLSR    ARG INTO 22-23
1900  0AD3 28 08 4C    >          PI    CHKSGN    CHECK SIGN OF RANDOM NUMBER
1901  0AD6 28 08 8B    >          PI    DIVIDE    RANNUM/INPUT #
1902  0AD9 69          >          LISL  1         ADD
1903  0ADA 4C          >          LR    A,S       ONE
1904  0ADB 1F          >          INC             TO  THE
1905  0ADC 5E          >          LR    D,A       REMAINDER
```

```
1906  0ADD 4C          >          LR    A,S          ADD IN
1907  0ADE 19          >          LNK                ANY
1908  0ADF 5C          >          LR    S,A          CARRY
1909  0AE0 2C          >          XDC                RESTORE TBP
1910  0AE1 29 0A 0C    >          JMP   PULLRT       RETURN
1911                   > *
1912                   > *                          VAL=PEEK(ADDR)
1913                   > *                          SET VAL TO THE BYTE AT ADDR
1914  0AE4 26 07 99    > PEEK     PI    PARN         GO GET ADDR IN 20-21
1915  0AE7 2C          >          XDC                SAVE TBP
1916  0AE8 4D          >          LR    A,I          LOAD
1917  0AE9 5A          >          LR    10,A         PEEK
1918  0AEA 4C          >          LR    A,S          ADDR
1919  0AEB 5B          >          LR    11,A         INTO
1920  0AEC 19          >          LR    DC,H         DC
1921  0AED 16          >          LM                 LOAD PYTE
1922  0AEE 5E          >          LR    D,A          INTO 21
1923  0AEF 70          >          CLR                ZERO
1924  0AF0 5C          >          LR    S,A          20
1925  0AF1 2C          >          XDC                RESTORE TBP
1926  0AF2 29 0A 0C    >          JMP   PULLRT       RETURN
1927                   > *
1928                   > *                          TAB(X) SPACE OVER TO COLUMN X
1929  0AF5 28 07 99    > TAB      PI    PARN         EVALUATE THE EXPRESSION
1930  0AF8 4D          >          LR    A,I          SEE IF
1931  0AF9 EC          >          XS    S            ZERO BY XOR
1932  0AFA CC          >          AS    S            AND ADDING
1933  0AFB 84 0C       >          BZ    JMPHOW       B IF ZERO AN ERROR
1934  0AFD 4E          >          LR    A,D          LOAD THE NUMBER OF BLANKS TO PRI
1935  0AFE 58          >          LR    8,A          INTO RP
1936  0AFF 28 01 86    > TA1      PI    PRLK         PRINT A BLANK
1937  0B02 38          >          DS    8            DEC LOOP COUNTER
1938  0B03 94 FB       >          BNZ   TA1          B IF MORE
1939  0B05 70          >          CLR                SETUP DUMMY
1940  0B06 90 09       >          BR    CH1          STRING FOR PRINT
1941  0B08 29 02 5B    > JMPHOW   JMP   GHOW         ERROR
1942                   > *
1943                   > *                          CHR(X) OUTPUT ASCII CODE X
1944  0B0B 28 05 FD    > CHR      PI    EXPR         GET ASCII
1945  0B0E 69          >          LISL  1            CODE
1946  0B0F 4E          >          LR    A,D          INTO
1947  0B10 11          > CH1      LR    H,DC         SAVE TBP
1948  0B11 2A 13 37    >          DCI   BUFF         BUFF
1949  0B14 17          >          ST                 AND
1950  0B15 70          >          CLR                MARK
1951  0B16 17          >          ST                 END
1952  0B17 1F          >          INC                AND
1953  0B18 55          >          LR    5,A          SET STRING FLAG
1954  0B19 10          >          LR    DC,H         RESTORE TBP
1955  0B1A 29 0A 0C    >          JMP   PULLRT       RETURN
1956                   > *
1957  0B1D             > H8000    EQU   $8000
1958  0B1D             > TAB1     EQU   *            DIRECT COMMANDS
1959  0B1D 4C 49 53 54 >          DC    *'LIST'*
1960  0B21 84 00       >          DA    LIST+H8000
1961  0B23 4E 45 57    >          DC    *'NEW'*
1962  0B26 84 48       >          DA    NEW+H8000
1963  0B28 52 55*4E    >          DC    *'RUN'*
1964  0B2B 85 CB       >          DA    RUN+H8000
1965  0B2D 4D 4F 4E    >          DC    *'MON'*
1966  0B30 8B FB       >          DA    MON+H8000
1967  0B32             > TAB2     EQU   *            DIRECT/STATEMENT
1968  0B32 4C 45 54    >          DC    *'LET'*
1969  0B35 83 2A       >          DA    LET+H8000
1970  0B37 49 46       >          DC    *'IF'*
1971  0B39 84 5E       >          DA    IF+H8000
1972  0B3B 47 4F 54 4F >          DC    *'GOTO'*
1973  0B3F 85 B5       >          DA    GOTO+H8000
1974  0B41 46 4F 52    >          DC    *'FOR'*
1975  0B44 84 74       >          DA    FOR+H8000
1976  0B46 4E 45 58 54 >          DC    *'NEXT'*
1977  0B4A 84 E5       >          DA    NEXT+H8000
1978  0B4C 47 4F 53 55 >          DC    *'GOSUB'*
1979  0B50 42          >
1980  0B51 83 91       >          DA    GOSUB+H8000
1981  0B53 52 45 54 55 >          DC    *'RETURN'*
1982  0B57 52 4E       >
1983  0B59 83 B4       >          DA    RETURN+H8000
1984  0B5B 50 52 49 4E >          DC    *'PRINT'*
1985  0B5F 54          >
1986  0B60 83 35       >          DA    PRINT+H8000
1987  0B62 49 4E 50 55 >          DC    *'INPUT'*
1988  0B66 54          >
1989  0B67 82 9F       >          DA    INPUT+H8000
1990  0B69 54 48 45 4E >          DC    *'THEN'*
1991  0B6D 85 B5       >          DA    GOTO+H8000   PROCESS LIKE GOTO
1992  0B6F 52 45 4D    >          DC    *'REM'*
```

```
1993  0B72 84 69       >          DA    REM+H8000
1994  0B74 52 45 41 44 >          DC    *'READ'*
1995  0B78 85 5C       >          DA    READ+H8000
1996  0B7A 44 41 54 41 >          DC    *'DATA'*
1997  0B7E 85 98       >          DA    DATA+H8000
1998  0B80 55 53 52    >          DC    *'USR'*      USER CALLED ROUTINE
1999  0B83 8A 1E       >          DA    USR+H8000
2000  0B85 50 4F 4B 45 >          DC    *'POKE'*     POKE X,Y
2001  0B89 8A 2F       >          DA    POKE+H8000
2002  0B8B 52 45 53 54 >          DC    *'RESTORE'*
2003  0B8F 4F 52 45    >
2004  0B92 85 AD       >          DA    RESTORE+H8000
2005  0B94 53 54 4F 50 >          DC    *'STOP'*
2006  0B98 84 58       >          DA    STOP+H8000
2007  0B9A 83 1F       >          DA    DEFLT+H8000
2008  0B9C             > TAB4     EQU   *            FUNCTIONS
2009  0B9C 52 4E 44    >          DC    *'RND'*      RND(X)
2010  0B9F 8A A0       >          DA    RND+H8000
2011  0BA1 41 42 53    >          DC    *'ABS'*      ABS(X)
2012  0BA4 8A 92       >          DA    ABS+H8000
2013  0BA6 50 45 45 4B >          DC    *'PEEK'*     PEEK IN TO CORE
2014  0BAA 8A E4       >          DA    PEEK+H8000
2015  0BAC 4C 45 4E    >          DC    *'LEN'*      GET THE STRING LENGTH
2016  0BAF 8A 82       >          DA    LEN+H8000
2017  0BB1 27          >          DC    #'           *'LITERAL*
2018  0BB2 8A 4D       >          DA    APOST+H8000
2019  0BB4 22          >          DC    #"           "LITERAL"
2020  0BB5 8A 51       >          DA    QUOTE+H8000
2021  0BB7 54 41 42    >          DC    *'TAB'*      TAB(X)
2022  0BBA 8A F5       >          DA    TAB+H8000
2023  0BBC 43 48 52    >          DC    *'CHR'*      CHR(X)
2024  0BBF 8B 0B       >          DA    CHR+H8000
2025  0BC1 87 45       >          DA    XP40+H8000
2026  0BC3             > TAB5     EQU   *            *'TO' IN *'FOR'
2027  0BC3 54 4F       >          DC    *'TO'
2028  0BC5 84 8E       >          DA    FR1+H8000
2029  0BC7 87 B2       >          DA    GWHAT+H8000
2030  0BC9             > TAB6     EQU   *            *'STEP' IN **'FOR'
2031  0BC9 53 54 45 50 >          DC    *'STEP'
2032  0BCD 84 9A       >          DA    FR2+H8000
2033  0BCF 84 9F       >          DA    FR3+H8000
2034  0BD1             > TAB8     EQU   *            RELATIONAL OPERATORS
2035  0BD1 3E 3D       >          DC    *'>='        GREATER THAN OR EQUAL TO
2036  0BD3 86 0E       >          DA    XP11+H8000
2037  0BD5 23          >          DC    *'#'         NOT EQUAL TO
2038  0BD6 86 15       >          DA    XP12+H8000
2039  0BD8 3E          >          DC    *'>'         GREATER THAN
2040  0BD9 86 1C       >          DA    XP13+H8000
2041  0BDB 3D          >          DC    *'='         EQUAL
2042  0BDC 86 2E       >          DA    XP15+H8000
2043  0BDE 3C 3D       >          DC    *'<='        LESS THAN OR EQUAL TO
2044  0BE0 86 25       >          DA    XP14+H8000
2045  0BE2 3C          >          DC    *'<'         LESS THAN
2046  0BE3 86 35       >          DA    XP16+H8000
2047  0BE5 86 3E       >          DA    XP17+H8000
2048  0BE7 52 45 41 44 > PROMPT   DC    *'READY'*
2049  0BEB 59          >
2050  0BEC 0D          >          DC    $0D
2051                   > *
2052                   > *                          IO JUMP TABLE
2053  0BED 41          > TTYO     LR    A,1          OUTPUT BYTE IN ACCUM
2054  0BEE 25 0D       >          CI    $0D          CR ?
2055  0BF0 84 07       >          BZ    TTCR         B IF YES
2056  0BF2 29 83 E5    > TTY1     JMP   $83E5        OUTPUT IT
2057  0BF5 29 83 AD    > TTYI     JMP   $83AD
2058  0BF8 29 83 D6    > TTCR     JMP   $83D6
2059  0BFB 29 80 80    > MON      JMP   $8080
2060  0BFE 00 00       > RANPNT   DA    $0000        RANDOM NUMBER POINTER
2061  0C00             > TXTB     EQU   *            BEGIN OF TEXT AREA
2062  1300             >          ORG   $1300        CHANGE IN MULTIPLES OF 256 BYTES
2063  1300             > TXTE     EQU   *            END OF TEXT AREA
2064  1300             > VARBGN   RS    54           @(0) LOCATION
2065  1336             > STRLGH   RS    1            STRING LENGTH
2066  1337             > BUFF     RS    73           BUFFER FOR 72 INPUT BYTES
2067  1380             > SKEND    RS    128          END OF STACK
2068  1400             > STACK    EQU   *            BEGIN OF STACK
2069  1400             >          END
```

PAGE LIST FOR THE APPLE

BY ROBERT D. DIAZ
2849 W. 235th St. #3
Torrance, Calif. 90505

The Apple Computer prints on the TV screen at a rate of about 1,000 characters per second. However, most of the Apple users would prefer to read a listing at a slower rate. This article presents a solution to the dilemma: the Page List. The Page List (or listings by page) enables the Apple to list 20 lines of text, stop, and wait for a command from the keyboard to continue or to quit listing.

When the command "CALL 770" is entered into the computer, every twentieth line printed on the TV screen causes the Apple to stop until any key is pressed. The following are the commands used with Page List:

CALL 770	(Turn on Page List)
LIST	(If Page List was turned on, then the computer lists 20 lines and stops)
ESC	(List one additional line)
CTRL C	(Jumps out of the list mode and back into BASIC)
All other keys except Reset	(Lists 20 more lines)
CALL 815	(Turn off Page List)

This program is very flexible. It will function with either Integer BASIC, Floating-Point BASIC, with or without DOS, or on any size memory. It also will function with other software such as Microproducts Assembler, with the exception that hitting a Control C will send the program to Monitor Mode rather than returning it to the assembler.

Loading Page List Into A System

In order to load the Page List into an Apple that does not have a disk drive, first push Reset, then type the program as shown in Figure 1.

```
*300.388

0300- 00 00 A2 04 B5 35 5D 39
0308- 03 D0 06 CA D0 F6 4C 20
0310- 03 A2 04 B5 35 9D 2A 03
0318- BD 39 03 95 35 CA D0 F3
0320- A9 28 8D 00 03 A9 15 8D
0328- 01 03 60 F0 FD 1B FD A2
0330- 04 BD 2A 03 95 35 CA D0
0338- F8 60 3E 03 1B FD 48 49
0340- 8D F0 05 CE 00 03 D0 0A
0348- A9 28 8D 00 03 CE 01 03
0350- F0 04 68 4C F0 FD A9 15
0358- 8D 01 03 AD 00 C0 10 FB
0360- 48 A9 00 8D 10 C0 68 C9
0368- 83 F0 0B C9 9B D0 E3 A9
0370- 01 8D 01 03 D0 DC A5 33
0378- C9 BE D0 03 4C 03 E0 C9
0380- DD D0 03 4C 00 00 4C 69
0388- FF
*
```

Figure 1.

Remember to type a colon and not a dash , then save the program onto tape. The program can be saved onto tape by typing "300.388W". Once the program is saved onto tape, it may be loaded into the Apple by pushing Reset and then typing "300.388R".

If the system has a disk drive, the program can be saved onto a disk instead of tape. The procedure is as follows:

a) Boot up DOS
b) Type "POKE 72,0"
c) Type "CALL –151" (This gets you into Monitor without killing DOS)
d) Type in the program shown in Figure 1
e) Use Control C or Control B to get back into BASIC
f) Save the program on to disk by typing "BSAVE PAGE LIST, A$300,L$89".

The program can be loaded into memory by typing "BLOAD PAGE LIST, A$300".

Some important points to remember about Page List are:
(1) When Page List is turned on, DOS is turned off.
(2) The Page List stays turned on until you turn it off.
(3) If Page List is left on and you run a program, the program will stop when 20 lines of text have been printed.

How It Works

Before the Apple prints anything, it first looks at locations $36 and $37 to find out where to go next. (The dollar sign indicates that the number is in hexidecimal). On a non-disk system, the computer would go to location $FDFO (the output routine in Monitor). When the Apple wishes to input information, it looks at locations $37 and $38 to determine where to go next. When the command "CALL 770" is entered, the Initiation Routine (location $302) first checks to see if Page List is already turned on. If it has not been turned on, the data at locations $36-$39 is moved to locations $32B-$32E, and the data at locations $33A-$33D is moved to locations $36-$39. Counter 1 (location $300) is set to $28 and Counter 2 (location $301) is set to $15.

When Page List is activated, all output is routed first to the Page List routine (location $33E). There are two counters in Page List that keep track of how much has been printed. Counter 1 counts the number of characters on a line and Counter 2 counts the number of lines printed.

The flow chart, (Figure 2), shows how the counters interact and how the Page List routine works.

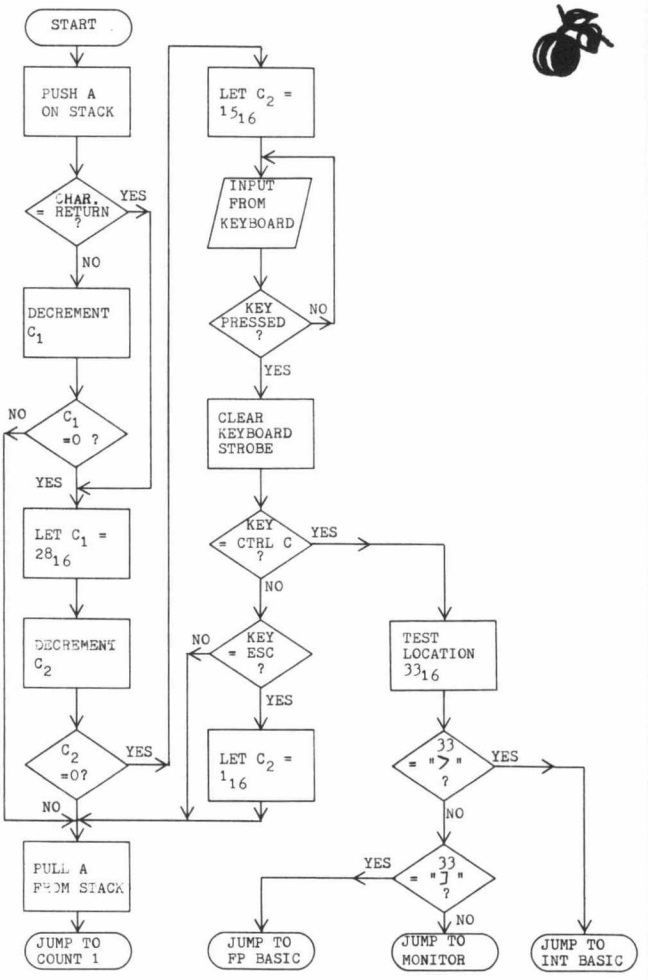


Figure 2. Flowchart to Page-List routine.
C_1 = Counter 1
C_2 = Counter 2

When "CALL 815" is entered, the Restore routine (location $32F) moves the data from locations $32B-$32E back to locations $36-$39.

Program Relocation

One of the concepts connected with Murphy's Law says, "No matter where you put it, that was the wrong place." While Page List and DOS do not conflict with each other as long as DOS is not rebooted, Page List and the Programmer's Aid #1 HIRES Routines do conflict with each other. I'm sure that there are a lot of other programs out there that conflict with Page List in its present location.

The easiest way to relocate the program is to set all Data Bytes, locations $300-$301, $32B-$32E, and $33A-$33D to $00 and relocate the program using Apple's Programmer's Aid #1. You will find relocation much easier if you tell the computer that all of the program is in Machine Code and none of it is Data. Once relocated, look at the program listing using Apple's Disassembler. The part you are looking for is the section with four "BRK's" followed by "PHA." The low order byte of the address is placed in the location of the first of the four "BRK's", and the high order byte is placed in the location of the second "BRK". Place $1B in the location of the third "BRK", and $FD in the final "BRK". The final step is figuring out the new Basic Calls to turn the program on and off.

If a Programmer's Aid is not available, then program relocation will have to be done by hand. The listing in Figure 3 should be of some help in doing the relocation manually.

```
300LLLL
0300-   00          BRK
0301-   00          BRK
0302-   A2 04       LDX   #$04
0304-   B5 35       LDA   $35,X
0306-   5D 39 03    EOR   $0339,X
0309-   D0 06       BNE   $0311
030B-   CA          DEX
030C-   D0 F6       BNE   $0304
030E-   4C 20 03    JMP   $0320
0311-   A2 04       LDX   #$04
0313-   B5 35       LDA   $35,X
0315-   9D 2A 03    STA   $032A,X
0318-   BD 39 03    LDA   $0339,X
031B-   95 35       STA   $35,X
031D-   CA          DEX
031E-   D0 F3       BNE   $0313
0320-   A9 28       LDA   #$28
0322-   8D 00 03    STA   $0300
0325-   A9 15       LDA   #$15
0327-   8D 01 03    STA   $0301
032A-   60          RTS
032B-   00          BRK
032C-   00          BRK
032D-   00          BRK
032E-   00          BRK
```

Continued on next page

```
032F-   A2 04      LDX   #$04
0331-   BD 2A 03   LDA   $032A,X
0334-   95 35      STA   $35,X
0336-   CA         DEX
0337-   D0 F8      BNE   $0331
0339-   60         RTS
033A-   00         BRK
033B-   00         BRK
033C-   00         BRK
033D-   00         BRK
033E-   48         PHA
033F-   49 8D      EOR   #$8D
0341-   F0 05      BEQ   $0348
0343-   CE 00 03   DEC   $0300
0346-   D0 0A      BNE   $0352
0348-   A9 28      LDA   #$28
034A-   8D 00 03   STA   $0300
034D-   CE 01 03   DEC   $0301
0350-   F0 04      BEQ   $0356
0352-   68         PLA
0353-   4C F0 FD   JMP   $FDF0
0356-   A9 15      LDA   #$15
0358-   8D 01 03   STA   $0301
035B-   AD 00 C0   LDA   $C000
035E-   10 FB      BPL   $035B
0360-   48         PHA
0361-   A9 00      LDA   #$00
0363-   8D 10 C0   STA   $C010
0366-   68         PLA
0367-   C9 83      CMP   #$83
0369-   F0 0B      BEQ   $0376
036B-   C9 9B      CMP   #$9B
036D-   D0 E3      BNE   $0352
036F-   A9 01      LDA   #$01
0371-   8D 01 03   STA   $0301
0374-   D0 DC      BNE   $0352
0376-   A5 33      LDA   $33
0378-   C9 BE      CMP   #$BE
037A-   D0 03      BNE   $037F
037C-   4C 03 E0   JMP   $E003
037F-   C9 DD      CMP   #$DD
0381-   D0 03      BNE   $0386
0383-   4C 00 00   JMP   $0000
0386-   4C 69 FF   JMP   $FF69
0389-   FF        ???
038A-   FF        ???
038B-   FF        ???
038C-   FF        ???
038D-   FF        ???
038E-   FF        ???
038F-   FF        ???
0390-   FF        ???
0391-   FF        ???
0392-   FF        ???
0393-   FF        ???
*
```

Figure 3.

Closing Comments

I have chosen not to copyright this program in order to produce maximum distribution. It is my hope that most of the Apple owners will be able to utilize this program. Therefore, any person or group wishing to copy this program is more than welcome to do so.

```
$322 - STA Counter 1
$343 - DEC Counter 1
$34A - STA Counter 1
$327 - STA Counter 2
$34D - DEC Counter 2
$358 - STA Counter 2
$371 - STA Counter 2
$315 - STA Tmpdata-1,X
$331 - LDA Tmpdata-1,X
$306 - EOR Listloc-1,X
$318 - LDA Listloc-1,X
$33A - The low order address for the start of the page list
       routine.
$33B - The high order address for the start of the page list
       routine.
$30E - JMP Setcounters
```

Table 1

A list of locations that would have to be changed if the program was relocated.

```
$33  - Promp Character
$300 - Counter 1
$301 - Counter 2
$302 - Initiation Routine
$320 - Setcounters
$32B - Tmpdata
$32F - Restore Routine
$33A - Listloc
$33C - Low order address of Keyin ($1D)
$33D - High order address of Keyin ($FD)
$33E - Page List Routine
$352 - JMP Cout 1
$37C - JMP Integer Basic
$383 - JMP FP Basic
$386 - JMP Monitor
```

Table 2
Important Locations

JUST POKING AROUND with my PET

BY GARY RATLIFF
101 East Street
Mendenhall, MS 39114

As most of the people who have purchased the PET come to realize, the PET is a crackerjack computer: there's a surprise in every package.

After patiently saving my money, I was finally the owner of a new PET. The first surprise was the almost total lack of instructions; the second was the three prong connection – yes: every wall outlet in the house was wired for two wire 120 volt AC.

The distinction between hardware and software made itself felt immediately. So after making the 180-mile return trip with my new computer from New Orleans, I made the short trip to Jackson to find an adapter for the wall outlet. The lack of good software support would be compensated by my training in computers obtained by reading most of the books I could find on the subject. My only hands-on experience with an actual computer was derived from one of my graduate electives: FORTRAN IV Computer Programming.

The lack of good documentation provided by the company has turned to the PET owners advantage, because PET owners across the country are eager to share the secrets they have been able to garner from their machines. The purpose of this article is to reveal my methods of dumping memory to the screen and how to view the elusive BASIC Interpreter.

There are two very simple procedures which may be used to obtain memory dumps on the screen. One method produces decimal values and the other ACSII values. Either form may be preferred, depending upon the end use of the information. Both methods may be used with either the BASIC Text RAM or the Operating System, with the exception of the BASIC Interpreter.

First load a short (very short – some of your first programming efforts may be put to good use here) program. After you discover the uses of this dump, you may create programs designed not to be run, but to learn some of the internal workings of the computer. When the "READY" message appears, enter the following direct statement:

PRINT 1024 + (7167 – FRE(P))

The printed result will be called X and should reveal the end address of the program you entered. Next follow this with the dump instruction:

FOR I = 1024 TO X: PRINT PEEK (I);: NEXT

The screen will fill with numbers in the range 0-255; And, if your program was short, these will not scroll off the screen.

The numbers will have meaning and be of value to learn: the method used by the PET to link one line to the next, to reveal the token symbol which the PET reduces the key words to, and to serve as an easy method to learn to read the numerical equivalent of decimal ASCII CODE.

A brief example will show exactly how these three items are made meaningful. Consider this short program, which (if you care to watch the screen for a half-hour while this program runs) will map the entire 65K memory to the screen:

```
    1 POKE 59468,14: X=32768: PRINT "clr"
    2 FOR I=ØØ TO 65: FOR J=Ø TO 1ØØØ: A=PEEK(I*1ØØØ+J):
  POKE X,A:X=X+1:IF X> 33767 THEN X=32768
    3 NEXTJ: NEXTI (note spaces have been added for ease in
reading; however, the program has every character spaced as
close together as possible).
    ? 1024 + (7167-FRE(P))
   1126
    FOR I= 1024 TO 1126: ? PEEK(I);: NEXT
0 28 4 1 0 151 53 57 52 54 56 44 49 52 58 88 178 51 50 55 54
56 58 153 34 147 34 0 93 4 2 0 129 73 178 48 48 164 54 53 58
129 74 178 48 164 49 48 48 48 58 65 178 194 40 73 172 49 48
48 48 170 74 41 58 151 88 44 65 58 88 178 88 170 49 58 139
88 177  51 51 55 54 55 167 88 178 51 50 55 54 56 0 103 4
3 0 130 74 58 130 73 0
```

The zero is used to delimit the lines. Now if you count 28 numbers from this zero you will find another zero which delimits the second line. Furthermore, if you count 93 numbers from the first zero you will find the beginning of the third line. The segments 28 4, 93 4, and 103 4 are the links which tell the interpreter where to find the next line. Using 256 bytes to a page of memory page 4 would be location 1024; and the second line would begin at location 28 in this page. The maximum line number is 65535 because this is the greatest number which may be stored in two bytes. The 1 0 indicates $1 \times 256^0 + 0 \times 256^1$, or just line one. Memory space is saved by reducing each of the key words to a token symbol. Thus 151 is the token symbol for POKE; 178 for =; 153 for PRINT; 147 for CLEAR THE SCREEN; 129 for FOR; 164 for TO; and in a similar manner for each of the key words used in the program. The remaining numbers are the decimal code numbers for the ASCII character set used in this program.

A second memory dump is the ASCII dump; this may be obtained with a short direct statement which is a modification

of the decimal dump. With the same program in memory use the following statement:

```
FOR I = 1024 TO 1126: PRINT CHR$(PEEK(I) );: NEXT
```

Notice that the number of characters printed is much less than the 102 byte program. This is due to the fact that certain of the characters are invisible and others represent control operations. But for a long BASIC program this is a convenient method of obtaining a listing without line numbers and without any spacing between lines. However, for a machine language program, the frequency of the op codes which cause the reverse field or the screen to clear or the cursor to jump and write over portions is annoying.

This jumping cursor may make the programmer and the programee change roles, as certain expletives might come to mind when a key to understanding a machine language program is suddenly wiped out by the screen being cleared. This difficulty is removed by the use of PEEK and POKE. These return a different value than the ASCII code, but there are only two invisible characters: 32 which represents a space, and 96 which when not being Maxwell Smart and a pal of 88 represents the op code for RTS (return from subroutine).

The example program is a memory dump in PEEK and POKE. By changing the initial value of I from 00 to, say, 59. We direct ourselves to the PET operating system. The reason for the line POKE 59468 was because the ASCII code for the alphabet runs from 65 to 90. If the graphic characters are to be printed, the text portions of a machine language program will be difficult to find. The values for PEEK and POKE from i = 65 to 90 are the alphabet in lower case; hence, messages in text of program become readable again.

Now to the elusive but not impossible BASIC Interpreter. By changing the value of I to 49 we soon reach this at the memory address 49152. Using the decimal dump, the screen fills up with zero upon zero. This is great if any of your school age children ask you: "...but Daddy, what is zero?" You now have an excellent method of showing them over eight thousand of them!! Programming in BASIC will not produce a meaningful dump of this area of memory.

Ah, on to the forbidden fruit. The computer will do anything it is told to do if machine language is used. And the PET has a very convenient method of using machine language.

Those of you who own not only a computer but also a programmable calculator should already be familiar with assembly language. In proper form this program, which we are going to assemble into the second cassette buffer, would be written as follows:

```
                SCRN = $ 8ØØØ        TRGT = $ CØØØ
                VPIA = $ E84C          * = $ Ø33A

Ø33A  18              CLC
Ø33B  A9 ØE           LDA # 14
Ø33D  8D 4C E8        STA  VPIA
Ø34Ø  A2 FF           LDX # 255
Ø342  BD ØØ CØ   LP1  LDA  TRGT , X
Ø345  9D FØ 8Ø        STA  SCRN + 24Ø , X
Ø348  CA              DEX
Ø349  DØ F7           BNE  LP1
Ø34B  6Ø              RTS
                      END
```

Therefore, in only 18 bytes of machine language, we have a tool which brings Birnam Wood to Dunsinane. This program translates into the following decimal code which is more usable with the POKE statement: 24, 169, 14, 141, 76, 232, 162, 255, 189, 0, 192, 157, 240, 128, 202, 208, 247, 96 ... Now the value of TRGT may be changed to yield the entire memory in one page blocks. You will note the speed of the machine language over the same type of memory mapping via the PEEK and POKE method. When assembling via the following program change this value from 192 to 0 and then be prepared to step through memory in one page blocks.

```
     PROGRAM NAME: MCHN LNG RTN LDR (machine language routine
loader)  by Gary L. Ratliff
     1 X = 826
     2 PRINT X : INPUT Y : POKE X,Y : X=X+1: IF Y <> 96 THEN 2
     3 PRINT " PGM ASMBLD  HIT ANY KEY TO CONTINUE "
     4 GET X$ : IF X$ =  "" THEN 4
     5 FOR I = Ø TO 255 : POKE 836 , I :  SYS (826): PRINT
" home " : ? " THIS IS PAGE "; I; " OF MEMORY  HIT ANY KEY "
     6 GET X$ : IF X$ = "" THEN  6
     7 NEXT  I
```

Again the program has been spaced for ease of reading to work as written when it reaches memory location 836 input a 0 instead of 192.

The next problem is that the output of decimal numbers or that of the POKE and PEEK set or the ASCII characters might represent a higher level of abstraction than most people would wish to take time to make concrete. However, to those people who are interested in how this computer works, the contents of the operating system and the basic roms are worthy of much study.

I mentioned that the lack of documentation had caused many PET owners to be eager to share what little information they had been able to gather about the PET. Several months ago one of the first articles I read concerning the PET was written by Mark Zimmerman and appeared in the September-October 1978 *People's Computers*: "Snooping with Your PET." This provided the raw material needed to create a 6502 disassembler which I named "MERRIMACK" together with the MCHN LNG RTN LDR and "MONITOR" comprise a machine language system. Therefore, I hope this article will inspire some new owners to learn more about their computers. And also some old hands who have ideas more worthwhile to perhaps be willing to share them.

Some people think computers have a brain. Do they? Does the interpreter not wish to be seen? After creating the system in which the loader calls in the disassembler and then the disassembler calls in monitor—linking the programs in one package and testing and debugging the system (and one day out of warranty)—the tape drive failed. No programs would load and no new programs could be saved. How about that, you fans of Murphy's Law? If a thing can go wrong it will—and it will go wrong at the most inopportune moment. Yes, my brother had a five game pack on order, my wife ordered "6502 Disassembler and PEEK-A-BOO" for my Christmas present. And I had ordered the "6502 ASSEMBLER IN BASIC."

These methods will allow you to view all of memory—but does the computer have a brain and will it retaliate?

Quick and Dirty Routines for the Sweet-16

BY STEVE WHEELER
P. O. Box 15
U. S. Naval Facility Antigua
FPO Miami 34054

The Apple II computer contains in its firmware a 16-bit pseudo-machine called SWEET-16. A subroutine call to SWEET-16 causes all following code to be interpreted as instructions to the 16-bit pseudo-machine until a return to 6502 instruction (RTN) is encountered. SWEET-16 is a powerful adjunct to the Apple II computer, and several programs have been written for the Apple containing SWEET-16 code, including programs to renumber or to recover integer BASIC programs. Up until now, only Apple has had an assembler which would recognize SWEET-16 mnemonics, which meant that all other programmers have been forced to hand-assemble SWEET-16 code, and insert it into their programs as hex data. The following program is a modification to the cassette version of the S-C Assembler II which gives it the capability to recognize and correctly assemble SWEET-16 mnemonics.

How It Works

The assembler is a line-oriented, fixed-field assembler. Each line in the program is copied from the text file into a buffer at $0200 prior to assembly. This program grabs control from the main assembler at location $1477, by replacing the instruction there with a hook to the SWEET-16 assembly routines. The branch to a jump is necessary to avoid disturbing the instruction at $1479. What this does is cause these routines to take control if the entry in the instruction field of the line being assembled is not recognized as a pseudo-op or a 6502 instruction mnemonic.

The mnemonic is then compared with entries in the instruction table. Spaces are ignored, with one exception which is covered in the section on program limitations. If there is a match, a type byte is used to determine which routine(s) will be used to produce the final opcode, and to return to the main assembler at the proper point.

The instruction table format is as follows: the letters in the mnemonic are in ASCII code with the high bit set on the last character only. No space characters are included. The mnemonic is followed by an opcode byte, and then by a type byte. The end of the table is signified by table entries of $FF.

Routines Used in the Main Assembler

As written, these routines will work only with the cassette version of the S-C Assembler II. The following explanation is for those who wish to adapt the routines to another assembler.

GTCH – This routine loads the accumulator with a character from the line buffer, and converts the character to true ASCII (high bit clear). The Y register is used as the index into the buffer. Location $06 is used to store the value of Y between calls. Before returning, the character in the accumulator is compared to a space, the carry flag is cleared, and Y is incremented and stored into $06.

BLD – This routine updates the location counter, and builds the file of assembled code if called during pass #2. The value in the X register is used to determine which pass is in progress.

BYT1 – This is the entry point for single-byte opcodes.

BYT2 – This is the entry point for branch instructions. This returns to the main assembler just after an opcode would have been obtained, and just before the calculation of the relative displacement.

BYT3 – This is the entry point used for the SET opcode. It is the second instruction of the subroutine which handles the .DA pseudo-op.

NOPE – This is the return point if a match is not found. It prints the error message "BAD OPCODE AT LINE", aborts the assembly, and returns the assembler to command mode.

How To Use the Program

Before assembling this program, one patch must be made to the assembler. Location $1010 must be changed from $1C to $1E to move the start of the symbol table past these routines. If this is not done, the program will be stored on top of its own symbol table, and the assembly will be aborted.

The following mnemonics are recognized:

Non-register operations	Register operations
BC	ADD R
BK	CPR R
BM1	DCR R
BP	INR R
BR	LD R
BS	LD @R
BZ	LDD @ R
RS	POP @R
BM	POPD @R
BNC	SET R
BNZ	ST R
BNM1	ST @R
RTN	STD @R
	STP @R
	SUB R

Spaces within a mnemonic are ignored. The register number is specified by one ot two decimal digits following the 'R' for a register operation.

Limitations of the Program

These routines were designed as a minimal add-on to an existing assembler. As such, there are no error checks written into them. The following are "gotchas":

1. The subroutine REG, which calculates the register number for a register operation, assumes that the first character following the 'R' is a number, a space, or an end of line token. If the second character is not one of those, it is treated as a number, and the previous character is assumed to be a '1'. This means that ADD R 3, ADD R%3, ADD R13, ADD R03, and ADD RQC will all produce the op-code 'AD'.
2. The assembler requires a fixed-format line, and assumes that a particular index into the buffer will point to the first character of the operand. This feature has been sidestepped for the SET instruction, but not for branches. This means that a two-character branch mnemonic must be followed by two spaces before the branch destination, a three-character branch mnemonic must have one space between it and the destination, and the four-character branch mnemonic (BNMI) must be immediately followed by the destination, with no intervening spaces.
3. Do no put a comma between the SET mnemonic and its operand. Just leave a space between them. Example:

SET R4 BUFF

These are the limitations that I know of. If anyone discovers other flaws, or comes up with a fix for these, I would appreciate hearing about them.

Good Points

1. Since these routines only produce the opcode, the host assembler handles all operands. Therefore, operands in branches and the SET instruction can be decimal, hexadecimal, or labels.
2. Where else can you get an assembler which handles SWEET-16?

Final Comments

If the above limitations are respected, assembly of SWEET-16 code is correct and almost uneventful. I say almost because I nearly knocked my Apple off the table the first time the routines worked correctly, and I still have to stifle a grin whenever I use them.

These routines are basically a "quick and dirty" implementation of a SWEET-16 assembler. Since I have only a few months of experience with assembly language, writing them was anything but quick. My purpose in publishing them is simple: I'm hoping that similar routines, suitably improved, will be incorporated in future assemblers for the Apple II computer. It's time that assemblers for the Apple were capable of recognizing and assembling ALL instructions which the Apple can execute.

I would also like to dedicate this program (Why not? It's done for books.) to Chris Johnson, who gave me my start in assembly language programming, and who provided the original idea for this program.

References

The Apple II computer is produced by the Apple Computer Company of Cupertino, California. The S-C Assembler II is available from S-C Software, P.O. Box 5537, Richardson, Texas 75080. The definitive article on SWEET-16 originally appeared in the November 1977 issue of BYTE magazine.

Producing Pictures on your Computer with a Diablo Printer

BY JAY C. BOWDEN
SAI Technology Company
4060 Sorrento Valley Blvd.
San Diego, CA 92121

ANNA K. SCHARSCHMIDT
Linkabit Corporation
10453 Roselle St.
San Diego, CA 92121

INTRODUCTION

While hobbyists might like to experiment with digital image processing and recognition, most are prevented from doing so by the prohibitive expense of dedicated equipment, i.e., a scan converter camera and complex interfacing. But if you own, or have access to, a Diablo or similar type incremental printer, you already have the basis for beginning your own experiments in image enhancement and recognition.

Before your home computer can be used for digital image processing, you must find a way to present it with the image data. This is usually done in the form of an array of intensities. This is an array of elements N x M, where each element represents an intensity at a discrete point i, j where $0 \leq i \leq N$, $0 \leq j \leq M$.

To obtain data for this array you would usually need an expensive, digitizing camera involving sophisticated interfacing. However, using only your Diablo printer, a phototransistor, and an Analog to Digital (A/D) converter, you can obtain intensity arrays for experimentation from still photographs.

THE TECHNIQUE

The Diablo-type printer has the ability to move its printhead horizontally and vertically in increments of 1/60 and 1/48 inches, respectively. To produce an intensity array, you can attach a phototransistor to the printhead while there is a picture in the platen, and cause the printhead to scan over the image surface as you digitize the intensity at each point. The phototransistor used was a TIL139 Emitter-Sensor package, with the emitter left unused. A slot can be cut in the package which allows you to slide the TIL139 onto the fin above the striker of the Diablo printhead (Fig. 1).

THE CIRCUIT

Since only the phototransistor of the TIL139 is used, you need to run wires from its two leads to your nearby A/D converter. There you should connect a resistor to the collector for sourcing current to the phototransistor, and ground to its emitter (Fig. 2).

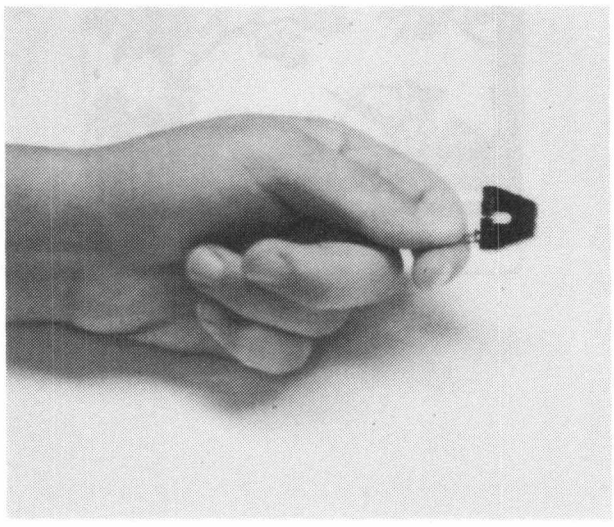


Figure 1. TIL139 with slot cut in it.

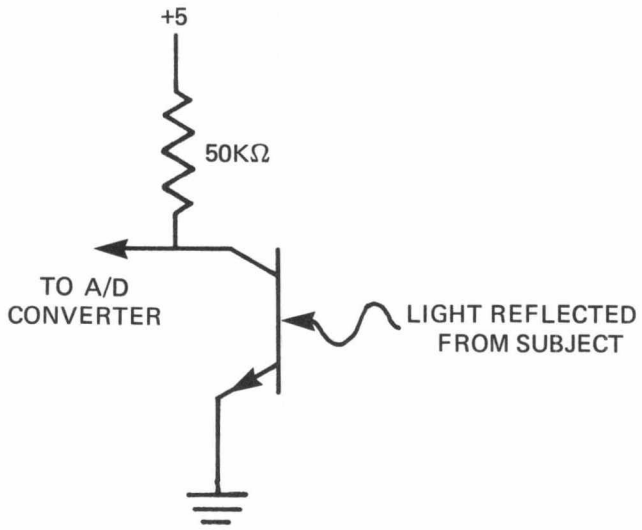


Figure 2. Connection of TIL139 phototransistor.

The resistor value should be about 50K ohms for a +5 volt supply. With this setup, the voltage from the collector will be inversely proportional to the amount of light that is reflected off the platen into the phototransistor. You can experiment with the value of the resistor by manually moving the printhead from light to dark areas and try to get the greatest light to dark variation.

Once the circuitry is set up, all you need to do to digitize your picture into an intensity array is to scan the printhead over the whole picture as you digitize the intensity values. This is done by a program in your computer that sends special control commands to the Diablo to make it move its printhead, then reading the A/D converter and storing the value in an array. Listing 1 is an example of a BASIC program that will scan the printhead using the Diablo's graphics mode. We assume here that the Diablo is connected as the console terminal.

STORING THE DATA

Once we have an intensity value for a point, we need a way of saving that value so it can be recalled later. Previous experimentation demonstrated that for this type of application, 64 discrete intensity levels is more than adequate (Ref. 1). This corresponds to 5 bits of data per point. Most BASICs deal with only two data types: the floating point variable and the character variable. ASCII characters range in value from 0 to 127, and if we eliminate the control codes and certain punctuation marks that tend to upset some systems, we find about 80 contiguous character values are available for our use.

To code our intensity value, which I presume here came from an 8 bit A/D converter, we first scale the value into the range 0 to 63 by multiplying by 64/256. Then we add an offset that transforms our value to a code for a printable ASCII character. We convert the code to a character string and append as many characters as we have samples per horizontal scan. This produces a string of characters that can be written to a file then recovered and decoded by another program.

THE SOFTWARE

The program in Listing 1 first asks for a file name into which it will write the digitized data in the form of lines of ASCII characters. Next, you should enter the number of horizontal steps the printhead should make between each sample point. Remember, in graphics mode each horizontal step is 1/60th of an inch. Next, enter the number of vertical steps, each being 1/48th of an inch. These numbers should be selected to match the aspect ratio of the device you will print your pictures out on, or your reproduction will be distorted, i.e., too tall and skinny, or too short and fat.

Next, you will enter the array dimensions, or the number of sample points in each axis. You are then asked to "POSITION SUBJECT . . . ". At this point you should remove the paper from the Diablo's platen and put in your picture to digitize. The program then expects you to enter a null string, e.g., space and return, before it begins scanning the printhead.

To initiate the scanning motion of the printhead, the printer is set to graphics mode by the program which sends the proper control codes in the form of a text string. It then digitizes a point, converts the value into an ASCII character, and appends it to the text string G$. To step to the next point, we send space codes which step the printhead 1/60th of an inch each. The process repeats for the number of x axis sample points requested.

At the end of a horizontal scan, we send the specified number of line feed codes, each of which moves the subject up 1/48th of an inch. Then we write the text line out to our file, and begin the next horizontal scan.

After stepping through the specified number of vertical sample intervals and writing out the last line, the program again awaits a null string to allow you to remove the picture and replace the blank paper. Enter a space and a return, and the program will finish by printing the maximum and minimum intensity values that were digitized.

REPRODUCING A PICTURE

To reproduce a picture, we can read back our data file line by line and decode the characters back into intensity values. If for each point we print a character whose density is based on that point's value, we will end up with a printout that looks something like our original picture.

Listing 2 is an example of a program that produces pictures from our coded ASCII files. The program also performs contrast expansion by accepting the minimum and maximum values in the file and scaling the data so the minimum value corresponds to the least dense character, while the maximum corresponds to the most dense.

CONCLUSION

The programs and ideas presented here should provide a starting point for those wishing to experiment with pictures on their home computers. I will suggest a few possible enhancements you may wish to try:

1. In Listing 1, the string R$ contains the characters used for different densities. Try substituting other characters or adding more.
2. Try overstriking characters to obtain a greater maximum density (Ref. 2).
3. Most of the "video to T-shirt" concessions use printers modified to eliminate the space between character positions, both horizontally and vertically. A Diablo can be set to do this also using the HMI and VMI (Ref. 3).
4. It is possible to reproduce your pictures on devices on which only two densities are available, e.g. "on" and "off," such as a high resolution graphic CRT. A simple technique is called Dot Space Modulation, and is covered in Ref. 2. A Diablo-produced example is shown in Figure 4.

References

1. Gonzales, Rafael C. and Wintz, Paul A. : *Digital Image Processing*; Addison-Wesley, 1977
2. Hale, Jacques A.G. : Dot Space Modulation For The Production of Pseudo Grey Pictures; Proceeding of the S.I.D. Vol 17/2, Second Quarter 1976 pp. 63-74
3. McDonough, Tom: Computer Graphics With the Diablo; Creative Computing vol. 5, no. 6, June 1979 pp. 32-35

```
100     L$=CHR$(10)
110     E$=CHR$(13)+CHR$(27)+"3"
200     INPUT "FILE NAME:";A$
214     INPUT "NO. OF HORIZ STEPS/SAMPLE:"; H2
218     INPUT "NO. OF VERT STEPS/SAMPLE"; V2
220     INPUT "WIDTH:";W1
230     INPUT "LENGTH:";L1
235     FILE A$
240     INPUT "POSITION SUBJECT.";A1$
242     ASPR=W1/L1 : ASPR=ASPR*ASPR
243     Z=SQR(ASPR*L1*L1+W1*W1)
245     L3=1000 : H3=0
250     FOR I1=1 TO L1
255     G$=""
260             FOR I2=1 TO W1
265             REM READ A/D CONVERTER HERE...
270             Q=SQR(I1*I1*ASPR+I2*I2)
280             Q=63*Q/Z
282             IF L3>Q THEN L3=Q
284             IF H3<Q THEN H3=Q
290             Q=Q+43
300             Q$=CHR$(Q)
310             G$=G$+Q$
312             FOR M=1 TO H2: PRINT " "; : NEXT M
320             NEXT I2
330     FOR M=1 TO V2: PRINT L$; : NEXT M
340     PRINT #1;G$
350     NEXT I1
352     INPUT A1$
354     PRINT : REM GET OUT OF GRAPHICS MODE
360     PRINT "DONE..."
365     PRINT "MAX= ";H3;" MIN= ";L3
370     CLOSE 1
380     STOP
390     END
```

Listing 1. Program to produce intensity array file. Lines 242, 243, 270, and 280 will cause a test file to be produced. Replace lines 270 and 280 with your READ A to D converter code. Note the program also determines the minimum and maximum values digitized. File conventions are for BASIC-E with CP/M.

```
"000112234456778899:;<<=>??@ABBCDDEFGGHIIJKLMMNOPPQRRSTTUVWXXY"
"233344455677889:;;<<=>??@AABCDDEFFGHIIJKLLMNNOPQQRSTTUVWWXYY"
"566666778899:;;;<==>>?@@ABBCDDEFGGHIIJKLLMNNOPQQRSSTUVVWXXYZ"
"8899999:;;;;<<==>>?@@AABCCDDEFFGHHIIJKLLMNOOPQQRSSTUVVWXXYZ["
";;<<<<<<==>>??@@@ABBCCDDEEFGGHIIJJKLLMNNOPPQRRSTTUVWWXYYZ[["
">>>?????@@@@AABBBCCDDEEFFGGHHIJJKKLMMNNOPPQRRSTTUVVWXXYZZ[\\"
"AAABBBBBBCCCCDDDEEFFFGGHHIIJJKLLMMNNOPPQQRSSTTUVVWXXYZZ[\\]↑"
"DDDDEEEEEEFFFFGGGHHIIIJJKKLLMMNNOOPPQQRSSTTUUVWWXYYZZ[\\]↑↑←"
"GGGGGHHHHHHIIIIJJJKKKLLMMMNNOOPPQQRRSSTTUVVWWXXYZZ[[\]]↑←``^"
"JJJJJKKKKKKLLLLLMMMNNNOOPPPQQRRSSSTTUUVWWXXYYZZ[[\]]↑↑←``aab"
"MNNMMMNNNNMNNOOOOPPPPQQQRRRSSTTTUUUVVWWXXYYZZ[[\\]]↑↑←``aabccd"
"PPPPPPQQQQQQQRRRRSSSSTTTUUUVVWWWXXYYYZZ[[\\]]↑↑←``abbccddef"
"SSSSSSSTTTTTTTUUUUVVVVWWWXXXYYYZZ[[[\\]]↑↑←``aabbccddeeffgg"
"VVVVVVVWWWWWWXXXXXYYYYZZZ[[[\\\]]↑↑↑←``aabbccddeeffgghhii"
"YYYYYYYYZZZZZZZ[[[[[\\\\]]]↑↑↑←``aabbbccddeefffgghhiijjkl"
```

Test output of program in Listing 1. Surrounding quotes are added by BASIC-E. They are not necessary and may not appear in your file.

```
100     R$=". :;-+*%$#@@"
110     N1=10
200     INPUT "WHAT FILE:";A$
210     FILE A$
300     INPUT "WIDTH:";W1
310     INPUT "LENGTH:";L1
320     INPUT "MAX VAL:";H1
330     INPUT "MIN VAL:";L0
340     REM
350     D=H1-L0
360     FOR I1=1 TO L1
365     Q$=""
370     READ #1;D$
380             FOR I2=1 TO W1
390             Q=ASC(MID$(D$,I2,1))
400             Q=Q-43-L0
410             Q=INT(Q*N1/D)+1
420             Q$=Q$+MID$(R$,Q,1)
430             NEXT I2
440     PRINT Q$
500     NEXT I1
600     CLOSE 1
700     PRINT "DONE..."
1000    STOP
```

Listing 2. Program to convert coded data files into printer pictures.

```
WHAT FILE:? WIM3
WIDTH:? 60
LENGTH:? 15
MAX VAL:? 63
MIN VAL:? 3
```

```
.....:::::::::::::::::::----------++++++++**********%%%%%%%%%%
.....:::::::::::::::::::----------++++++++**********%%%%%%%%%%
.:::::::::::::::::::::::----------++++++++**********%%%%%%%%%%$
:::::::::::::::::::::::----------++++++++**********%%%%%%%%%%$$
:::::::::::::::::::----------++++++++**********%%%%%%%%%%$$$$
;;;;;;;;;;;;;;;----------++++++++**********%%%%%%%%%%$$$$$
;;;----------++++++++**********%%%%%%%%%%$$$$$$$
----------++++++++**********%%%%%%%%%%$$$$$$$$
-----++++++++++++++++++++**********%%%%%%%%%%$$$$$$$$$$
++++++++++++++++++++**********%%%%%%%%%%$$$$$$$$$$#####
++++++**********%%%%%%%%%%$$$$$$$$$$#########
**********%%%%%%%%%%$$$$$$$$$$##########@
*******%%%%%%%%%%$$$$$$$$$$##########@@@@
%%%%%%%%%%$$$$$$$$$$##########@@@@@@@@
%%%%%%%%%%$$$$$$$$$$##########@@@@@@@@@@@@
DONE....
```

Test output generated when program of Listing 2 operates on file generated by program of Listing 1.

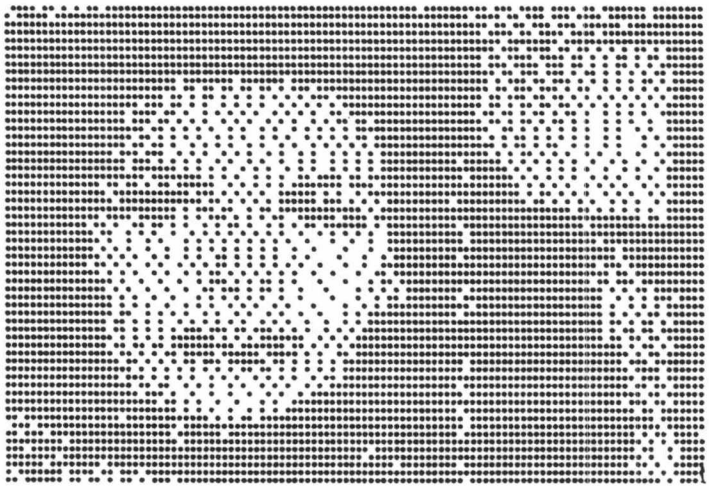


Figure 3. Example of original and line printer reproduction. Array was 110 by 60 with R$=" .:; + =8@ # #".

```
A>TYPE SAPR.BAS

2000    REM SUCCESSIVE APPROXIMATION SUBROUTINE
2010    REM P IS PORT NUMBER
2020    T=0 : L=128
2030    FOR I=0 TO 7
2033            T=T+L
2040            OUT P,T
2050            B=INP(P)
2060            IF B=1 THEN 2090
2070            T=T-L
2090            L=L/2
2100    NEXT I
2110    RETURN
```

BASIC subroutine for performing Analog to Digital conversion.

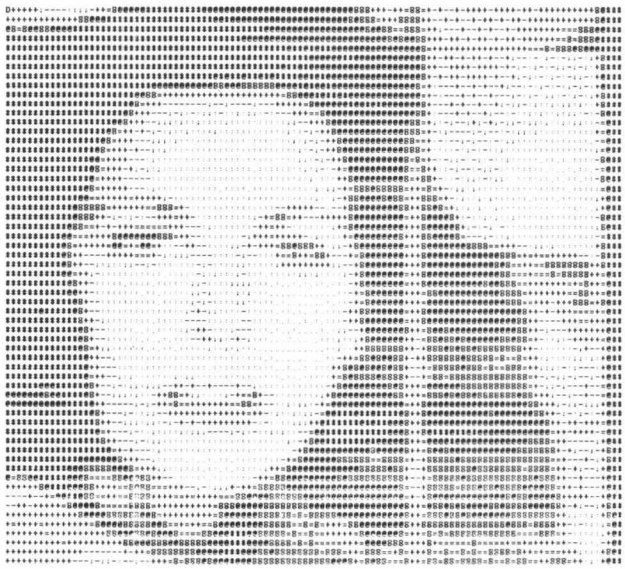


Figure 4. Example of two level reproduction. (110 X 60 dots)

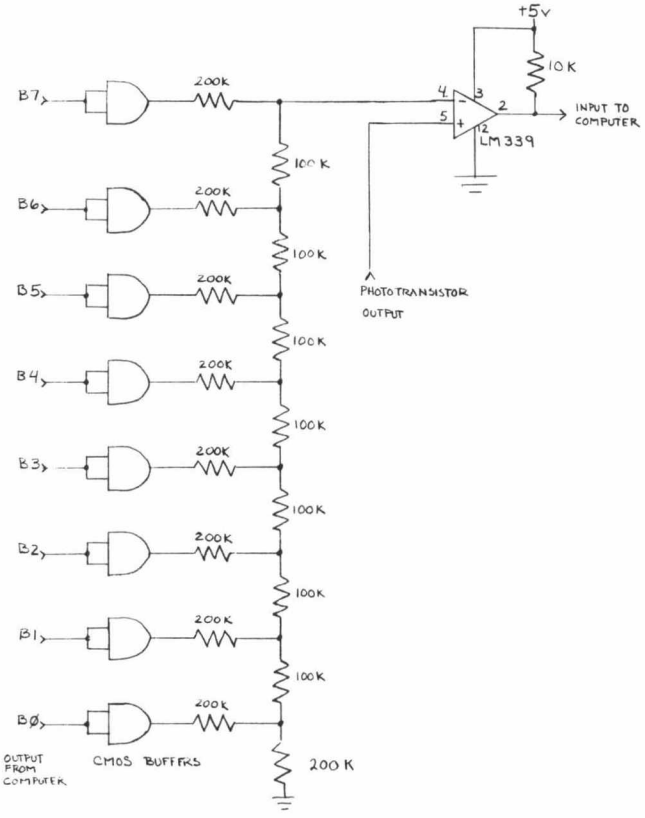


Figure A. Digital to Analog converter.

WHAT TO DO IF YOU DON'T HAVE AN A/D CONVERTER

Realizing that not everyone has an A/D converter lying around, here's how to make one with a few resistors, a comparator and some CMOS gates.

By building a D to A converter and running a simple BASIC program, you can perform A to D conversion. You will need a latched eight bit output port and an input port that has a readable bit. Build the circuit shown in Figure A. It is important to use CMOS gates because of their matched output levels. TTL gates would cause non-linearities in the analog output of the R/2R ladder.

The BASIC program will do a successive approximation or "binary search" to find the digital value nearest to the analog value from the phototransistor.

The program sets one bit at a time starting with the most significant, then looks at the output of the comarator to see if the phototransistor value is larger or smaller. If larger, it leaves that bit on and tries setting the next lower bit. If smaller, it turns that bit off and sets the next lower bit, until all eight bits have been set or reset.

David and Goliath

The Little Guy is a Challenger!

BY DAVID MORGANSTEIN
9523 48th Place
College Park, MD 20740

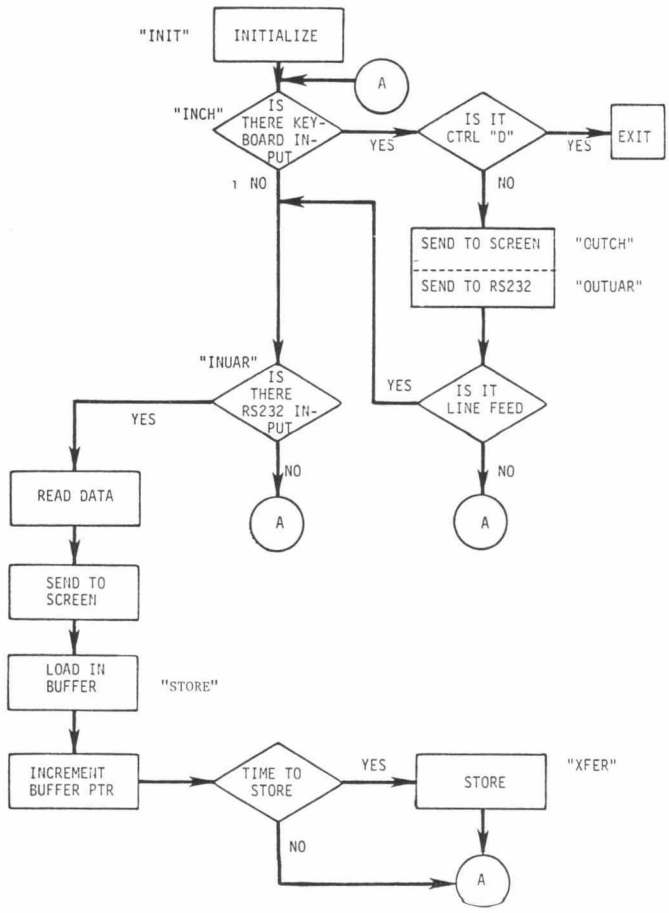
Ever since acquiring a micro, one of my keen desires was to sign on to a main-frame and say "Hello." Through my work as a statistician I have occasion to use the big-guys for one thing or another. Still, my micro hobby pushed me towards a two-way conversation. This article describes how I succeeded in setting up the OSI 430 Board RS232 interface so that I could connect to a modem, and then out the phone lines.

I cannot claim to be a hardware specialist so I will not try to explain that which I do not fully understand. I did manage to write a monitoring routine so that my 6502 MPU played smart terminal, storing all that I received from the main-frame. I left out of the accompanying source listing the routine which would store the one track buffer on disk at a specified track, then increment the track counter in preparation for more data. I left this out because it is not really to the point. Also, the form of the routine would vary according to the OSI system you were using.

On to the main point. The source program has several responsibilities. It must initialize the RS232 port, monitor my keyboard in case I want to send something to the mainframe, and monitor the port in case the big-guy has something for me. In either case, the desired data must be picked up and sent to the other member of this two-way conversation. The flow diagram gives a quick overview as to how this is all accomplished.

In the 'INIT' routine, the 430 board is reset and set up for the proper transmission pattern. Next, the starting location of the storage buffer is established. This routine is followed by the 'INCH' (INput CHaracter) routine. This code examines the Challenger ACIA status byte to see if there are any serial keyboard entries. If so, the data byte is picked up and examined. If it is 'CTRL' D, a return to BASIC occurs (note, this is also system dependent). Otherwise, the data byte is sent to the 430 board for delivery to the main-frame. Also, the character is echoed to the screen. Lastly, the byte is compared to a 'line feed'. If it is one, transfer to the 'INUAR' routine occurs. The 'INUAR' performs a similar operation with the incoming RS232 data. If data is available, it is transferred to the screen and added to the buffer. If the buffer fills, a transfer to disk routine is called. Otherwise, the storage pointers are merely incremented for the next byte. The port is then cleared and tested for a new data byte. If none appears, a return to the 'INCH' occurs. Thus, the program ping-pongs between the two points, keyboard and RS232, transferring anything found at one point to the other.

One last remark. Before attempting to write this code, I asked other micro enthusiasts how they might tackle the interfacing problem. The advice always suggested that a simple BASIC program would do. After trying repeatedly to accomplish my goal in BASIC, I concluded that it was just too slow. To use it, I had to write the sign-on protocol into BASIC 'PRINT' statements. Then I had to hope for proper timing since an 'INPUT' statement executed after the computer had sent a line would hang up the system. Rather than using 'INPUT', I tried 'PEEK'ing the port. This solved one problem but demonstrated that BASIC was too slow to keep up with even 300 baud. I kept losing random characters. Finally in desperation I went to this machine code approach and what do you know: it worked!



```
                ; ***********************************************
                ; *                                             *
                ; *    TELECOMMUNICATIONS INTERFACE PROGRAM     *
                ; *     WRITTEN FOR O.S.I. 430 BOARD RS232      *
                ; *        BY DAVID MORGANSTEIN  4/20/79        *
                ; *                                             *
                ; ***********************************************
                        ; INPUT ADDRESS
FB03=           INBYT=$FB03
                        ; OUTPUT ADDRESS
FB04=           OUTBYT=$FB04
                        ; STATUS/CONTROL REGISTER
FB05=           UART=$FB05
                        ; INITIALIZATION REGISTER
FB06=           RESET=$FB06
                        ; KEYBOARD CONTROL REGISTER
FC00=           ACIAST=$FC00
                        ; KEYBOARD DATA REGISTER
FC01=           ACIADA=$FC01
                        ; BUILD BUFFER POINTER ADDR HERE
00FE=           PNL=$FE
                        ; BUFFER START PAGE
00D0=           LOBUF=$D0
                        ; BUFFER END PAGE
00E0=           HIBUF=$E0
                        ;
                        ; BEGIN
                        ;
A000                *=$A000
                        ; INITIALIZE UART, STORAGE POINTERS
A000 AD06FB     INIT    LDA RESET
A003 A9FF               LDA #$FF
A005 8D05FB             STA UART
A008 A900               LDA #0
A00A 85FE               STA PNL
A00C A9D0               LDA #LOBUF
A00E 85FF               STA PNL+1
                        ; INPUT CHARACTER FROM KEYBOARD
A010 A220       INCH    LDX #$20
                        ; DELAY LOOP TO PAUSE FOR INPUT
A012 CA         LOOP    DEX
A013 F019               BEQ INUAR
A015 AD00FC             LDA ACIAST
A018 4A                 LSR A
A019 90F7               BCC LOOP
                        ; INPUT AVAILABLE
A01B AD01FC             LDA ACIADA
                        ; MASK OFF PARITY BIT
A01E 297F               AND #$7F
                        ; CHECK FOR CONTROL 'D' TO RETURN
A020 C904               CMP #$04
A022 F01F               BEQ END
                        ; ECHO TO SERIAL TERMINAL
A024 2046A0             JSR OUTCH
                        ; SEND KEYBOARD ENTRY TO RS232
A027 8D04FB     OUTUAR  STA OUTBYT
                        ; CHECK FOR LINE FEED
A02A C90A               CMP #$0A
A02C D0E2               BNE INCH
                        ; MONITOR RS232 FOR INPUT
A02E AD05FB     INUAR   LDA UART
A031 4A                 LSR A
A032 90DC               BCC INCH
                        ; DATA AVAILABLE
A034 AD03FB             LDA INBYT
                        ; SEND TO SERIAL TERMINAL
A037 2046A0             JSR OUTCH
A03A 2053A0             JSR STORE
                        ; CLEAR RS232 INPUT REGISTER
A03D AD06FB             LDA RESET
A040 4C2EA0             JMP INUAR
                        ; RETURN TO BASIC
A043 4C251B     END     JMP $1B25
                        ; OUTPUT TO ACIA
A046 48         OUTCH   PHA
A047 AD00FC             LDA ACIAST
A04A 4A                 LSR A
A04B 4A                 LSR A
                        ; CHECK ACIA STATUS
A04C 90F9               BCC OUTCH+1
A04E 68                 PLA
A04F 8D01FC             STA ACIADA
A052 60                 RTS
                        ; ADD TO BUFFER
A053 A000       STORE   LDY #0
A055 91FE               STA (PNL),Y
                        ; INCREMENT BUFFER POINTER
A057 E6FE               INC PNL
A059 D00F               BNE RTN
                        ; INCREMENT BUFFER PAGE COUNT
A05B E6FF               INC PNL+1
A05D A5FF               LDA PNL+1
A05F C9E0               CMP #HIBUF
A061 D007               BNE RTN
                        ; BUFFER FULL, XFER TO DISK
A063 206BA0             JSR XFER
                        ; RESET BUFFER POINTER
A066 A9D0               LDA #LOBUF
A068 85FF               STA PNL+1
A06A 60         RTN     RTS
                        ; PUT TRANSFER ROUTINE HERE
A06B 60         XFER    RTS
```

Continued from pg. 33

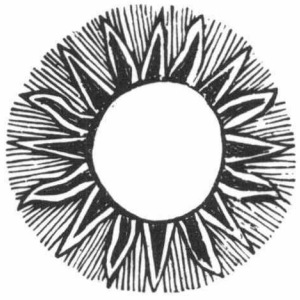
6	−.20	.2431	3.4256
5	−.20	.0674	3.8358
4	−.20	−.1259	4.0935
3	−.20	−.3385	4.1644
2	−.20	−.5724	4.0085
1	−.20	−.8296	3.5797
0	−.20	−1.1126	

The revised estimate is x = 1.4108. Repeating the cycle, I get estimates of 1.3132, 1.2483, 1.2194, and 1.2142. Try the next cycle and see if we haven't almost reached convergence (these particular numbers seem to be converging quite slowly). By the way, the interest rate is so high, about 21%, because you don't get the full use of your money; you have to start paying back right away rather than getting to use the full $1.00 for all 10 years.

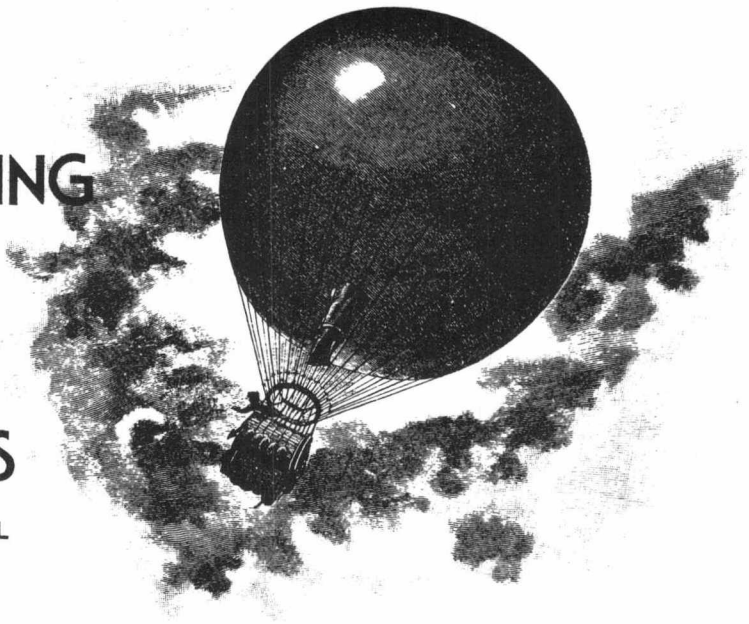
Morals of This Story

A loan in which you make a repayment as soon as you get your money may seem a little silly. But if the loan is really a lease and you receive not money but a new Transamerica pyramid, then it makes sense to pay some rent the day you get your front door key. Big businesses, both lessors and lessees, do these calculations constantly to make sure their leases are profitable and well-managed. In fact, I learned about this problem by consulting for a software house in Silicon Gulch on their lease reporting program; the mathematics was worth a lot to me and more to them.

What are the morals to this story? First, although you don't need to be a mathematical wizard to be a programmer, sometimes a little mathematics helps. Second, having a little mathematics isn't nearly as important as knowing when to ask for help from someone who does have some mathematics. Notice that in our method for solving the interest problem, only simple arithmetic is used. The mathematics comes in the analysis of the problem so that an arithmetic solution is available. Anybody can do the arithmetic; if you can't do the mathematics, don't be too proud to ask for help. Third, even after the analysis is over, there is still a lot of programming to do. I worked for about 10 hours on the analysis; the company worked 10 weeks on building the analysis into a complete lease reporting package for their customers. All the mathematics would have been useless without a good programmer to make the method available to the end user, an accountant who doesn't care about either programmers or mathematicians.



PROGRAMMING PASTIMES AND PLEASURES



BY CHARLES WETHERELL

© 1979, Charles Wetherell

Charlie Wetherell has been programming since 1961 – business programming, applications programming, systems programming. His book, Etudes for Programmers, *was published by Prentice-Hall. He enjoys hearing from readers.*

An Interesting Problem

The naive view of programming is that it requires a great deal of mathematical skill and knowledge. Certainly logical analysis and deductive thinking are of value to programmers, but specific training in mathematics is not necessary. COBOL bears witness to this fact; it is the economically most important language and it is difficult to express anything more complicated than a division in COBOL. Occasionally, though, mathematics problems arise even in the most resolutely businesslike of programs. I would like to describe one such problem this month.

Compound Interest

If I loan you $1.00 for 10 years at 5% *simple* interest, in 1989 you will owe me $1.50 – the original $1.00 and 5¢ a year interest. On the other hand, if I loan you the same dollar for 10 years at 5% *compound* interest, you will owe me much more in 1989. To compute the exact amount, we must find what you owe at the end of each intervening year. In 1980, you will owe $1.05 or $1.00 plus 5% of $1.00. In 1981, you will owe $1.05 plus 5% of $1.05 or $1.1025. In 1982, it will be $1.1025 plus 5% of $1.1025 or $1.157625. In general, after n years you will owe $(\$1.05)^n$; after 10 years this comes to $1.63 rounded to the nearest penny. Compounding cost you an extra 13¢, or 26% more in interest (quite a lot if the loan were for one million instead of one lousy smacker).

Let's turn the problem around. Assume that I loan you $1.00 now and that you promise to repay $2.00 in 1989. What rate of interest am I charging you? If the interest is simple, the rate is obviously 10% or a dime a year. But if we want to know the compound interest rate, we have to solve the equation

$$(1.00+i)^{10} = 2.00$$

Using logarithms we find

$$10 \cdot \log(1.00+i) = \log 2.00$$

or

$$\log (1.00+i) = (\log 2.00)/10$$

I checked my tables and found that log 2.00 = .30103, so log (1.00+i) = .030103 and 1.00+i = 1.0712. Hence you would be be paying 7.12% compound interest, a cheap rate in these days of inflation.

There are two difficulties with the above calculation. First, we might not have logarithms and whatnot available in our programming language; COBOL, for example, has none. Fortunately, Newton's method for finding equation roots uses only basic arithmetic and will work quite well for the interest problem. In particular, if you want to find the n-th root of any positive amount A, make some guess x at the root. Then I guarantee that

$$x' = [A/x^{n-1} + (n-1)x]/n$$

is a better guess at the root. If you don't start with a pretty good guess and if n is big, say 50 or so, this process can take a long time to produce an accurate answer. But repeat it

over and over, each time refining your guess, and eventually you will have as accurate an answer as you want. By the way, you know your answer is accurate enough if

$$|x-x'|/A < e$$

for some small number e (say, e = .00001).

More Complicated Loans

The second difficulty is that real loans are more complicated than our first example. In general, I will agree to loan you L dollars at the beginning of an n-year period. You will agree to pay P_0 dollars back at the beginning of year zero (as we see below, this is not as silly as it sounds), where the numbering of the years starts at zero. At the beginning of year 1 you will pay P_1 dollars, at the beginning of year 2 P_2 dollars . . . . at the beginning of year n–1 P_{n-1} dollars, and at the end of year n–1 (the beginning of year n) P_n dollars. These payments might not all be the same size and some might even be zero dollars. I do assume that the total of the payments is at least as much as the loan amount L and that there are no separate interest payments.

What is the interest rate for this loan with all these funny payments? A straightforward root calculation won't do the job here. Let's assume for the moment that the fair interest rate is known and that it is i% (that is, a bank would give either of us i% compounded yearly for deposits made during the period of the loan). Then if I had put my money L in a bank instead of lending it to you, I could have made

$$A = L\cdot(1+i)^n$$

dollars. Similarly, you could have made

$$P_0(1+i)^n + P_1(1+i)^{n-1} + P_2(1+i)^{n-2} + \ldots + P_{n-1}(1+i) + P_n = B$$

dollars by depositing your payments instead of using them to pay off the loan. So the loan would be worth B dollars to you n years from now and A dollars to me, the amounts we each could have made from a bank. To find the interest, i, all we must do is equate these amounts, A=B, and solve the resulting equation

$$L\cdot(1+i)^n = P_0(1+i)^n + P_1(1+i)^{n-1} + \ldots + P_n$$

for i. Obviously more complicated than our little rootfinder. Luckily, though, a variation of Newton's method, with help from Horner, Birge, and Vieta, still allows a simple arithmetical technique to be used.

The Method

Define a new series of numbers

$$\begin{aligned} Q_0 &= -P_n \\ Q_1 &= -P_{n-1} \\ Q_2 &= -P_{n-2} \\ \ldots &= \ldots \\ Q_{n-1} &= -P_1 \\ Q_n &= L - P_0 \end{aligned}$$

Now take a guess x at the value of (1+i) and form the following double series

$$b_n = c_n = Q_n;$$

for j from n–1 down to 1,

$$\begin{aligned} b_j &= x\cdot b_{j+1} + Q_j \\ c_j &= x\cdot c_{j+1} + b_j; \end{aligned}$$

finally,

$$b_0 = x\cdot b_1 + Q_0.$$

Be sure to start with b_n and work down to b_0, alternating b's and c's. I guarantee that

$$x' = x - b_0/c_1$$

will be a better guess for 1+i. In fact, if you start out with a first guess of

$$1 + [(P_0 + P_1 + \ldots + P_n) - L]/n$$

it should take no more than five or six cycles (often fewer) to find 1+i to four decimal places. Not bad for not knowing any math.

As an example, let's try another variation on the original loan. Assume that I loan you $1.00 and that you agree to repay 20¢ at the end of each of the next 10 years for a total payment of $2.00. Then the table of P's and Q's is

i	P_i	Q_i
0	0¢	−20¢
1	20¢	−20¢
2	20¢	−20¢
3	20¢	−20¢
4	20¢	−20¢
5	20¢	−20¢
6	20¢	−20¢
7	20¢	−20¢
8	20¢	−20¢
9	20¢	−20¢
10	20¢	80¢

Now making a first guess of i = 10% or x = 1.10, the computation for the first cycle is

i	Q_i	b_i	c_i
10	.80	.80	.80
9	−.20	.68	1.56
8	−.20	.548	2.264
7	−.20	.4028	2.8932

Continued on pg. 31

Main Software
Teletype Terminal Interface

BY DONALD FAULKNER
1245 Springdale Dr.
Jackson, Miss. 39211

This software is intended to allow operation of the Digital Group's Z-80 system with an ASCII teletype (printer and keyboard). Basically a send and receive software UART, the routine is written to support teletype operation at 300 baud (alterable with a change in timing constants) with the following software:

1. DGSS Maxi-BASIC Versions 1.1 & 2.0 (Audio)
2. DGSS Z-80 Assemblers I & II
3. DGSS Text-Editor and Super Editor-Formatter
4. DGSS Z-80 Disassembler (true 64-char. version)

Although the routine has been written in such a way as to make it as universal as possible, still some custom tailoring is necessary when incorporating it into any of the above programs.

The successful union of this routine with one of the DGSS programs makes possible the operation of the system from both the regular keyboard (hereafter called the system keyboard) and the teletype terminal. Inclusion of the routine will not disable the system keyboard, since the software continually monitors both the system AND teletype keyboards.

Virtually all of the main software capabilities available from the system keyboard are also available at the teletype. One function which is denied the teletype terminal is the issuing of an ESCAPE command which would place the computer back in the operating system (at the option list) and out of communication with the teletype. This feature prevents the teletype operator from accidentally punching ESC and leaving himself dead in the water. An ESC command is still honored, however, if initiated by the system keyboard.

The teletype operation is full-duplex, i.e., the teletype printer responds not to the operator's key depression, but rather to the character echoed back from the computer in acknowledgement of a character received.

On occasion, the system echoes a character which differs from the received character. A good example of this is the teletype's DEL key. Because in this case, the printer doesn't respond to a DEL code (octal 177) the software instead echoes an UNDERLINE character to the printer to acknowledge receipt of a DEL keystroke. The printing head then prints one UNDERLINE character each time an erasure is caused by pressing the DEL key.

Provision has been made for transmitting a message from the system to the printer when the hardcopy option is first selected. The NORTH JACKSON COMPUTER NETWORK text in the source-code may, of course, be changed to any start-up message desired. The user may wish to use the initialization message to identify the particular piece of software being accessed, e.g., MAXI-BASIC V1.1, etc.

The least significant bits of I/O port no. 3 are used as a serial port for the teletype. Of course, the teletype may (if it is TTL compatible) be connected to the port either directly or through a pair of full-duplex modems. Since modems were used in the development of this routine, a special check has been included to ensure that the input bit at port 3 has not gone to a sustained low (logic 0) level indicating the loss of the normally high (logic 1) level produced by the audio carrier over the interconnecting telephone line. Many modems

are designed to produce a low logic level if the telephone inter-connection is interrupted. If this condition occurs, it will display a '*** NO CARRIER PRESENT ***' message on the video monitor. This message remains on the screen until either: (a) the input bit at port 3 is brought high by the remote modem again becoming active (tone returning to a high state) or, (b) the operator at the computer presses the ESC key. Doing the latter allows the system to escape from the main software program and re-enter the operating system's option list in the normal video-only mode. If the input port goes high again while the no carrier message is on the screen, the message clears and normal software operation may resume. A hard-wired terminal on port 3 will also display the no carrier message if it goes to a sustained logic 0 ('space' in teletype terminology). Note also that the port 3 input bit will have to be brought high to avoid the persistent no carrier message even when the non-hardcopy entry is made into the main program from the operating system's option list.

The octal code produced in the software by a continuous low bit at the input port is 200, the same code with which the operator, when necessary, interrupts an executing Maxi-BASIC, Assembler-II, or Editor-Formatter program (producing, in BASIC, the familiar 'STOP IN LINE XXXX' message). The logic state of the output line of the teletype may be forced low by depressing the teletype's BREAK key. Depressing this key momentarily while running any of the above three programs will stop program execution just as if CONTROL/C had been depressed on the system keyboard. Sustained holding of the BREAK key will cause the no carrier message to display on the video monitor. Depressing the BREAK key wile the programs are not executing will display the no carrier message on the video monitor for the duration of the BREAK key's closure. This, however, is a fully recoverable condition as mentioned above. After releasing the BREAK key, the monitor may be cleared and program execution resumed with any character transmitted by the teletype. This is not seen by the teletype operator since the no carrier message is displayed only at the video monitor. The execution of these programs may be halted from the system keyboard, as well, by the usual CONTROL/C command. However, some other key must be pressed thereafter to allow the teletype operator to continue normal operation, since the conventional keyboard latches the CONTROL/C at its output. Each time thereafter that the keyboard is checked by the altered program, it finds a keyboard interrupt in effect until it is cleared by some other key-stroke.

The auto-numbering system of the Assembler may still be entered by a "!" from the teletype keyboard. The auto-numbering mode is exited by sending an octal 037 (corresponding to a MINUS SIGN pressed while holding down both the shift and control keys . . . at least on a DEC-Writer). If this code is inconvenient, it may be changed by replacing the 000 bytes in the Assemblers with the desired octal code at the locations specified in implementation steps (I) and (I1) which follow. This change of the auto-numbering exit character from the usual DGSS CONTROL/SHIFT/P is necessary because the old character produces an octal 200 code . . . The same code which the software interprets as a 'no carrier' condition.

The DEL key on the teletype keyboard serves as a RUBOUT key. Lines 530-580 of the source code are responsible for echoing a 337 octal code (UNDERLINE symbol) to the printer to acknowledge a successful character erasure. Maxi-BASIC and Assembler-II already echo the necessary character, so lines 530-580 of the source code must be deleted for incorporation of the routine into these programs. Some may prefer to leave the lines intact in the source code and replace the resulting object code with NOP's. This would allow the various subroutines to remain in the same core locations among the various DGSS programs.

The RUBOUT or DEL command is, of course, not honored by either the Text-Editor or Editor-Formatter, which (as in their unmodified states) honor only their own special RUBO characters. The Editors' KILL and RUBO symbols may, as usual, be selected by the user. It will be necessary to select characters which are printable on the teletype keyboard if they are to be successfully echoed to the printer.

Finally, a few additional words about the no carrier feature. As written, this subroutine responds to any break in the teletype-computer link with a no carrier message and then begins a continuous check for re-established communication. At present, a great deal of interest exists in the operation of computer systems through auto-call modem interfaces. An auto-call modem (with the appropriate hardware and software) allows the system to be powered up and initialized by a call from a remote terminal. The system can then be operated from the remote point and later powered down automatically by a command from the remote operator. A necessity, of course, is the provision for powering down the system in the event of an accidental loss of communication. The no carrier subroutine can provide that function by simply incorporating a jump to a shut-down routine after the first half-dozen or so flashes of the '*** CARRIER NOT PRESENT ***' message.

Implementing the Routine

Below is a list of steps required to successfully incorporate the main software–teletype interface routine into the various DGSS programs. Not all steps are required for each software package. Perform only the steps indicated for your package in the following table.

SOFTWARE MODIFIED	STEPS REQUIRED
Maxi-Basic 1 & 2	A, B, C, D, E, F, G, H, J
Z-80 Assembler-I	B, C, D, E, F, H, I, J
Z-80 Assembler-II	A, B, C, D, E1, F1, G1 H1, I1, J, K
Text Editor	B, C, D, E, F, H, J
Editor Formatter	B, C, D, E, F, G2, H, J
Z-80 Disassembler	B, C, D, E, F, H, J

Implementation Steps

A . . . . Delete Source code statements 0530-0580 in the routine prior to assembly . . . (or replace the resulting object code with NOP's).

B . . . Assemble the object code for some arbitrary location (for inclusion in Maxi, Text Editor, etc.). Starting location 006000 is suggested since (for most users) the original hardcopy routine at that location will no longer be used. (The listing indicates assembly at this location).

C . . . . Note the assembled locations of the subroutines labelled as: HCOPY, INIT, ENTRY, and BASTOP.

D . . . . Change the address referred to in Restart-4 of the

main software (location 001014-001015) to the byte and page location of the HCOPY subroutine.

E . . . . Change the address called in the main software (at location 011326-011327) for purposes of hardcopy initialization to the assembled byte and page location of INIT.

E1 . . . Change the address called in the main software (at location 011331-011332) for purposes of hardcopy initialization to the assembled byte and page location of INIT.

F . . . . Change the keyboard call in the main software (at location 011253-011254) to the byte and page location of the assembled location of ENTRY.

F1 . . . Change the keyboard call in the main software (at location 011260-011261 to the byte and page location of the assembled location of ENTRY.

G . . . . Replace the 5-byte-long routine beginning at 017000 in the main software (only in Maxi-Basic) with a jump to the assembled location of BASTOP.

G1 . . . At location 021330-021337 in the main software, place the following (octal) code: 333 000 376 003 303 BBB PPP 000 (where BBB and PPP represent the assembled byte and page location of the BASTOP subroutine). Replace the eleven bytes of the assembled BASTOP routine with the following (octal) code: 050 005 333 003 313 107 300303 000 016 000.

G2 . . . Replace the seven bytes at locations 030152 through 030160 with a JUMP to the assembled location of the BASTOP subroutine.

H . . . Load octal 000 codes at locations 011276 through 011302 in the main software to prohibit acceptance of an ESC command from the teletype terminal.

H1 . . . Load octal 000 codes at locations 011275 through 011301 in the main software to prohibit acceptance of an ESC command from the teletype terminal . . . . . .

I . . . Replace the octal 000 bytes at locations 047261 and 054205 (only in the Z-80 Assembler-I) with octal 037. (This changes the character used to exit from the auto-numbering system).

I1 . . . Replace the octal 000 bytes at location 017341 in Assembler-II with octal 037. (This changes the character used to exit from the auto-numbering system).

J . . . . Modify the wording of the operating system's hardcopy option, if desired, to reflect the inclusion of the new routine.

K . . . At location 011046-011047 in the main software, place (octal) bytes 330-011. This will allow the Assembler to remain in the hardcopy mode on issuance of a NEWF command rather than jumping into the video-only mode.

Note. . .the baud rate for the routine may be altered by modifying the timing constants in the source-code at lines 0560, 0570, 0800, and 0810. Course adjustment is made in lines 0560 and 0800 . . . Fine adjustment in lines 0570 and 0810.

Listing begins on next page

Continued from pg. 43

A RESPONSE TO "ON COMPUTER CRIME BILL S240"

Dear *Dr. Dobb's*:

As a computer professional and an educator, I would like to make a few comments on the article "On Computer Crime Bill S240" by Jon K. Taber (*DDJ* # 37). The main points of Taber's article can be summarized as follows:

1. Computer crime is not prevalent enough to be singled out for legislative action;
2. The proposed legislation is a Federal intrusion into State matters;
3. The FBI and the Justice Department cannot be trusted;
4. "Prison Programming" is open to abuse through the creation of inequitable working conditions;
5. Certain common practices such as game-playing on an employer's computer, that are, at best, perquisites and, at worst, unethical, are outlawed by Bill S240;
6. The Bill does not discriminate between "minor" and "major" offenses.

I am not in a position to comment on points 2, 3, and 4 above, but as an educator and a computer professional I am appalled by the tenor of the article and, especially, the statement

> "Generally speaking, computer professionals are technically oriented and do not know or care how laws and sausages are made."

This is an admission of a belief that "most programmers (and many other computer users)" form an elite group that is almost a law unto themselves. A sad admission, I think, in this Post-Watergate period. If my evaluation is correct, then our education system has failed to produce the socially responsible computer person and Bill S240 is needed to protect the community at large from an arrogant group which only pays lip service to ethics.

What is or is not a case of computer abuse may be an arguable matter, but I personally like the following (A.D. Chambers, *The Computer Journal*, *21*, No. 3 (August 78), pp. 194-197), "By computer abuse is meant:

1. The misappropriation of goods, services, or money making use of the computer system.
2. The manipulation of computer accounts not involving defalcation.
3. The misappropriation of one's employer's computer-based information and disposal of it for gain (e.g., accounts, profits forecasts, tenders, computer programs, etc.).
4. The manipulation of equipment or systems of an organization so as to place the organization at some kind of disadvantage."

The fact is that the impact of computers on society will be large and many-faceted and it is our duty as computer professionals to instill confidence in the general public that we are responsible people who will not abuse the position of trust in which we *choose* to place ourselves.

Bill S240 may be inadequate on legal grounds and it may be too sweeping in its penalties, but as we, the computer professionals, have not been able to show Society that we can be trusted not to think of ourselves first, then the provisions of Bill S240 do not appear to be overly excessive.

Sincerely,
Prof. N. Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ont., Canada
L8S 4K1

```
006000                          0100 ************************************************************
006000                          0110 *
006000                          0120 *       MAIN SOFTWARE - TELETYPE TERMINAL INTERFACE
006000                          0130 *             SERIAL I/O TO LSB OF PORT #3
006000                          0140 *
006000                          0150 *                WRITTEN  12-11-77
006000                          0160 *                MODIFIED  2-5-78
006000                          0170 *                MODIFIED  4-2-78
006000                          0180 *                MODIFIED  4-7-78    ... D. FAULKNER
006000                          0190 *
006000                          0200 ************************************************************
006000                          0210 MONTR  EQU  000000 * VECTORS FORWARD TO SYSTEM MONITOR
006000                          0220 HOME   EQU  000346
006000                          0230 TVEDT  EQU  002000 * PHIMON USERS REPLACE THIS WITH 343210
006000 305                      0240 ENTRY  PUSH BC     * ENTRY POINT FOR KEYBOARD CALL
006001 325                      0250        PUSH DE
006002 345                      0260        PUSH HL
006003 333 000                  0270 STBYT  IN   0      * CHECK KEYBOARD STROBE
006005 376 033                  0280        CP   033    * EXIT TO  OP. SYS. ON 'ESC'
006007 312 000 000              0290        JP   Z,MONTR
006012 313 177                  0300        BIT  7,A
006014 040 076                  0310        JR   NZ,KEY * KEYBD ACTIVE - JP TO KEYBD INPUT
006016 021 010 000              0320        LD   DE,000010
006021 333 003                  0330        IN   3      * CHECK TELETYPE PORT FOR START-BIT
006023 313 107                  0340        BIT  0,A
006025 040 354                  0350        JR   NZ,STBYT
006027 006 004                  0360        LD   B,4
006031 016 263                  0370 WSYNC  LD   C,263  * DELAY UNTIL CENTER OF LSB
006033 015                      0380        DEC  C
006034 040 375                  0390        JR   NZ,WSYNC+2
006036 020 371                  0400        DJNZ WSYNC
006040 333 003                  0410 GDATA  IN   3      * SAMPLE DATA BIT
006042 306 002                  0420        ADD  2      * CLEAR ALL BITS BUT LSB FROM PORT
006044 202                      0430        ADD  D
006045 127                      0440        LD   D,A    * ADD NEW BIT TO BYTE FORMING IN D-REG
006046 313 012                  0450        RRC  D      * PREP D-REG FOR NEXT BIT FROM PORT
006050 315 102 006              0460        CALL WDATA
006053 035                      0470        DEC  E
006054 040 362                  0480        JR   NZ,GDATA  * SAMPLE NEXT BIT
006056 313 372                  0490        SET  7,D    * SET PARITY BIT TO 1
006060 172                      0500        LD   A,D
006061 376 200                  0510        CP   200    * CHECK FOR CARRIER LOSS
006063 050 163                  0520        JR   Z,MESS
006065 376 377                  0530        CP   377    * CHECK FOR TTY 'DELETE' (SOFTWARE R/O)
006067 040 033                  0540        JR   NZ,RET
006071 365                      0550        PUSH AF     * SAVE A-REG. BEFORE JUMP TO XMIT
006072 026 337                  0560        LD   D,337  * LD D-REG. WITH 'UNDERLINE' SYMBOL
006074 315 201 006              0570        CALL XMIT   * ECHO 'UNDERLINE' TO TTY TERMINAL
006077 361                      0580        POP  AF
006100 030 022                  0590        JR   RET
006102 006 004                  0600 WDATA  LD   B,4    * DELAY TO CENTER OF NEXT BIT
006104 016 177                  0610        LD   C,177
006106 015                      0620        DEC  C
006107 040 375                  0630        JR   NZ,WDATA+4
006111 020 371                  0640        DJNZ WDATA+2
006113 311                      0650        RET
006114 333 000                  0660 KEY    IN   0      * KEYBD ACTIVE - SEND A CHARACTER
006116 313 177                  0670        BIT  7,A
006120 040 372                  0680        JR   NZ,KEY
006122 313 377                  0690        SET  7,A    * SET PARITY BIT TO 1
006124 341                      0700 RET    POP  HL     * RETURN TO MAIN FROM KEYBOARD CALL
006125 321                      0710        POP  DE
006126 301                      0720        POP  BC
006127 311                      0730        RET
006130 365                      0740 HCOPY  PUSH AF     * ENTRY POINT FOR MAIN'S HARDCOPY CALL
006131 305                      0750        PUSH BC     * SAVE REGISTERS OF CALLING ROUTINE
006132 325                      0760        PUSH DE
006133 345                      0770        PUSH HL
006134 127                      0780        LD   D,A    * PUT ASCII CHAR. IN D-REG FOR XMIT
006135 006 000                  0790        LD   B,0
006137 260                      0800        OR   B
006140 352 145 006              0810        JP   PE,PAREV
006143 313 372                  0820 PAROD  SET  7,D
006145 172                      0830 PAREV  LD   A,D    * CHECK FOR CONTROL CHARACTERS
006146 376 215                  0840        CP   215
006150 050 005                  0850        JR   Z,CARRT
006152 315 201 006              0860        CALL XMIT   * CARR/RETURN NOT REQUIRED - XMIT CHAR.
006155 030 015                  0870        JR   HCRET
006157 315 201 006              0880 CARRT  CALL XMIT   * TRANSMIT CARR-RETURN
006162 026 212                  0890        LD   D,212  * GENERATE LINE-FEED FOR TTY
006164 315 201 006              0900        CALL XMIT
006167 026 223                  0910        LD   D,223  * GENERATE X-OFF AS TIME KILLER
006171 315 201 006              0920        CALL XMIT
006174 341                      0930 HCRET  POP  HL     * RETURN TO MAIN FROM HARDCOPY CALL
006175 321                      0940        POP  DE     * RESTORE REGS. OF CALLING ROUTINE
006176 301                      0950        POP  BC
006177 361                      0960        POP  AF
006200 311                      0970        RET
006201 257                      0980 XMIT   XOR  A      * TRANSMIT SERIAL CHARACTER TO TELETYPE
006202 323 003                  0990        OUT  3      * SEND START-BIT
006204 315 102 006              1000        CALL WDATA
006207 036 010                  1010        LD   E,8D   * PREP TO XMIT 8 DATA BITS
006211 313 012                  1020 SHIFT  RRC  D      * CHECK EACH BIT FOR 0 OR 1
006213 070 027                  1030        JR   C,ONE
006215 257                      1040        XOR  A
006216 323 003                  1050 OUT    OUT  3      * SEND 0 BIT IF NOT-CARRY
006220 315 102 006              1060        CALL WDATA
006223 035                      1070        DEC  E
006224 040 363                  1080        JR   NZ,SHIFT  * PREP NEXT BIT
006226 036 002                  1090        LD   E,2    * SEND TWO "1" STOP BITS
006230 076 001                  1100        LD   A,1
006232 323 003                  1110        OUT  3
006234 315 102 006              1120 STOBT  CALL WDATA  * STOP-BIT TIMING LOOP
006237 035                      1130        DEC  E
006240 040 372                  1140        JR   NZ,STOBT
006242 172                      1150        LD   A,D
006243 311                      1160        RET
006244 076 001                  1170 ONE    LD   A,1
006246 030 346                  1180        JR   OUT
006250 315 346 000              1190 MESS   CALL HOME   * PRINT "CARRIER LOST" MESSAGE
006253 041 346 006              1200        LD   HL,MESS1
006256 315 000 002              1210        CALL TVEDT
006261 076 010                  1220        LD   A,010
006263 315 325 006              1230        CALL SECS
006266 315 346 000              1240        CALL HOME
006271 303 003 006              1250        JP   STBYT
006274 333 000                  1260 BASTOP IN   0      * MAXI'S INTERRUPT CHECK DURING EXEC.
006276 376 003                  1270        CP   003    * CHECK FOR KEYBOARD INTERRUPT (CTL/C)
006300 050 004                  1280        JR   Z,HALT
006302 333 003                  1290        IN   3
006304 313 107                  1300        BIT  0,A    * CHECK TTY FOR INTERRUPT
006306 311                      1310 HALT   RET
006307 041 004 007              1320 INIT   LD   HL,INTMS * TTY INITIALIZATION ROUTINE
006312 176                      1330        LD   A,(HL)
006313 315 130 006              1340 LOOP   CALL HCOPY  * OUTPUT INITIALIZATION MESSAGE
006316 043                      1350        INC  HL
006317 176                      1360        LD   A,(HL)
006320 376 000                  1370        CP   0
006322 040 367                  1380        JR   NZ,LOOP
006324 311                      1390        RET
006325 026 031                  1400 SECS   LD   D,031  * TIME DELAY ROUTINE REMOVED FROM DG
006327 001 003 000              1410 SEC1   LD   BC,000003 * OPERATING SYSTEM... FOR MAXI-2
006332 020 376                  1420 SEC2   DJNZ SEC2          * PHIMON VERSION... WHICH HAS
006334 015                      1430        DEC  C                * NO SYS/MONITOR INCLUDED.
006335 040 373                  1440        JR   NZ,SEC2
006337 025                      1450        DEC  D
006340 040 365                  1460        JR   NZ,SEC1
006342 075                      1470        DEC  A
006343 040 360                  1480        JR   NZ,SECS
006345 311                      1490        RET
006346 177                      1500 MESS1  DB   177
006347 103                      1510        DB   103
006350                          1520        DW   '*** CARRIER NOT PRESENT ***'
 252 252 252 240 303
 301 322 322 311 305
 322 240 316 317 324
 240 320 322 305 323
 305 316 324 240 252
 252 252
007003 000                      1530        DB   0
007004 215                      1540 INTMS  DB   215    * INITIALIZATION MESSAGE
007005 212                      1550        DB   212
007006 212                      1560        DB   212
007007                          1570        DW   '***** NORTH JACKSON '
 252 252 252 252 252
 240 316 317 322 324
 310 240 312 301 303
 313 323 317 316 240
007033                          1580        DW   'COMPUTER NETWORK *****'
 303 317 315 320 325
 324 305 322 240 316
 305 324 327 317 322
 313 240 252 252 252
 252 252
007061 215                      1590        DB   215
007062 212                      1600        DB   212
007063 000                      1610 FINIS  DB   0


SYMB
MONTR  000000   HOME   000346   TVEDT  002000   ENTRY  006000
STBYT  006003   WSYNC  006031   GDATA  006040   WDATA  006102
KEY    006114   RET    006124   HCOPY  006130   PAROD  006143
PAREV  006145   CARRT  006157   HCRET  006174   XMIT   006201
SHIFT  006211   OUT    006216   STOBT  006234   ONE    006244
MESS   006250   BASTOP 006274   HALT   006306   INIT   006307
LOOP   006313   SECS   006325   SEC1   006327   SEC2   006332
MESS1  006346   INTMS  007004   FINIS  007063
```

COMMON INSTRUCTIONS

OF THE 6800 AND 6502

BY B. T. G. TAN
Dept. of Physics
University of Singapore
Singapore 10

The 6800 and 6502 microprocessors share many common features in their architectures, instruction sets, busses, and general system philosophy. I have compiled for teaching purposes a list of their common instructions and their addressing modes. This list is used in conjunction with a programming model which shows only the two MPUs' common features (Fig. 1). The programming model has an 8-bit accumulator, index register, stack pointer and status register and a 16-bit program counter. Only 5 bits of the status register are used: carry/borrow, overflow, zero, negative and interrupt mask.

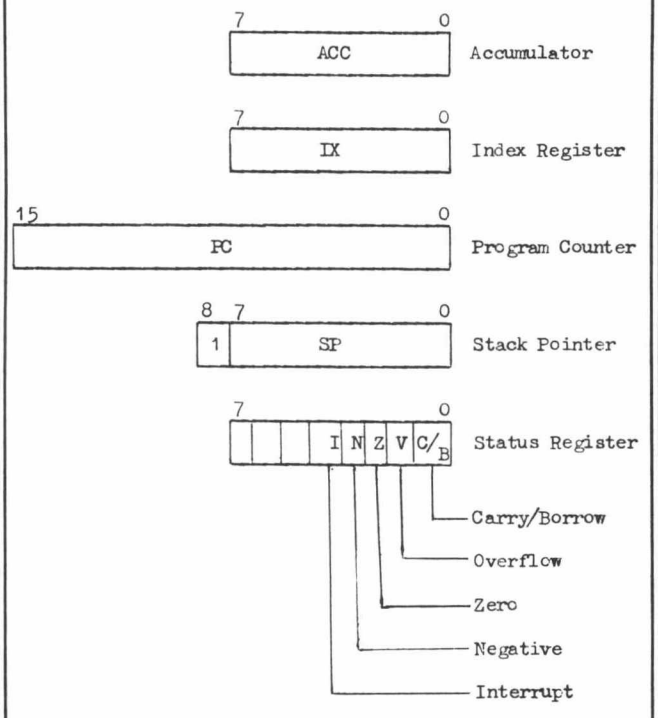


Figure 1. Programming Model

There are 43 instructions in the list (Fig. 2) common to the 6800 and 6502, which form a viable instruction set. This set is useful in the teaching of the two MPUs as well as in defining a common assembly language for them (In my microcomputer classes, I have given the name "5800" to this common microprocessor model).

	IMM	DIR	EXT	IMP	REL	IND
ADC	x	x	x			x
AND	x	x	x			x
ASL/ASLA			x	x		x
BCC					x	
BCS					x	
BEQ					x	
BIT		x	x			
BMI					x	
BNE					x	
BVC					x	
BVS					x	
BPL					x	
CLC/CLB				x		
CLI				x		
CLV				x		
CMP	x	x	x			x
CPX	x	x	x			
DEC			x			x
DEX				x		
EOR	x	x	x			x
INC			x			x
INX				x		
JMP			x			
JSR			x			
LDA	x	x	x			x
LDX	x	x	x			
LSR/LSRA			x	x		x
NOP				x		
ORA	x	x	x			x
PSH				x		
PUL				x		
ROL/ROLA			x	x		x
ROR/RORA			x	x		x
RTI				x		
RTS				x		
SBC	x	x	x			x
SEC/SEB				x		
SEI				x		
STA		x	x			x
STX		x	x			
SWI				x		
TSX				x		
TXS				x		

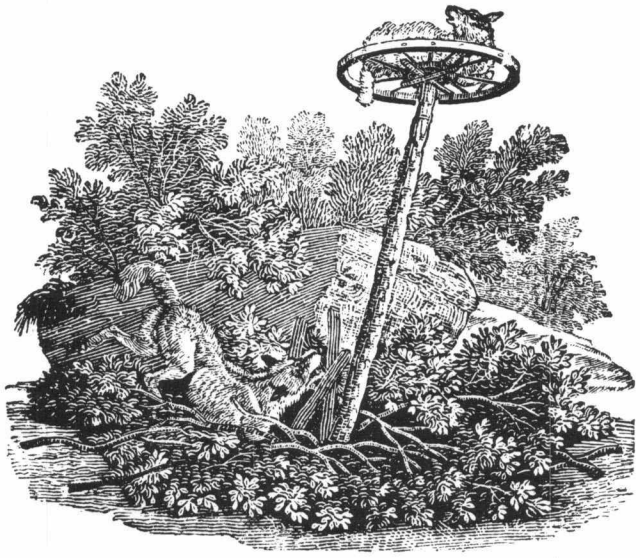
Figure 2. Instruction Set

Certain points should be noted about this instruction set:

1. The mnemonics used are generally those of the 6800, with the A or B accumulator specification omitted. For the 6502, PSH, PUL and SWI correspond to PHA, PLA and BRK respectively.
2. There are six addressing modes: immediate, direct, extended, implied, relative and indexed, as for the 6800. For the 6502 direct, extended and indexed correspond to zero page, absolute and zero page indexed respectively.

3. The indexed mode is restricted to a zero page addressing range, with a one-byte operand and an 8-bit index register. For the 6800, "00" should be loaded into the high byte of the 16-bit index register to simulate an 8-bit index register.
4. The 8-bit stack pointer operates like that of the 6502; its contents are always preceded by hex 01, so that the stack is always in the range hex 0100 to 01FF. For the 6800, hex 01 should be loaded into the high byte of the 16-bit stack pointer.
5. The borrow bit and the SBC instruction operate differently for the 6800 and 6502. For the 6800, the borrow bit is equal to the carry bit while for the 6502 it is the inverse of the carry bit. Two "pseudo- instructions" have been added to the instruction set: clear borrow bit (CLB) and set borrow bit (SEB). These are to be used in conjunction with the SBC instruction. For the 6800, CLB and SEB are replaced by CLC and SEC respectively, while for the 6502 the reverse is peformed, *i.e.* SEC replaces CLB and CLC replaces SEB. This enables us to add SBC to the list of common instructions.

The list and programming model have been useful to me both in teaching and writing source programs that will run on both MPUs. I have prepared for my students a more comprehensive data sheet that also gives the respective op codes for the 6800 and 6502, which I will be happy to send to interested readers (the new Motorola 6805 microcomputer, strangely enough, resembles the above model in some respects).



Letters

EVEN *MORE* ABOUT AF

To *Dr. Dobb's Journal*:

I was intrigued by Mike Gabrielson's article on Ackermann's function in *DDJ* #31. I have found analytic expressions for A(m,n) for m = 0, 1, 2, 3, and an iterative expression for m = 4. This allows filling out the table of values for m = 0 to m = 4 (though the latter has large numbers).

First, let me restate the definition of the function, quoted by Gabrielson from D.W. Barron:

$$A(m,n) = \begin{cases} m=0 \rightarrow n+1, \\ n=0 \rightarrow A(m-1,1), \\ A(m-1,A(m,n-1)) \end{cases}$$

For m=0, $A(0,n) = n + 1$

For m=1, $A(1,n) = A(0,A(1,n-1)) = A(1,n-1)+1$
also, $A(1,0) = A(0,1) = 2$
from these results,

$A(1,n) = n+2$

For m=2, $A(2,n) = A(1,A(2,n-1)) = A(2,n-1)+2$
also, $A(2,0) = A(1,1) = 3$
from these results,

$A(2,n) = 2n + 3$

For m=3, things begin to get interesting:
$A(3,n) = A(2,A(3,n-1)) = 2A(3,n-1) + 3$
from this, $A(3,n) + 3 = 2\,(A(3,n-1) + 3)$
i.e., we have powers of 2: $A(3,n) + 3 = 2^{n+b}$
Now . $A(3,0) = A(2,1) = 5$, which allows us to find b:
$A(3,0) + 3 = 8 = 2^{b}$, hence b=3.

$A(3,n) = 2^{n+3} - 3$

For m=4, $A(4,n) = A(3,A(4,n-1)) = 2^{A(4,n-1)+3}-3$
also, $A(4,0) = A(3,1) = 13$

At this point, the notation failed me, so I will give a few values:

$A(4,0) = 13$
$A(4,1) = 2^{16}-3 = 65533$

$A(4,2) = 2^{(2^{16})}-3$ (the parentheses are necessary to avoid ambiguity)

$A(4,3) = 2^{(2^{(2^{16})})}-3$

$A(4,4) = 2^{(2^{(2^{(2^{16})})})}-3$

You can see that A(4,n) grows v-e-r-y rapidly for increasing n. I doubt that there is enough memory or processor time in the world to allow A(4,3) to be calculated recursively using the definition of Ackermann's function. For bigger numbers, try m = 5:

$A(5,n) = A(4,A(5,n-1))$
also, $A(5,0) = A(4,1) = 65533$
hence, $A(5,1) = A(4,65533) = 2^{(2^{(2^{(\cdots^{(2^{16})})})})}-3$ (65533 '2's)

I leave it to others to tackle bigger m and n.

Sincerely,
Peter Traneus Anderson

103 N. Union St.
Burlington, VT 05401

PROCESSOR TECH, ETCETERA

To the Editor:

First off, I would like to thank you for publishing 3 source listings: "A Compact 8080 Disassembler Dumper" by Richard Greenlaw, "POW: an 8080 Processor of Words" by Herman Watson", and most of all "Falconer Floating Point" by Charles Falconer. It would have cost me a fortune to buy the SOURCES for programs like those, so your magazine (at least in that respect) has paid for itself many times over. With "Falconer Floating Point," after writing a SIN/COS routine (in which the algorithms came from another *Dr. Dobb's* article) I have been able to write graphics design programs that are 20 to 30 times faster than their BASIC equivalents! Keep the good sources coming!

Also I (and I'm sure many readers) would like to know exactly what has happened to the Processor Technology Corporation. They no longer advertise or even answer their phone! The dealers aren't too specific about what's going on either. I have heard lots of rumors (they were bought out, they folded, etc.) but no facts. I would really appreciate it if you would fill us all in.

Sincerely,
John Peterson

1820 Camino Drive
Forest Grove OR 97116

Processor Technology Corporation is no longer in existence: the doors are closed, the phone has been disconnected. Supposedly, Proteus, the Processor Technology Users Society, will keep things together for users. Contact Proteus at 1690 Woodside Road, Suite 219, Redwood City, CA 94061.

"VICE-VERSA" CORRECTIONS

Dear Suzanne:

Keep up the good work. *Dr. Dobb's* continues its excellent quality. I hope you are getting our club newsletter.

Here are some corrections for the "Vice-Versa" program, pgs. 26-29, *DDJ*, # 33.

1. At address 335 change "JMP BDOS" to "CALL BDOS"
2. Add "WRITNAME EQU WRITFCB+1"

Sincerely,
Don & Sol Libes

Amateur Computer Group of New Jersey
1776 Raritan Rd. Scotch Plains, N.J.07076

CONVERTING TO JULIAN

In our area of Project Management Information System, we do a lot of reports by Calendar or Julian dates with the Department of National Defense. The reports are keyed on due dates, target dates, etc. It is much easier to work with Julian dates than with Calendar dates. For example, to find how many projects are due in the next 14 days, you must ensure you either stay within the month or add 1 to the month. However, with Julian dates, all you need to do is add 14.

For those who have home computers, such a conversion would be useful in determining your next mortgage payment due date, next car checkup, when to buy certain staples, etc.

The Julian calendar starts on January first with 001 and ends on December 31 with 365 or 366, depending on leap year. The little program below (in GIS) is made to convert Calendar days to Julian days for years with 365 days. To use it for leap year, add 1 to each day from March to December; i.e., in March 'Decrease JULDATE BY 240' rather than 'Decrease JULDATE BY 241',etc.

Although written in GIS, the program is so general that it can be used in most other languages with little effort. Have fun! I did.

Sincerely
Jack Dekok

2129 Benjamin
Ottawa, Canada KLA 1P3

```
   CHANGE CALDATE TO JULDATE
        IF CALDATE BT 780000, 790000
             CHANGE DATSPC TO 78
             DECREASE CALDATE BY 780000
                  GO TO STEP 1
        IF CALDATE BT 790000, 800000
             CHANGE DATSPC TO 79
             DECREASE CALDATE BY 790000
                  GO TO STEP 1
STEP 1  IF CALDATE BT 0100, 0131
             DECREASE CALDATE BY 100
        IF CALDATE BT 0200, 0231
             DECREASE CALDATE BY 169
        IF CALDATE BT 0300, 0331
             DECREASE CALDATE BY 241
        IF CALDATE BT 0400, 0431
             DECREASE CALDATE BY 310
        IF CALDATE BT 0500, 0531
             DECREASE CALDATE BY 380
        IF CALDATE BT 0600, 0631
             DECREASE CALDATE BY 449
        IF CALDATE BT 0700, 0731
             DECREASE CALDATE BY 519
        IF CALDATE BT 0800, 0831
             DECREASE CALDATE BY 588
        IF CALDATE BT 0900, 0931
             DECREASE CALDATE BY 657
        IF CALDATE BT 1000, 1031
        IF CALDATE BT 1100, 1131
             DECREASE CALDATE BY 796
        IF CALDATE BY 1200, 1231
             DECREASE CALDATE BY 866
        IF DATSPC EQ 78
             INCREASE CALDATE BY 780000
        IF DATSPC EQ 79
             INCREASE CALDATE BY 790000
                  GO TO H1
                     ?
```

MORE HELP WITH THE TRS-80

Dear Ed,

DDJ # 36 contains a letter from Greg Perry concerning the direct access to the keyboard from programs in the TRS-80 level II. It should be pointed out that it is not necessary to use U and D for up and down when using the INKEY function, since this will also return a value for these keys. However, the ASCII values returned are not exactly as stated in the TRS-80 level II reference manual. The up, down, left and right arrows will return 91, 10, 9 and 8 without the shift key depressed, and 27, 26, 25 and 24 if the shift key is depressed. The clear key will return ASCII value 31 (irrespective of shift), and entering will give value 13. If you are lucky, you can also get the break key in with value 1.

If, however, you need to know how long a key is depressed, or if two keys can be held down at the same time, then INKEY is not very helpful. Here the direct access to the keyboard can be useful. Eight locations will have to be accessed if all keys are required, with or without shift, and if several keys are depressed at the same time then the values are OR'ed together if they share a common location. They keyboard area starts at location 14336 (3800H), and for some reason uses a complete 1K of memory. The following table may be useful for anyone wishing to use this possibility for direct keyboard access. There is no guarantee that the table is complete or correct for all systems, but it functions on my system. The first column is the displacement of the first byte from the beginning of the keyboard area, and at the same time the number of repeats before the next break. For example, the keys in the row with value 8 do not affect bytes 0 to 7 in the area, will set a value in bytes 8 to 15, do not affect 16 to 23, etc. The heading is the bit value set by the particular key in the affected bytes. The individual keys can be separated by using the AND function with the value at the head of the column. An example in BASIC:

```
10  IN%=PEEK (14336+64)

20 IF IN% AND 8 THEN PRINT "UP"

30 IF IN% AND 16 THEN PRINT "DOWN"
```

Table of key codes for TRS-80 level II

Position (relative to 14336)	Value 1	2	4	8	16	32	64	128
1	@	A	B	C	D	E	F	G
2	H	I	J	K	L	M	N	O
4	P	Q	R	S	T	U	V	W
8	X	Y	Z					
16	0	1	2	3	4	5	6	7
32	8	9	:	;	,	-	.	/
64	Ent	Clr	Brk	↑	↓	←	→	Space
128	Shift							

I hope that others may also find this table useful. It seems an oversight on Radio Shack's part that it has not been included in the reference manual for the TRS-80. Another oversight is the table of graphic characters, but these are slightly easier to find. A simple method for remembering them is by numbering the 'pixels' from left to right, top to bottom, and the bits in a byte from 1 to 8 counting from right to left. Element 1 is controlled by bit 1, element 2 by bit 2, . . . , bits 7 and 8 are 0 and 1 respectively.

Yours faithfully,
Arne Rohde

Pilevej 31
7600 Struer Denmark

MORE ITHACA AUDIO Z-80 MODS

Dear *DDJ:*

I have just been given *Dr. Dobb's* #35 and wish to comment on the Ithaca Audio Z-80 board modification discussed on page 50.

The +5V regulator on this board is extremely overworked (under-designed?). Mr. Sargent's modification only aggravates the situation by placing an additional 100 mA load on the poor thing. He does not take advantage of the low power standby mode of the Intel 2716; he does not even remove the heat producing -5V and +12V zener diodes, which are no longer needed after the modification. His claim that five modified boards "work just fine" is meaningless; too many poorly engineered products (claimed to "work just fine") are available on the S-100 market.

The modification is not quite as simple as Mr. Sargent suggests, but it is feasible. I recommend the following steps:

1. Cut traces leading to U23, pins 4 and 11. These are the two traces emerging from under the right side of the U23 socket, on the component side of the board.
2. Cut trace to U23 pin 12, on the component side.
3. Cut trace to U36 pin 20. This must be done on the component side of the board, *under* the U36 socket. TI sockets (as well as some other makes) let one pry off the plastic part of the socket, leaving all pins intact, thus facilitating this delicate operation. At this time, check to see that no trace connects to pin 18. If any trace does connect to pin 18, cut it.
4. Cut trace to U36 pin 19, on the print side of the board.
5. On the print side of the board, connect the trace which has just been disconnected from U36 pin 20, to U36 pin 18.
6. Jumper U30 pin 1 to U29 pin 5.
7. Jumper U36 pin 19 to U30 pin 40.
8. Jumper U36 pin 20 to U37 pin 8.
9. Remove and discard R3, R4, D1, D2 and C3.
10. Remove C2 and reinstall with reversed polarity. Correct the silk screen on the board to reflect this change.
11. Jumper the "hot" side (lower lead) of C5 to the hole vacated by D2 anode.
12. Replace the 2708 with an Intel 2716, TMS2516 or a similar EPROM.
13. Replace three 8T98 ICs (U31, U37 and U39) with 74LS368 ICs. This saves about 150 mA (using Max. specs).
14. If the system will tolerate it, replace eight 8T97 ICs (U3, U4, U11, U20, U26, U27, U33 and U40) with 74LS367 ICs. My system, which uses a 15 slot, actively terminated Integrand mother-board at 4 MHz, does not mind, but I am driving only three boards.

Incidentally, all the above refers to a Rev. 1.3 board, which I have modified per these instructions, and which "works just fine."

Sincerely yours,
R. A. Hoffman

9 Greenway Court
Brookline, MA 02146

INTERFACING AN H-14 TO A SOL

Dear *DJJ*:

When I got my H-14 printer from Heathkit I had the opportunity to create an interface between the printer and my SOL. Here are the results.

There are two types of data available from the H-14 which can be used for handshaking between the two machines:

1. The REQUEST TO SEND and DATA TERMINAL READY signals on pins 4 and 20 of the H-14 EIA connector
2. The control characters transmitted by the H-14 to indicate its status.

My method uses the first approach.

Since the SOL serial port and the H-14 are both configured as data terminals, the H-14 cable cannot be plugged directly into the SOL. I made up an adapter as follows:

Male "D" connector	Female "D" connector
1	1
2	3
3	2
5	4
6	20
7	7

(The wire from male pin 3 to female pin 2 is not essential, since the H-14's control character transmissions will be ignored. Since the data is available, I decided to bring it into the SOL anyway and deal with its effects in the software. You never can tell: someday I may find a use for these characters.)

The male "D" connector plugs into the SOL serial port; the female connector plugs into the H-14 cable. Set the baud rates of the two machines to be equal. Set the SOL's serial word definition switches to NO PARITY, 1 STOP BIT, and 8-BIT DATA WORD LENGTH.

At speeds of 110 and 150 baud the SOLOS serial output driver routine SDROT (located at CO4A in SOLOS) is adequate to handle this setup. Output pseudoport 1 can be used to invoke SDROT without losing any characters, since the H-14 line buffer will never be completely filled. At 300 baud and above, however, SDROT will not work since it will continue to send characters to the H-14 even when the H-14 line buffer is full.

To overcome this problem you should enter the routine in listing #1 into the SOL system RAM. Notice that the ANI OAFH instruction strips off the SERIAL OVERRUN ERROR and SERIAL DATA READY signals at the serial status port. SDR is used only for serial input, and SOE is a product of those unused CTRL-Q and CTRL-S characters from the H-14.

Next, go into SOLOS command mode and type:

> SE CO C900 <CR>

Last, to use the printer as the output device toggle the LINE/LOCAL key to LINE. Specify the SOL output pseudoport as port 3. All subsequent output from the SOL will be printed by the H-14.

A useful variation from the routine in listing #1 can be obtained by changing the RET instruction to JMP VDMOT (C3 54 CO). All output will appear at both the H-14 and the video monitor.

LISTING #1

```
C900   DB F8        L0 IN     SERST
C902   E6 AF           ANI    0AFH
C904   FE A1           CPI    0A1H
C906   C2 00 C9       JNZ    L0
C909   78             MOV    A,B
C90A   D3 F9           OUT    SDATA
C90C   C9              RET
```

Sincerely,
Stephen Grant

1310 Broughton Dr.
Beverly, MA 01915

ALTERING DOS

Dear *Dr. Dobb's*:

Listed below are some alterations to the NorthStar DOS that might interest some people. They permit the directory listing to be presented two columns wide instead of one and allow the listing of the 'goto' data for file types other than 1. The changes are given for both Release/.4and for Release/.5 and coding is given for both 64 character lines and 80 character lines.

If you have activated the 'paging mode,' the number of lines must be doubled as the DOS will count names instead of lines.

Sincerely,
Amberse M. Banks
Mathematics Department

Fred C. Beyer High School
1717 Sylvan Avenue, Modesto, CA

A. To change the 'goto' listing so it appears for types 0, 1 and 2.

```
Rel.4   was:  26Ø1  FE Ø1          change to:  FE Ø3
              26Ø3  C2 11 26                   D2 EE 29 (or 11 26)

Rel.5         2579  FE Ø1                      FE Ø3
              257B  C2 89 25                   D2 EE 29  (89 25?)
```

The 29EE address assumes that you are going to the two-column format.

B. To change to the two-column format.

```
Rel.4   was:  2611  CD ØC 27       change to:  CD F6 29

Rel.5         2589  CD FD 25                   CD F6 29
```

C. The coding that will space over properly is given for Release.4 with a 64-column format and for Relase.5 with 80 columns. The numbers can be changed of course.

```
Rel.4                29EE  Ø E ØC          MVI   C,12
64-character         29FØ  CD F8 29        CALL  XXX
    screen           29F3  C 3 14 26       JMP   (back to the DOS)

                     29F6  Ø6 Ø7           MVI   C,7
                     29F8  CD Ø4 27  XXX    CALL  (print a blank)
                     29FB  ØD              DCR   C
                     29FC  C2 F8 29        JNZ   XXX
                     29FF  C9              RET

Rel.5                29EE  0E 12           MVI   C,18
80-character         29FØ  CD F8 29        CALL  XXX
    screen           29F3  C3 8C 25        JMP   (back to the DOS)

                     29F6  ØE ØD           MVI   C,13
                     29F8  CD F5 25  XXX    CALL  (print a blank]
                     29FB  ØD              DCR   C
                     29FC  C2 F8 29        JNZ   XXX
                     29FF  C9              RET
```

Continued on pg. 36

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 40 November/December 1979 Volume 4, Issue 10

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

Some Improvements of 8080 Pilot 4
Tim Scully

DAL: Data Acquisition Language 26
Gregg Marshall

An Introduction to North Star Disc STOIC 20
B. W. Lee

Complex Pseudorandom Sequences from Interlaced Simple Generators 33
H. T. Gordon

Dumping Files Under Flex 36
Freeman Moore

Preliminary Programming Specs from the VDM-2/Graphic Display 38
Lee Felsenstein

An Interactive Heath H-8 Disassembler 42
Ronald T. Borochoff

Interfacing My Home Computer to a Large Scale System 46
Glenn Story

Protect Your Space 53
L. H. Reid

Some Improvements on 8080 PILOT

BY TIM SCULLY
35267-136 SH
P.O. Box 1000
Steilacoom, WA 98388

When I first saw Dr.Starkweather's version 1.1 8080 PILOT (*DDJ* #s 14, 15), I was excited by the possibility of writing programs for personal educational applications with this simple language. One of the projects I had in mind was learning Spanish, and I experimented with some simple programs for drilling myself in vocabulary and grammar. I soon learned that it takes a lot of time and effort to write a useful CAI program – more effort than the results justified if the programs were to be only for my use. Being lazy, I decided to try to work out an easier way, and by adding a few extra instructions to version 1.1 PILOT, succeeded in writing simple and effective vocabulary and grammar drill programs.

One of the classical techniques for doing vocabulary drill is the use of flash cards with a word or phrase in one language on one side and the second language on the other. Flash cards are portable and easy to create, but they suffer from a few disadvantages. The most important one is that the user can't check the accuracy of an answer without seeing the correct answer.

In writing a PILOT program for this sort of drill, I wanted to have the computer pose the problem, accept an answer and report on its accuracy. If the answer is correct, positive feedback should be supplied and if the answer is incorrect, a chance to try again should be offered before the correct answer is revealed. If the problem involves the translation of a phrase, it is desirable for the computer to indicate which, if any, of the words in the phrase were translated correctly, to aid in a second effort at translation.

As lessons are learned, it is necessary to add new material to existing programs or to construct new programs covering new material. This isn't easy to do with version 1.1 PILOT, both because adding each new word or phrase requires quite a few instructions and because this version has no provision for easily adding to existing programs.

To solve these problems, a few new instructions were added: one allowing the value of string variables to be defined or assigned directly (instead of through an accept statement), a modification of the match statement which allows matching to a string variable, a new matching instruction for phrases which provides feedback on the correct portions of a phrase, and an instruction allowing easy addition of instructions to an existing PILOT program. These new instructions made it possible to write a subroutine package in PILOT, which made the main portion of a 'flash card' program very compact and easy to write.

As I was working on these modifications to PILOT, my computer grew in capability by the acquisition of two SA400 disk drives. This logically led to the addition of instructions for disk loading and saving of programs. My system uses several cards designed by Polymorphic Systems Inc., and shares a common property of Poly systems: low memory is dedicated to ROM and system software, leaving application programs in higher memory addresses. So I reassembled Dr. Starkweather's PILOT to a higher address space, beginning at 3200H, together with my additions to his software.

The bulk of Dr. Starkweather's program is unchanged, except for reassembly to a different address space. The modifications which have been made can be considered in two groups: those peculiar to the Poly disk system, and those which could be useful on any 8080 system. The table CTLST has been expanded and somewhat modified to include these two groups of changes.

The string variable detection and matching routines have been interfaced to version 1.1. PILOT by placing their addresses in the table locations corresponding to 'M' and 'MC' instructions, so that 'MOP' is replaced by 'NEWMOP' and 'MCOP' is replaced by 'NEWMC'.

The new instructions ADD:, P:, D:, L: and S: were added at the end of the CTLST table. A simple editor has been interfaced to the existing EDIT: command. The disk system commands are S: for Saving programs on disk and L: for Loading programs from disk; the existing LPRG: and SPRG: instructions were not modified.

The ADD: routine blanks memory from the top of the existing program to the end of the buffer. It jumps to the old INPT1 routine after printing the usual 'ENTER PILOT PROGRAM . . .' message. Additional program text may then be added in the usual way, terminated with CTRL Z.

M: and MC: The M: instruction has been modified to check for the presence of a string variable immediately to the right of the colon. If no '$' is found, the old MOP routine is entered. If '$' is found, a SEARCH routine is called which looks for the value of the string variable. If no value is found, an error is reported. If a value is found, it is passed to the old MOP as the string to be matched. Anything to the right of the string variable will be ignored by this modified MOP. The MC: routine has been modified in exactly the same manner.

D: In version 1.1 PILOT string variables are assigned values only through 'A:' or accept statements. To make it possible to write subroutines manipulating string variables, the 'D:' or 'define' operation has been created. A string variable should

appear immediately to the right of the colon. Separated by one space, it should be followed by the string which defines the variable. Thus $A can be defined as 'TEST STRING' by the following statement: D:$A TEST STRING.

P: The P: statement allows matching for a list of words, a phrase or a sentence. It first tests for the presence of a string variable to the right of the colon, just as the modified M: and MC: statements do, and substitutes the value of the variable if found. Then it checks for the presence of each word in the string, counting each found as a hit and each not found as a miss. If there are zero misses, the Y/N flag is set to YES. Each word that is found is printed, followed by the word 'FOUND', followed in turn by the number of words not found. This provides detailed feedback for partially accurate translation of phrases or grammar exercises. The P: statement will treat commas as break characters separating words, i.e. a phrase to be matched may contain commas just as in the MC: statement. Although it is possible to search for two or more acceptable strings (they should be separated by carets, as in the MC: statement), this isn't a very good idea, since the hit/miss counting of the P: statement will get very confused.

L: The L: command allows owners of Poly 8810 or 8813 systems to load PILOT programs from disk. This command has been used to expand the initialization dialogue of PILOT so that you are asked "Want to load a new program?", and if you answer "yes", then you'll be asked "From disk?". If that question is answered yes, an L: command is executed and the system prompts you by printing "file name:". You should respond with a legal Poly file name just as you would respond to EXEC. It is also possible to type CTRL Z followed by L: <cr> to force loading of a program from disk. Once the requested program has been loaded, PILOT will ask if you want to load a new program, and if you type "no", you'll begin execution of the newly loaded program.

S: The S: or save, command may be used by Poly 8810/8813 owners to save PILOT programs. The normal entry to the S: command will be via the immediate mode, typing CTRL Z, S:, <cr>. The system responds with the prompt "file name:" and waits for a legal Poly file name, just as you might type into EXEC. Once a legal file name has been typed, the system will save whatever program exists in PILOT's workspace. It will then loop back to ask if you want to load a new program. At this point, your program is still in memory and may be executed by answering "no".

Subroutines

I wrote a package of subroutines for use in writing programs for Spanish vocabulary drill. These appear at the beginning of each of the two PILOT programs included with this article: SPANISH - 1 and SPANISH - 2. Once the subroutine package has been debugged, I recorded it with the S: command as a separate disk file. To create a new SPANISH lesson, I L: the subroutine file and then execute the ADD: command in immediate mode. Now only the very simple main program has to be entered, with about three lines of program text for each word or phrase to be learned. Once a program has been completed, I use the S: command to record it (together with the subroutine package) for later use.

Let's follow a program through a few steps to see how the subroutine package works. Keep in mind that the dialogue is taking place on a 16 x 64 video display.

```
computer                student                comments
-----------------------------------------------------------------
$                                          (Poly EXEC prompt)
                2 PILOT                  (invoke PILOT from disk 2)
PILOT-8080 1.1
LOAD A NEW PROGRAM?
                Y
FROM DISK?
                Y
file name:
                2 SPANISH-2
PILOT-8080 1.1
LOAD A NEW PROGRAM?
                N
THIS IS ENGLISH/SPANISH VOCABULARY DRILL
DO YOU WANT ENGLISH FIRST?
                Y
TRANSLATE "RULER" INTO SPANISH, PLEASE.
                REGLA
RIGHT! "REGLA" IS "RULER".
TRANSLATE "LIGHT,ILLUMINATION" INTO SPANISH, PLEASE.
                LUZE
THAT"S NOT IT. WANT TO TRY AGAIN?
                Y
TRANSLATE "LIGHT,ILLUMINATION" INTO SPANISH, PLEASE.
                LUZ
RIGHT! "LUZ"IS "LIGHT".
TRANSLATE "CHALK" INTO SPANISH, PLEASE.
                TIZ
THAT'S NOT IT. WANT TO TRY AGAIN?
                N
THE TRANSLATION OF "CHALK" IS "TIZA"
HIT RETURN WHEN READY TO CONTINUE
                 CR
(the computer types 15 carriage returns to erase the TV)
TRANSLATE "CHALK" INTO SPANISH, PLEASE.
                TIZA
RIGHT! "TIZA" IS "CHALK".
TRANSLATE "TWELVE AND A HALF,12:30" INTO SPANISH, PLEASE.
                DOCE Y MITADA
THAT'S NOT IT. HERE'S HOW YOUR TRANSLATION COMPARES TO MINE...
DOCE Y FOUND 01 NOT FOUND
WANT TO TRY AGAIN?
                Y
TRANSLATE "TWELVE AND A HALF,12:30" INTO SPANISH, PLEASE.
                DOCE Y MEDIA
RIGHT! "DOCE Y MEDIA" IS "TWELVE AND A HALF,12:30".
........
```

The subroutine package, as you can see from the above sample, is designed to make it impossible to get past a lesson until it has been learned well (at least well enough for the student to remember it for a half minute or so, while the TV screen is blanked after the right answer has been displayed). The frustration level associated with this requirement is minimized by allowing the student to "give up" and ask for the correct answer as often as he or she likes, and that correct answer may be studied for as long as the student likes.

Adding new vocabulary to an existing program is very easy. If you have the program in memory already, and want to add one new vocabulary entry, here's how you do it.

```
computer                student                          comments
-----------------------------------------------------------------
                 CTRL Z ADD: CR    (invoke ADD: in immediate mode)
ENTER PILOT PROGRAM
TERMINATE INPUT WITH CTRL/Z
                D:$A LUEGO
                D:$B LATER
                U:*W
                CTRL Z                          (terminate input)
PILOT 8080-1.1
LOAD A NEW PROGRAM?
........
```

The subroutine package can of course be easily adapted to flash card drill with other languages or for memorization of other material. I hope that you find it as useful as I have.

.....Listing.....

```
;PILOT-36 12/23/78 TIM SCULLY
STKPR EQU 1000H
PSTRT EQU 34E0H
PBUFE EQU 7FFFH
MNTR EQU 403H ;EXEC -PolyMorphics disk
;operating system warm entry point
LINE EQU 80
WH1 EQU 0C24H ;prints character in A on TV
WH0 EQU 0C20H ;returns char from keyboard in A
 ORG 3200H
 IDNT 3200H,3200H
 JMP START
 JMP RSTRT
CI JMP CHI ;CHAR IN TO A
CO JMP CHO ;CHAR OUT FROM C
RI JMP CHI ;READER, INPUT FROM A
LO JMP CHO ;LIST OUTPUT FROM C
PO JMP CHO ;PUNCH OUT FROM C
EXIT JMP MNTR ;TO MONITOR
EDIT JMP EDITOR
ASCAN JMP BASIC ;ALTSCAN INTERPRETER
;DATA STORAGE AREAS
TOPP DW 0
HLSAV DW 0
HLLSAV DW 0
HL2SAV DW 0
DESAV DW 0
LLSAV DW 0
RETSAV DW 0
 DW 0
 DW 0
 DW 0
 DW 0
 DW 0
 DW 0
 DW 0
APTR DW 0
EPTR DW 0
CPTR DW 0
IPTR DW 0
MPTR DW 0
MEMTP DW 0
OUTADR DW 0
SCANB DW 0
CHMAX DS 1
LEVEL DS 1
LNSKP DS 1
MBRCH DS 1
SCNT DS 1
TEMP DS 1
VARSAV DS 1
YNSW DS 1
TSAVE DS 81
EBUFF DS 81
MSAVE DS 81
LABSAV DS 12
LASTOP DS 11
NVAR DS 53
WORD DS 81
 ORG PSTRT
START LXI SP,STKPR
 LXI H,PBUFE-1 ;INITIALIZE APTR
 SHLD APTR
 LXI H,IOJMP
 LXI D,CI
 MVI C,24
 CALL BLKTFR
 JMP RSTRT
IOJMP JMP CHI
 JMP CHO
 JMP CHI
 JMP CHO
 JMP CHO
 JMP MNTR
 JMP EDITOR
 JMP BASIC
RSTRT CALL BLANK ;erases all memory above current
;existing program, resets end flag
 LXI SP,STKPR
 CALL INIT
 CALL SCAN
 JMP RSTRT
 DB '020677',0DH
 DB 'PILOT-8080, 1.1',0DH
 DB 'COPYRIGHT <C> 1977,J.A.STARKWEATHER',0DH
 DB 'modified by Tim Scully 12/23/78',0DH
 DB 'COPYRIGHT <C> 1978, Tim Scully,
                          all rights reserved',0D
INIT LXI H,IBUFF
 SHLD IPTR
 SHLD SCANB
 LXI H,PBUFE
 SHLD MEMTP
 CALL NEWN
 CALL INITV
 MVI M,1
 MVI A,LINE
 STA CHMAX
 XRA A
 STA LEVEL
 STA LNSKP
 LXI H,CO
 SHLD OUTADR
 RET
SCAN LHLD IPTR
 MOV A,M
 CPI 1
 RZ
 CPI 0DH
 JNZ CKEND
 INX H
 JMP SCAN+3
CKEND CALL CNTLN
 CPI 1
 RZ
 INX H
 SHLD IPTR
 DCX H
 CALL BACKUP
 CALL SKLN
 CALL GETCH
 CPI ':'
 CZ CONTIN
 JZ SCAN
 CPI '*'
 CZ GETWD
 JZ SCAN+3
 CALL OPS
```

```
 JMP SCAN
CONTIN INX H
 SHLD HLSAV
 LXI H,LASTOP
 SHLD LLSAV
 MVI B,':'
 CALL INDX
 CALL OLDOP
 XRA A
 RET
OPS SHLD LLSAV
 CALL SAVOP
 LHLD LLSAV
 MVI B,':'
 CALL INDX
 MOV A,C
 ORA A
 JZ ALTSC
 INX H
 SHLD HLSAV
 DCX H
OLDOP DCX H
 CALL YNCHK
 ORA A
 RZ
 CALL VARCHK
 RZ
 LHLD LLSAV
 CALL GETCTL
 CALL CTLMCH
 CPI 1
 RNZ
ALTSC LHLD LLSAV
 CALL ASCAN
 RZ
 LHLD LLSAV
 SHLD HLSAV
 CALL TOP
 RET
TXTCK LXI H,PBUFF
 CALL CNTLN
 INX H
 MOV A,M
 CPI 0AH
 JNZ CKLC
 CALL CNTLN
 INX H
 MOV A,M
 CPI 0AH
 JNZ CKLC
 LDA LNSKP
 ADI 1
 STA LNSKP
CKLC LXI H,PBUFF
 CALL CNTLN
 INX H
 CALL SKLN
 MOV E,M
 MVI D,0
 DCX D
 DAD D
 MOV A,M
 CPI 0DH
 JNZ CKLN
 INX H
 CALL SKLN
 MOV E,M
 DCX D
 DAD D
 MOV A,M
 CPI 0DH
 JNZ CKLN
 LDA LNSKP
 ADI 1
 STA LNSKP
CKLN LXI H,PBUFF+1
 CALL CNTLN
 INX H
 CALL SKLN
 MOV A,M
 CALL NUM
 RNZ
 LDA LNSKP
 ADI 4
 STA LNSKP
 RET
NUM CPI '0'
 RM
 CPI '9'+1
 JM YNUM
 ORA H
 RET
YNUM XRA A
 RET
SKLN LDA LNSKP
 ORA A
 RZ
 INX H
 DCR A
 JMP SKLN+3
YNCHK CALL GETLCH
 CPI 'Y'
 JZ YCHK
 CPI 'N'
 JZ NCHK
 ORA A
 RET
YCHK LDA YNSW
 ORA A
 JZ DONT
 ORA H
 RET
NCHK LDA YNSW
 ORA A
 JNZ DONT
 ORA H
 RET
DONT XRA A
 RET
VARCHK CPI ')'
 JZ VCHK
 ORA H
 RET
VCHK DCX H
 DCX H
 MOV A,M
 CPI '('
 JNZ BADFRM
 INX H
 MOV B,M
 CALL VARMCH
 CPI 1
 JZ BADFRM
 INX H
 MOV A,M
 CPI 1
 JM VOFF
 ORA H
 RET
VOFF XRA A
 RET
BADFRM LHLD LLSAV
 CALL TOP+3
 LXI H,EXPMSG
 CALL ERROR
 RET
VARMCH LXI H,NVAR
 MOV A,M
 CPI 1
 RZ
 CMP B
 RZ
 INX H
 INX H
 JMP VARMCH+3
CTLMCH LXI D,CTLST
 CALL LSTMCH
 CPI 1
 RZ
 XCHG
 INX H
 LXI D,RTRN
 PUSH D
 MOV E,M
 INX H
 MOV D,M
 PUSH D
 RET
RTRN XRA A
 RET
LSTMCH LXI H,WORD
 CALL CMPR
 ORA A
 RNZ
 INX H
 XCHG
 INX H
 INX H
 INX H
 MOV A,M
 CPI 1
 RZ
 XCHG
 JMP LSTMCH
CTLST DB 'T',0DH
 DW TOP
 DB 'A',0DH
 DW AOP
 DB 'M',0DH
 DW NEWMOP ;new MOP routine checks
;for string variable
 DB 'MC',0DH
 DW NEWMC ; new MCOP also checks
;for string variables
 DB 'J',0DH
 DW JOP
 DB 'R',0DH
 DW ROP
 DB 'C',0DH
 DW COP
 DB 'U',0DH
 DW UOP
 DB 'E',0DH
 DW EOP
 DB 'Y',0DH
 DW TOP
 DB 'N',0DH
 DW TOP
 DB 'LOAD',0DH
 DW LOAD
 DB 'INMAX',0DH
 DW INMAX
 DB 'NEW$',0DH
 DW NEWN
 DB 'DP',0DH
 DW DPRG
 DB 'PRINT',0DH
 DW LPRG
 DB 'SAVE',0DH
 DW SPRG
 DB 'IEP',0DH
 DW IEP
 DB 'BYE',0DH
 DW EXIT
 DB 'EDIT',0DH
 DW EDIT
```

```
 DB 'ADD',0DH ;allows more program text to
 DW ADDPROG ;be added to existing program
 DB 'P',0DH ;compares phrases and lists
 DW PHRASE ;hits and misses
 DB 'D',0DH ;allows definition of string
 DW DEFINE ;variables directly
 DB 'L',0DH ;disk load routine
 DW DLOAD
 DB 'S',0DH ;disk save routine
 DW DSAVE
 DB 1 ;END OF TABLE
IEP LXI H,PBUFF
 SHLD IPTR
 SHLD SCANB
 CALL TXTCK
 RET
INMAX CALL NMCTL
 MOV A,E
 CPI 73
 JM INMX2
 MVI A,72
 MOV E,A
INMX2 LXI H,CHMAX
 MOV M,E
 RET
NMCTL LHLD HLSAV
 CALL GETCH
 CPI 0DH
 RZ
 CALL GETWD
 DCX H
 SHLD HLSAV
 LXI H,WORD
 CALL LETTER
 JNZ CVNUM
 MOV B,M
 CALL VARMCH
 CPI 1
 CZ BADFRM
 RZ
 INX H
 MOV E,M
 JMP CVNUM+3
CVNUM CALL GETNM
 MOV A,E
 ORA A
 RP
 MVI E,0
 RET
JOP LHLD HLSAV
 CALL GETCH
 CPI '*'
 JZ JOP2
 MVI A,'*'
 STA WORD
 LXI D,WORD+1
 LHLD HLSAV
 CALL WDTFR
 JMP JOP2+3
JOP2 CALL GETWD
 LHLD SCANB
 CALL LOOKL
 CPI 1
 CZ NTFND
 RZ
 INX H
 SHLD IPTR
 RET
NTFND LXI H,WORD
 CALL DSPLY
 LXI H,BLMSG
 CALL ERROR
 RET
UOP CALL SAVRET
 JMP JOP
BLKSET MOV M,B
 INX H
 MOV A,C
 SUI 1
 MOV C,A
 JNZ BLKSET
 RET
BLKBF LHLD APTR
 XCHG
 LXI H,PBUFF
 MVI B,' '
BLKB2 MOV M,B
 INX H
 CALL ADRCMP
 JNZ BLKB2
 RET
ADRCMP MOV A,H
 CMP D
 RM
 RNZ
 MOV A,L
 CMP E
 RET
GETNM MVI E,0
 INX H
 CALL BRCHAR
 JZ SDIG
 CPI '+'
 JZ SDIG
 CPI '-'
 JZ SDIG
 DCX H
 MOV A,M
 CPI '0'
 RM
 CPI '9'+1
 RP
 SUI '0'
 ADD A
 MOV E,A
 ADD A
 ADD A
 ADD E
 MOV E,A
 INX H
UNITS MOV A,M
 CPI '0'
 RM
 CPI '9'+1
 RP
 SUI '0'
 ADD E
 MOV E,A
 RET
SDIG DCX H
 JMP UNITS
PUTNM MVI C,0
 MOV A,E
 ORA A
 CM NEG
 CPI 10
 JM FRMCH
 SUI 10
 MOV E,A
 MOV A,C
 ADI 1
 MOV C,A
 JMP PUTNM+2
FRMCH MOV A,C
 ADI '0'
 CPI '0'
 JZ FRMU
 MOV M,A
 INX H
FRMU MOV A,E
 ADI '0'
 MOV M,A
 INX H
 MVI A,0DH
 MOV M,A
 RET
NEG MVI A,'-'
 MOV M,A
 INX H
 XRA A
 SUB E
 MOV E,A
 RET
LOOKS XRA A
 STA TEMP
 JMP LOOK
LOOKL ORA H
 STA TEMP
 CALL SKLN
LOOK CALL GETCH
 CPI 1
 RZ
 CPI '*'
 JZ CHK
 CPI '$'
 JZ CHK
 CALL CNTLN
 INX H
 LDA TEMP
 ORA A
 JZ LOOK
 JMP LOOK-3
CHK SHLD HLSAV
 CALL CNTWD
 MOV A,C
 CPI 13
 JM MVLAB
 MVI C,12
MVLAB LHLD HLSAV
 LXI D,LABSAV
 CALL BLKTFR
 XCHG
 DCX H
 MVI A,0DH
 MOV M,A
 LXI H,LABSAV
 LXI D,WORD
 CALL CMPR
 ORA A
 JNZ LFND
 LHLD HLSAV
 CALL CNTWD
 JMP LOOK
LFND LHLD HLSAV
 CALL CNTWD
 RET
CMPR MOV A,M
 CPI 0DH
 JZ XEND
 MOV C,A
 INX H
 XCHG
 MOV A,M
 CPI 0DH
 JZ YENDB
 CMP C
 JNZ NOMCH
 INX H
 XCHG
 JMP CMPR
XEND XCHG
 MOV A,M
 CPI 0DH
 JZ MCH
 CALL CNTWD
 XCHG
 XRA A
 RET
NOMCH CALL CNTWD
YENDB XCHG
 CALL CNTWD
 XRA A
 RET
MCH XCHG
 ORA H
 RET
GETCH MOV A,M
 CPI 20H
 RNZ
 INX H
 JMP GETCH
GETLCH MOV A,M
 CPI 20H
 RNZ
 DCX H
 JMP GETLCH
GETWD CALL GETCH
 LXI D,WORD
 CALL WDTFR
 RET
CNTWD MVI C,1
 CALL BRCHAR
 RZ
 MOV A,C
 ADI 1
 MOV C,A
 INX H
 JMP CNTWD+2
WDTFR MVI C,1
 CALL BRCHAR
 JZ MVBR
 INX H
 XCHG
```

```
 MOV M,B
 MOV A,C
 ADI 1
 MOV C,A
 INX H
 XCHG
 JMP WDTFR+2
MVBR INX H
 XCHG
 MVI A,0DH
 MOV M,A
 INX H
 XCHG
 XRA A
 RET
GETCTL LXI D,WORD
 CALL WDTFR
 MOV A,C
 CPI 3
 RM
 MOV H,D
 MOV L,E
 DCX H
 DCX H
 CALL GETLCH
 CPI 'Y'
 JZ YNOUT
 CPI 'N'
 RNZ
YNOUT MVI A,0DH
 MOV M,A
 RET
SAVOP MVI B,':'
 CALL INDX
 LHLD LLSAV
 LXI D,LASTOP
 CALL BLKTFR
 RET
CNTLN MVI C,1
 MOV A,M
 CPI 0DH
 RZ
 CPI 1
 RZ
 MOV A,C
 ADI 1
 MOV C,A
 INX H
 JMP CNTLN+2
BACKUP MOV A,C
 CPI 1
 RZ
 SUI 1
 MOV C,A
 DCX H
 JMP BACKUP
BRCHAR MOV A,M
 MOV B,A
 CPI ' '
 RZ
 CPI 0DH
 RZ
 CPI ','
 RZ
 CPI ';'
 RZ
 CPI ':'
 RZ
 CPI '.'
 RZ
 CPI '?'
 RZ
 CPI 21H
 RZ
 CPI '"'
 RZ
 CPI '('
 RZ
 CPI ')'
 RZ
 CPI 27H
 RZ
 CPI 1
 RET
INDEX XRA A
 STA SCNT
INDE2 LHLD HLSAV
 XCHG
 LHLD HLLSAV
 MOV B,M
 INX H
 XCHG
 CALL INDX
 MOV A,C
 ORA A
 RZ
 SHLD HLSAV
 LDA SCNT
 ADD C
 STA SCNT
 LHLD HLLSAV
 CALL CNTLN
 MOV A,C
 SUI 1
 MOV C,A
 LXI D,WORD
 LHLD HLSAV
 CALL BLKTFR
 SHLD EPTR
 XCHG
 MVI A,0DH
 MOV M,A
 LHLD HLLSAV
 XCHG
 LXI H,WORD
 CALL CMPR
 ORA A
 JNZ SETCNT
 LHLD HLSAV
 INX H
 SHLD HLSAV
 JMP INDE2
SETCNT LXI H,SCNT
 MOV C,M
 RET
INDX MVI C,1
 MOV A,M
 CMP B
 RZ
 CPI 0DH
 JZ ZC
 MOV A,C
 ADI 1
 MOV C,A
 INX H
 JMP INDX+2
ZC XRA A
 MOV C,A
 RET
SINDX MVI C,1
 MOV A,M
 CPI '$'
 RZ
 CPI 23H
 RZ
 CPI 0DH
 RZ
 MOV A,C
 ADI 1
 MOV C,A
 INX H
 JMP SINDX+2
LETTER MOV A,M
 MOV B,M
 CPI 41H
 JM NOTL
 CPI 5AH
 JP NOTL
 XRA A
 RET
NOTL ORA H
 RET
SETUP CALL CNTLN
 MOV B,C
 CALL BACKUP
 MOV C,B
 RET
TOP LHLD HLSAV
 SHLD CPTR
 SHLD LLSAV
 LXI H,TSAVE
 SHLD DESAV
 XCHG
TMORE LHLD CPTR
 CALL SINDX
 CPI '$'
 JZ GETXT
 CPI 23H
 JZ GETNUM
 JMP TMOVE
GETXT CALL INSERT
 CPI 1
 JNZ TMORE
 LXI D,TSAVE
 LHLD LLSAV
 JMP TALL
GETNUM CALL INSNUM
 JMP GETXT+3
TMOVE LHLD DESAV
 XCHG
 LHLD CPTR
TALL CALL SETUP
 CALL BLKTFR
 MVI A,0DH
 XCHG
 MOV M,A
 LXI H,TSAVE
 CALL DSPLY
 XRA A
 RET
INSNUM MOV A,C
 SUI 1
 JZ VBL
 MOV C,A
 LHLD DESAV
 XCHG
 LHLD CPTR
 CALL BLKTFR
VBL MOV B,H
 MOV C,L
 XCHG
 SHLD DESAV
 MOV H,B
 MOV L,C
 INX H
 MOV C,M
 INX H
 SHLD CPTR
 MOV B,C
 CALL VARMCH
 CPI 1
 RZ
 INX H
 MOV E,M
 LXI H,WORD
 CALL PUTNM
 LHLD DESAV
 XCHG
 LXI H,WORD
 CALL SETUP
 MOV A,C
 SUI 1
 MOV C,A
 CALL BLKTFR
 XCHG
 SHLD DESAV
 XRA A
 RET
INSERT MOV A,C
 SUI 1
 JZ LBL
 MOV C,A
 LHLD DESAV
 XCHG
 LHLD CPTR
 CALL BLKTFR
LBL MOV B,H
 MOV C,L
 XCHG
 SHLD DESAV
 MOV H,B
 MOV L,C
 CALL GETWD
 DCX H
 SHLD CPTR
 LHLD APTR
 INX H
 CALL LOOKS
 CPI 1
 RZ
 CALL GETA
 RET
GETA INX H
 MOV B,H
 MOV C,L
 LHLD DESAV
 XCHG
 MOV H,B
 MOV L,C
 CALL SETUP
 MOV A,C
 SUI 1
```

```
 MOV C,A
 CALL BLKTFR
 XCHG
 SHLD DESAV
 XRA A
 RET
AOP LHLD HLSAV
 CALL GETCH
 CPI '$'
 JNZ NIN
 LXI D,LABSAV
 CALL WDTFR
 CALL ENTR
 CALL ASTORE
 RET
NIN CPI 23H
 JNZ CENT
 INX H
 MOV A,M
 STA VARSAV
 CALL ENTR
 CALL CKNUM
 CALL NSTORE
 RET
CENT CALL ENTR
 RET
ENTR CALL KEYIN
 LXI H,EBUFF
 MOV A,M
 CPI 1AH
 RNZ
 INX H
 MOV A,M
 CPI 0DH
 JZ RSTRT
 CALL OPS
 JMP ENTR
CKNUM CPI '0'
 JM NERR
 CPI '9'+1
 RM
NERR LXI H,NMSG
 CALL ERROR
 CALL ENTR
 JMP CKNUM
ASTORE LXI H,EBUFF
 CALL CNTLN
 MOV B,C
 LHLD APTR
 CALL DECA
 MOV A,C
 ORA A
 RZ
 LXI H,EBUFF
 CALL BLKTFR
 LXI H,LABSAV
 CALL CNTLN
 MOV B,C
 LHLD APTR
 CALL DECA
 MOV A,C
 ORA A
 RZ
 LXI H,LABSAV
 CALL BLKTFR
 MOV L,E
 MOV H,D
 DCX H
 MVI A,' '
 MOV M,A
 RET
DECA MOV E,L
 MOV D,H
 DCX H
 MOV A,M
 CPI 1
 JZ STOVF
 MOV A,B
 SUI 1
 MOV B,A
 JNZ DECA
 SHLD APTR
 MVI M,1
 RET
STOVF LXI H,NRMSG
 CALL ERROR
 MVI C,0
 RET
NSTORE LXI H,EBUFF
 CALL GETCH
 CALL GETNM
 LXI H,VARSAV
 MOV B,M
 CALL VARMCH
 CPI 1
 CZ BADFRM
 RZ
 INX H
 MOV M,E
 RET
MC LXI H,MBRCH
 MVI M,'^'
 JMP MOP1
MOP LXI H,MBRCH
 MVI M,','
MOP1 LHLD HLSAV
 SHLD MPTR
NEXTM CALL MMOV
 LHLD MPTR
 SHLD HLSAV
 CALL SQUEZ
 LXI H,EBUFF
 SHLD HLSAV
 CALL PAD
 CALL SQUEZ
 LXI H,EBUFF
 SHLD HLSAV
 LXI H,MSAVE
 SHLD HLLSAV
 CALL INDEX
 MOV A,C
 ORA A
 CNZ SWY
 RNZ
 LHLD MPTR
 DCX H
 MOV A,M
 CPI 0DH
 JZ MDONE
 INX H
 MOV A,M
 CPI 0DH
 JZ MDONE
 JMP NEXTM
MDONE CALL SWN
 RET
SWY ORA H
 STA YNSW
 RET
SWN XRA A
 STA YNSW
 RET
MMOV LHLD MPTR
 LXI D,MSAVE
MMOV2 CALL MBR
 JZ SMOV
 INX H
 XCHG
 MOV M,B
 INX H
 XCHG
 JMP MMOV2
SMOV INX H
 XCHG
 MVI A,0DH
 MOV M,A
 INX H
 XCHG
 SHLD MPTR
 RET
MBR MOV A,M
 MOV B,A
 CPI 0DH
 RZ
 LDA MBRCH
 CMP B
 RET
PAD LXI H,WORD
 MVI A,' '
 MOV M,A
 INX H
 XCHG
 LHLD HLSAV
 CALL CNTLN
 LHLD HLSAV
 CALL BLKTFR
 XCHG
 DCX H
 MVI A,' '
 MOV M,A
 INX H
 MVI A,0DH
 MOV M,A
 LHLD HLSAV
 XCHG
 LXI H,WORD
 CALL CNTLN
 LXI H,WORD
 CALL BLKTFR
 RET
SQUEZ LHLD HLSAV
 SHLD HLLSAV
 MOV A,M
CKEOL CPI 0DH
 RZ
 CPI ' '
 JZ CKNC
 INX H
 JMP SQUEZ+3
CKNC INX H
CKNC1 MOV A,M
 CPI ' '
 JNZ CKEOL
 SHLD HLLSAV
 CALL SHIFT
 LHLD HLLSAV
 JMP CKNC1
SHIFT INX H
 MOV B,M
 DCX H
 MOV M,B
 MOV A,M
 CPI 0DH
 RZ
 INX H
 JMP SHIFT
COP LHLD HLSAV
 MVI B,'='
 CALL INDX
 MOV A,C
 ORA A
 JZ EXMSG
 INX H
 CALL GETCH
 CPI '-'
 JNZ CGVAL
 SHLD HLLSAV
 XRA A
 STA TEMP
 LHLD HLLSAV
 JMP SUBV
CGVAL CALL GVALUE
 LXI H,TEMP
 MOV M,E
 LHLD CPTR
 INX H
 CALL GETCH
 CPI 0DH
 JZ AVAL
 CPI '-'
 JZ SUBV
 CPI '+'
 JZ ADDV
 CALL GVALUE
 JMP AVAL
ADDV INX H
 CALL GVALUE
 LDA TEMP
 ADD E
 CPI 100
 JP OVMSG
 MOV E,A
 JMP AVAL
SUBV INX H
 CALL GVALUE
 LDA TEMP
 SUB E
 CPI 9DH
 JP AVAL-1
 CPI 100
 JP UNMSG
 MOV E,A
AVAL CALL ASSIGN
 RET
EXMSG CALL TOP
 LXI H,EXPMSG
 CALL ERROR
 RET
OVMSG MVI E,99
 CALL ASSIGN
 CALL TOP
 LXI H,OVFMSG
```

```
 CALL ERROR
 RET
UNMSG MVI E,-99
 CALL ASSIGN
 CALL TOP
 LXI H,UNFMSG
 CALL ERROR
 RET
GVALUE CALL GETCH
 MOV E,A
 CALL LETTER
 JZ LTR
 CALL GETNM
 SHLD CPTR
 RET
LTR SHLD CPTR
CONV MOV B,M
 CALL VARMCH
 CPI 1
 CZ BADFRM
 RZ
 INX H
 MOV E,M
 RET
ASSIGN LHLD HLSAV
 CALL GETCH
 CALL LETTER
 JNZ EXMSG
 MOV B,M
 CALL VARMCH
 CPI 1
 CZ BADFRM
 RZ
 INX H
 MOV M,E
 RET
BASIC ORA H
 RET
ROP RET
EOP LDA LEVEL
 ORA A
 JZ RSTRT
 CALL RESRET
 RET
LOAD LXI H,PBUFF
 CALL INPUT
 RET
NEWN LHLD APTR
 MVI M,20H
 LHLD MEMTP
 MVI M,1
 DCX H
 SHLD APTR
 RET
INITV LXI H,NVAR
 MVI B,'A'
 MOV A,B
NV CPI 'Z'+1
 RZ
 MOV M,A
 INX H
 MVI A,0
 MOV M,A
 INX H
 MOV A,B
 ADI 1
 MOV B,A
 JMP NV
SAVRET LXI H,LEVEL
 MOV A,M
 ADI 1
 CPI 8
 JM SAV2
 LXI H,STMSG
 CALL ERROR
 RET
SAV2 MOV M,A
 ADD A
 MOV C,A
 LHLD IPTR
 XCHG
 LXI H,RETSAV
 MOV A,L
 ADD C
 MOV L,A
 MOV M,D
 ADI 1
 MOV L,A
 MOV M,E
 RET
RESRET LDA LEVEL
 ADD A
 MOV C,A
 LXI H,RETSAV
 MOV A,L
 ADD C
 MOV L,A
 MOV D,M
 ADI 1
 MOV L,A
 MOV E,M
 XCHG
 SHLD IPTR
 LXI H,LEVEL
 MOV A,M
 SUI 1
 MOV M,A
 RET
BLKTFR MOV A,C
 ORA A
 RZ
 MOV B,M
 INX H
 XCHG
 MOV M,B
 INX H
 XCHG
 MOV A,C
 SUI 1
 MOV C,A
 JNZ BLKTFR
 RET
KEYIN LXI H,EBUFF
 SHLD EPTR
 LXI H,CHMAX
 MOV B,M
KIN2 LHLD EPTR
 CALL CI
 CPI 5FH ;FOR DELETE
 JZ CANC
 CPI 7FH ;FOR RUBOUT
 JZ CANC
 CPI 18H ;CTRL X
 JZ CANL
 CPI 61H
 JM NTR
 XRI 20H
NTR MOV M,A
 INX H
 SHLD EPTR
 CPI 0DH
 JZ KOUT
 DCR B
 MOV A,B
 ORA A
 JNZ KIN2
 MVI C,0DH
 MOV M,C
 CALL CO
KOUT CALL LF
 RET
CANL MVI C,3CH
 CALL CO
 CALL CRLF
 JMP KEYIN
CANC MOV A,B
 LXI H,CHMAX
 MOV C,M
 CMP C
 JZ KIN2
 INR B
 LHLD EPTR
 DCX H
 SHLD EPTR
 JMP KIN2+3
CRLF MVI C,0DH
 CALL CO
LF MVI C,0AH
 CALL CO
 RET
INPUT CALL BLKBF
 LXI H,PBUFF
INPT1 MOV A,M
 SHLD LLSAV
 CPI 1
 JZ CHOP
 CALL RI
 CPI ' '
 JNZ INPT3+3
 JMP INPT1
INPT2 MOV A,M
 CPI 1
 JZ CHOP
INPT3 CALL RI
 CPI 0
 JZ INPT3
 CPI 7FH
 JZ CLC
 CPI 1AH ;CTRL Z TERMINATES INPUT
 JZ INEND
 CPI 18H ;CTRL X TO KILL LINE
 JZ KLN
 CPI 5FH
 JZ CLC ;CANCEL LAST CHAR
 MOV M,A
 INX H
 CPI 0DH
 JNZ INPT2
 CALL LF
 JMP INPT1
INPT4 MOV M,A
 INX H
 JMP INPT2
CLC DCX H
 JMP INPT3
KLN MVI C,3CH
 CALL CO
 CALL CRLF
 LHLD LLSAV
 JMP INPT3
CHOP LHLD LLSAV
 CALL DSPLY
 LXI H,IOVMSG
 CALL ERROR
INEND MVI M,1
 SHLD TOPP
 CALL CRLF
 LXI H,LEVEL
 MVI M,0
DSPLY MOV A,M
 INX H
 MOV D,H
 MOV E,L
 CPI 1
 RZ
 MOV C,A
 LHLD OUTADR
 CALL OVCTR
 CPI 0DH
 JZ ENDOL
 XCHG
 JMP DSPLY
ENDOL MVI C,0AH
 LHLD OUTADR
 CALL OVCTR
 XCHG
 CALL SKLN
 RET
OVCTR PCHL
PRGOUT LXI H,PBUFF
 CALL DSPLY
 CPI 1
 JNZ PRGOUT+3
 RET
DPRG LXI H,CO
 SHLD OUTADR
 CALL PRGOUT
 RET
```

```
LPRG LXI H,PBUFF ;PRINT PROGRAM
 MOV A,M
 INX H
 CPI 1
 RZ
 CALL WH7 ;printer output for poly system
 JMP LPRG+3
SPRG LXI H,PO
 SHLD OUTADR
 CALL PRGOUT
 LXI H,CO
 SHLD OUTADR
 RET
 MOV C,A
;CHO is modified because Poly uses interrupt
;driven input and a ring buffer
CHO PUSH H
 LHLD 2D84H
 LDA 2D82H ;poly ring buffer addresses
 CMP L
 POP H
 JZ PRINT
 CALL WHO
 CPI 1AH ;CTRL Z
 JNZ PRINT
 CALL INTR
PRINT MOV A,C
 CALL WH1
 RET
INTR PUSH H
 PUSH D
 PUSH B
 LXI H,INTMSG
 CALL ERROR
 CALL ENTR
 POP B
 POP D
 POP H
 RET
CHI CALL WHO
 CALL WH1
 RET
ERROR CALL DSPLY
 XRA A
 RET
;add: allows additional program text
;to be added to an existing program
;*for example, additonal vocabulary
;drill
BLANK LXI H,PBUFF-1 ;erases all storage
 INX H ;above existing program
 MOV A,M
 CPI 1
 JNZ BLANK+3 ;look for end of old program
 SHLD TOPP ;and save it
 INX H
 XCHG
 LXI H,PBUFE
 XCHG
 CALL BLKBF+7 ;blank the free buffer space
 MVI M,1 ;set end flag
 RET
ADDPROG CALL BLANK
 LXI H,RSTRT
 PUSH H ;set up return address
 LXI H,ENTMSG1+5
 CALL DSPLY
 LXI H,ENTMSG2+1
 CALL DSPLY
 LHLD TOPP
 MVI M,20H ;elminate previous end flag
 JMP INPT1 ;input new program text
CHECK LHLD HLSAV ;look for '$'
 MVI A,'$'
 CMP M,
 RET
;search finds the text corresponding
;to a label found at (hl) or else
;reports an error by returning with
;the z flag not set, exits with hlsav
;pointing to desired text, if found
SEARCH CALL GETWD ;get label and stash in 'word'
 DCX H
 SHLD CPTR
 LHLD APTR
 INX H
 CALL LOOKS
 SHLD HLSAV ;save address
 CPI 1 ;set error flag
 RET
NEWMOP CALL CHECK ;is there a $?
 JNZ MOP ;normal match
 CALL SEARCH ;get text
 JNZ MOP ;then match
ERRORS LXI H,BLMSG ;complain if label
 CALL ERROR ;not found
 RET
NEWMC CALL CHECK ;check for $
 JNZ MC ; normal match
 CALL SEARCH ;get text
 JNZ MC ;then match
 JMP ERRORS ;if not found
PHRASE CALL CHECK ;look for $
 JNZ POPS ;normal phrase match
 CALL SEARCH ;find text
 JNZ POPS
 JMP ERRORS
; define allows labels to be defined
;directly, without going thru an
;accept statement.
DEFINE LHLD HLSAV
 CALL GETCH
 CPI '$' ;dop must have a label
 JNZ ERRORS ;immediately after the :
 CALL CNTLN ;how long is it?
 MOV B,C
 LHLD APTR
 CALL DECA ;find storgae address
 MOV A,C
 ORA A
 RZ ;return if no room for storage
 LHLD HLSAV
 CALL BLKTFR ;store the label & definition
 RET
;new storgae locations required
;by modified program
LASTONE DS 1 ;save last break char
NBUF DS 5 ;buffer for numerical conversion
HITS DS 1 ;count hits
MISSES DS 1 ;count misses
BUF DS 44 ;buffer for Gfid
NDA DS 2 ;next free disk address
NSEC DS 2 ;number of sectors
LADD DW PBUFF ;load address
SADD DW 3203H ;warm start address
FADD DW PBUFF ;from address
```

```
NDA1 DS 2
Overto EQU 0412H ;fetch overlay routine
NFDIR EQU 2DA0H ;drive # of directory in memory
Msg EQU 040CH ;message printer
Look EQU 0421H ;looks up files
SUBF1 EQU 2800H ;directory storage buffer
Runr EQU 0424H ;fetch & run disk file
Dio EQU 0406H ;disk i/o routine
WH7 EQU 0C3CH ;poly printer wormhole
MOVE EQU 0100H
DEOUT EQU 03D1H ;prints DE
;pmbr looks for words in a phrase
;by finding breaks between them
;which are spaces, commas or carets
PMBR XRA A
 STA LASTONE
 MOV A,M
 MOV B,A
 CPI 0DH
 RZ
 STA LASTONE
 CPI '^'
 RZ
 CPI ','
 RZ
 CPI ' '
 RNZ
 RET
;pmmov isolates one word of
;a phrase and moves it to be
;searched for
PMMOV LXI D,MSAVE
 CALL PMBR
 JZ SMOV
 INX H
 XCHG
 MOV M,B
 INX H
 XCHG
 JMP PMMOV+3
;if done checking a phrase,
;then it is time to report
;on hits and misses
FINISH LDA MISSES
 CPI 0
 CZ SWY ;set YES switch if no misses
 LDA HITS
 CPI 1
 JZ NOHITS
 LXI H,BLMSG+11
 CALL DISP
NOHITS MVI A,' '
 CALL CHO-1
 LDA MISSES
 LXI H,NBUF
 MOV E,A
 CALL PUTNM
 LXI H,NBUF
 CALL DISP
 LXI H,BLMSG+6
 CALL DSPLY ;PRINT 'NOT FOUND'<CR>
 POP H
 RET
;counts one hit
;and checks for end
OK LDA HITS
 INR A
 STA HITS
 LXI H,MSAVE
 CALL DISP
 LDA LASTONE
 CPI 0
 JZ FINISH
 RET
;counts one miss and then
;checks for end
BADS LDA MISSES
 INR A
 STA MISSES
 JMP OK+13
SPAD MVI A,20H
 STAX D
 INX D
 JMP SMOV
;pops matches the text from
;an accept statement to program
;text, printing the matching
;words and the number of missing
;words
POPS XRA A ;clear counters
 STA HITS
 STA MISSES
 LHLD HLSAV
NEXTMP CALL PMMOV ;get a word
 LHLD MPTR
 SHLD HLSAV
 CALL SQUEZ
 LXI H,EBUFF
 SHLD HLSAV
 CALL PAD
 CALL SQUEZ
 LXI H,EBUFF
 SHLD HLSAV
 LXI H,MSAVE
 SHLD HLLSAV
 CALL INDEX ;check for match
 MOV A,C
 ORA A
 CNZ OK ;count hits
 CZ BADS ;or misses
 LHLD MPTR ;pick up where we left off
 JMP NEXTMP ;keep checking
;disp prints a message starting at (hl)
;and ending with <cr> or 1, it does not
;print <cr>
DISP MVI A,' '
 CALL CHO-1
 MOV A,M
 CPI 0DH
 RZ
 CPI 1
 RZ
 CALL CHO-1
 INX H
 JMP DISP+5
;poly system 88 disk routines
OOPS XRA A ;disk error handler
 STA NFDIR
 CALL Overto
 DB 'Emsg' ;fetch error overlay
 MVI A,0DH
 CALL CHO-1
 JMP RSTRT ;back to warm start after error
DISKIO CALL Dio
 RNC
 JMP OOPS
```

```
DSAVE LXI H,PBUFF
 SHLD FADD
 DCX H
 MVI A,1
 INX H
 CMP M
 JNZ DSAVE+9 ;find end of program
 LXI D,-PBUFF
 DAD D
 MOV A,H
 INR A
 STA NSEC ;# sectors to save
 LXI H,PROMPT
 CALL Msg
 LXI H,1
 LXI B,474FH ;default extension = .GO
 LXI D,BUF
 MVI A,0A0H
 CALL Overto
 DB 'Gfid'
 JC OOPS
 LXI H,BUF+1
 MOV A,M
 ANI 1FH
 ORI 20H
 MOV M,A
 DCX H ;get drive #
 MOV A,M
 ANI 3
 INX H
 CALL Look
 JC NOTEXT
 LXI D,0600H ; error code
 JMP OOPS
NOTEXT MOV A,E
 ORA A
 JNZ OOPS
 MOV A,D
 CPI 3
 JNZ OOPS
 LHLD SUBF1+0DH
 SHLD NDA
 SHLD NDA1
 LXI H,BUF+1
 MOV A,M ;fetch flag byte
 ANI 1FH
 ADI 3
 MOV E,A
 MVI D,0
 DAD D
 LXI D,NDA
 MVI B,8
LP1 LDAX D
 MOV M,A
 INX H
 INX D
 DCR B
 JNZ LP1
 LHLD FADD
 XCHG
 LHLD NDA1
 MVI B,0 ;write flag
 LDA BUF ;get drive #
 ANI 3
 MOV C,A
 LDA NSEC ;# sectors to dump
 CALL DISKIO
 LXI H,BUF
 MVI A,1
 CALL Overto ;make directory entry
 DB 'Gfid'
 JC OOPS
 JMP RSTRT
DLOAD LXI H,PROMPT
 CALL Msg
 LXI H,1
 LXI D,BUF
 MVI A,0A0H
 CALL Overto
 DB 'Gfid'
 JC OOPS
 LDA BUF+1
 ANI 0FH ;mask name length
 STA BUF+1
 LXI H,BUF+1
 LDA BUF
 ANI 83H ;extension needn't match
 CALL Runr
 JC OOPS
 PCHL
 JMP RSTRT
;this editor was documented
;in DDJ 3(2):43, 1977
EDITOR LXI H,PBUFF
 MVI A,0CH
 CALL WH1
 SHLD FADD ;save starting address
LP2 LHLD FADD
 XCHG
 CALL DEOUT
 XCHG
 MVI A,' '
 CALL WH1
LP3 MOV A,M
 CALL WH1
 INX H
 CPI 0DH
 JNZ LP3
LP2A CALL WH0
 CPI ' '
 JNZ LP4
 LHLD FADD
 INX H
 JMP LP2-3
LP4 CPI 7FH
 JNZ LP5
 LHLD FADD
 PUSH H
 POP D
 LXI B,0
 MVI A,1
 DCX H
LP4A INX H
 DCX B
 CMP M
 JNZ LP4A
 LHLD FADD
 INX H
 CALL MOVE
 JMP LP2
LP5 CPI 0DH
 JZ LP2-3
 CPI 0AH
 JNZ LP6
 LHLD FADD
 DCX H
```

```
LP5A DCX H
 MOV A,M
 CPI 0DH
 JNZ LP5A
 INX H
 JMP LP2-3
LP6 CPI 1AH ;CTRL Z
 JZ RSTRT ;BACK TO PILOT
 CPI 4 ;CTRL D inserts a space
 JNZ LP7
 MVI A,' '
 JMP INSERTS
LP7 CPI 5 ;CTRL E inserts <cr>
 JNZ LP8
 MVI A,0DH
 JMP INSERTS
LP8 CPI 1FH
 JC LP2A
INSERTS LHLD FADD
 PUSH PSW
 MVI A,1
 LXI B,0
 DCX H
LP8A INX H ;find end of program
 DCX B ;counting down in bc
 CMP M
 JNZ LP8A
 MOV D,H
 MOV E,L
 INX D
LP8B MOV A,M ;move block
 STAX D
 DCX D
 DCX H
 INR C
 JNZ LP8B
 INR B
 JNZ LP8B
 POP PSW ;INSERT
 STAX D
 INX H
 INX H
 JMP LP2-3
PROMPT DB 'file name:',0
 ;error messages
BLMSG DB '-LABEL NOT FOUND',0DH
IOVMSG DB '/OVERFLOW',0DH
NRMSG DB '*NO ROOM',0DH
EXPMSG DB '*ILLEGAL EXPRESSION',0DH
OVFMSG DB '*VALUE > 99',0DH
UNFMSG DB '*VALUE < -99',0DH
STMSG DB '*USE DEPTH EXCEEDED',0DH
NMSG DB '*NUMERIC RESPONSE REQUIRED',0DH
INTMSG DB '*INTERRUPTED',0DH
IBUFF:DB 'T:',0DH
 DB 'T:PILOT-8080, 1.1',0DH
 DB ':LOAD A NEW PROGRAM?',0DH
 DB 'A:',0DH
 DB 'M: Y',0DH
 DB 'JN:*%',0DH
 DB 'T:FROM DISK?',0DH
 DB 'A:',0DH
 DB 'M:Y',0DH
 DB 'JN:*&',0DH
 DB 'L:',0DH
ENTMSG1 DB '*& T:ENTER PILOT PROGRAM',0DH
ENTMSG2 DB ':TERMINATE INPUT WITH CTRL/Z',0DH
 DB 'LOAD:',0DH
 DB '*% IEP:',0DH
 ORG 44FFH
 DB 0DH
PBUFF DB 1
 END
```

drills: 1

```
U:*E1A
T:THIS IS ENGLISH/SPANISH VOCABULARY DRILL.
:DO YOU WANT ENGLISH FIRST?
A:
M:Y,O,U
CY:A=1
CN:A=-1
J:*START
R:SUBROUTINES FOLLOW
R:*G IS FOR GRAMMER QUESTIONS
R:*P IS FOR MATCHING PHRASES AND *W FOR WORDS
*P J(A):*PE
T:TRANSLATE " $A " INTO ENGLISH, PLEASE.
A:
MC:$B
JY:*E
T:THAT'S NOT IT. HERE'S HOW YOUR TRANSLATION COMPARES TO MINE...
P:$B
T:WANT TO TRY AGAIN?
A:
M:Y,O,U
JY:*P
U:*E2
J:*P
*E2 T:THE TRANSLATION OF " $A " IS " $B ".
*CLEAR T:$C
:HIT RETURN WHEN READY TO CONTINUE.
A:
T:
:
:
:
:
:
:
:
:
:
:
:
:
E:
*E T:RIGHT! " $A " IS " $B ".
*E1 T:$C
*E1A NEW$:
D:$C ..................................................
E:
*PE T:TRANSLATE " $B " INTO SPANISH, PLEASE.
A:
MC:$A
JY:*E
T:THAT'S NOT IT. HERE'S HOW YOUR TRANSLATION COMPARES TO MI
P:$A
T:WANT TO TRY AGAIN?
A:
M:Y,O,U
JY:*PE
U:*E3
U:*CLEAR
J:*PE
*E3 T:THE TRANSLATION OF " $B " IS " $A ".
E:
*W J(A):*WE
T:TRANSLATE " $A " INTO ENGLISH, PLEASE.
A:
M:$B
JY:*E
T:THAT'S NOT IT. WANT TO TRY AGAIN?
A:
M:Y,O,U
JY:*W
U:*E2
J:*W
*WE T:TRANSLATE " $B " INTO SPANISH, PLEASE.
```

```
A:
M:$A
JY:*E
T:THAT'S NOT IT. WANT TO TRY AGAIN?
A:
M:Y,O,U
JY:*W
U:*E3
U:*CLEAR
J:*W
*G T:$B
A:
M:$A
JY:*E4
P:$A
JY:*E4
T:WANT TO TRY AGAIN?
A:
M:Y,O,U
JY:*G
T:THE ANSWER IS $A
U:CLEAR
J:*G
*E4 T:RIGHT! THE ANSWER IS $A
J:*E1
*START D:$A SENOR
D:$B MR,SIR,MAN,GENTLEMAN
U:*W
D:$A SEIS
D:$B SIX,6
U:*W
D:$A PROFESORA
D:$B TEACHER,PROFESSOR,(FEMALE)
U:*W
D:$A NUMERO
D:$B NUMBER
D:$C NU'MERO
U:*W
D:$A CERO
D:$B ZERO,O
U:*W
D:$A SENORA
D:$B MRS,MADAM,LADY,WIFE
U:*W
D:$A NUEVE
D:$B NINE,9
U:*W
D:$A BUENOS DIAS
D:$B GOOD MORNING
D:$C BWENOZDIAS
U:*P
D:$A CINCO
D:$B FIVE,5
U:*W
D:$A PROFESOR
D:$B TEACHER,PROFESSOR,(MALE)
U:*W
D:$A UNO
D:$B ONE,1
U:*W
D:$A BUENAS NOCHES
D:$B GOOD NIGHT
D:$C (EVENING)
U:*P
D:$A SENORITA
D:$B MISS,YOUNG LADY
D:$C SENYORITA
U:*W
D:$A SIETE
D:$B SEVEN,7
U:*W
D:$A ADIOS
D:$B GOODBY,GOOD-BY
D:$C ADIO'S
U:*W
D:$A QUE TAL?
D:$B HOW GOES IT?
U:*P
D:$A DOS
D:$B TWO,2
U:*W
D:$A BIEN
D:$B WELL
D:$C (BYEN)
U:*W
D:$A ESTOY
D:$B I AM
U:*W
D:$A OCHO
D:$B EIGHT,8
U:*W
```

```
D:$A BUENAS TARDES
D:$B GOOD AFTERNOON
U:*P
D:$A HOLA
D:$B HELLO
U:*W
D:$A TRES
D:$B THREE,3
U:*W
D:$A HASTA MANANA
D:$B UNTIL TOMORROW
D:$C MAN~ANA
U:*P
D:$A OLA
D:$B HI
U:*W
D:$A CUATRO
D:$B FOUR,4
U:*W
D:$A CON PERMISO
D:$B EXCUSE ME
D:$C KONPERMISO
U:*P
D:$A DIEZ
D:$B TEN
U:*W
D:$A ME LLAMO
D:$B MY NAME IS
D:$C I CALL MYSELF
U:*P
D:$A COMO SE LLAMA USTED?
D:$B WHAT IS YOUR NAME?
U:*P
D:$A DIALOGO UNO: THOMAS Y LUISA
D:$B DIALOG ONE: TOM AND LOUISE
U:*P
D:$A HOLA, LUISA! QUE TAL?
D:$B HI, LOUISE! HOW GOES IT?
U:*P
D:$A BASTANTE BIEN. Y TU, COMO ESTAS?
D:$B PRETTY WELL. AND HOW ARE YOU?
U:*P
D:$A ESTOY MUY BIEN, GRACIAS.
D:$B I'M VERY WELL, THANKS
U:*P
D:$A ESTOY BIEN GRACIAS, Y USTED?
D:$B I'M FINE THANKS, AND YOU?
U:*P
T:THAT'S ALL FOR LESSON ONE. THANKS FOR YOUR ATTENTION.
```

2

```
U:*E1A
T:THIS IS ENGLISH/SPANISH VOCABULARY DRILL.
:DO YOU WANT ENGLISH FIRST?
A:
M:Y,O,U
CY:A=1
CN:A=-1
J:*START
R:SUBROUTINES FOLLOW
R:*G IS FOR GRAMMER QUESTIONS
R:*P IS FOR MATCHING PHRASES AND *W FOR WORDS
*P J(A):*PE
T:TRANSLATE " $A " INTO ENGLISH, PLEASE.
A:
MC:$B
JY:*E
T:THAT'S NOT IT. HERE'S HOW YOUR TRANSLATION COMPARES TO MINE...
P:$B
T:WANT TO TRY AGAIN?
A:
M:Y,O,U
JY:*P
U:*E2
J:*P
*E2 T:THE TRANSLATION OF " $A " IS " $B ".
*CLEAR T:$C
:HIT RETURN WHEN READY TO CONTINUE.
:
:
:
:
:
:
:
:
E:
*E T:RIGHT! " $A " IS " $B ".
```

```
*E1 T:$C
*E1A NEW$:
D:$C ....................................................
E:
*PE T:TRANSLATE " $B " INTO SPANISH, PLEASE.
A:
MC:$A
JY:*E
T:THAT'S NOT IT. HERE'S HOW YOUR TRANSLATION COMPARES TO MINE...
P:$A
T:WANT TO TRY AGAIN?
A:
M:Y,O,U
JY:*PE
U:*E3
U:*CLEAR
J:*PE
*E3 T:THE TRANSLATION OF " $B " IS " $A ".
E:
*W J(A):*WE
T:TRANSLATE " $A " INTO ENGLISH, PLEASE.
A:
M:$B
JY:*E
T:THAT'S NOT IT. WANT TO TRY AGAIN?
A:
M:Y,O,U
JY:*W
U:*E2
J:*W
*WE T:TRANSLATE " $B " INTO SPANISH, PLEASE.
A:
M:$A
JY:*E
T:THAT'S NOT IT. WANT TO TRY AGAIN?
A:
M:Y,O,U
JY:*WE
U:*E3
U:*CLEAR
J:*W
*G T:$B
A:
M:$A
JY:*E4
P:$A
JY:*E4
T:WANT TO TRY AGAIN?
A:
M:Y,O,U
JY:*G
T:THE ANSWER IS $A
U:*CLEAR
J:*G
*E4 T:RIGHT! THE ANSWER IS $A
J:*E1
*START D:$A REGLA
D:$B RULER
U:*W
D:$A LUZ
D:$B LIGHT,ILLUMINATION
U:*W
D:$A TIZA
D:$B CHALK
U:*W
D:$A DOCE Y MEDIA
D:$B TWELVE AND A HALF,12:30
U:*P
D:$A ESPERANDO
D:$B WAITING
U:*W
D:$A OJO!
D:$B WATCH OUT!
U:*W
D:$A QUIEN
D:$B WHO
D:$C QUIE'N
U:*W
D:$A VOY HA TRANCAR LA PUERTA
D:$B I'M GOING TO LOCK THE DOOR
U:*P
D:$A PIZARRA
D:$B BLACKBOARD
U:*W
D:$A ALLI
D:$B THERE
U:*W
D:$A VOY HA ABRIR EL CANDADO
D:$B I'M GOING TO OPEN THE LOCK
U:*P
D:$A MITADAS
D:$B HALF
D:$C AS IN HALF AN APPLE
U:*W
D:$A LUEGO
D:$B LATER
U:*W
D:$A ELLAS
D:$B THEY,(FEMALE)
U:*W
D:$A LA MITADA
D:$B ONE HALF
U:*W
D:$A JABON
D:$B SOAP
D:$C HAVON
U:*W
D:$A PAJARA
D:$B BIRD
U:*W
D:$A CUADERNO
D:$B NOTEBOOK
U:*W
D:$A SILLA
D:$B CHAIR
U:*W
D:$A VENTANA
D:$B WINDOW
U:*W
D:$A MIRA LA PAJARA
D:$B LOOK AT THE BIRD
U:*P
D:$A VERDAD
D:$B TRUE, REALLY
U:*W
D:$A ALMUERZO
D:$B LUNCH
U:*W
D:$A DESAYUNO
D:$B BREAKFAST
U:*W
D:$A ESTOY ALMORZANDO
D:$B I'M HAVING LUNCH
U:*P
D:$A ESTOY CENANDO
D:$B I'M HAVING DINNER
U:*P
D:$A AYUNAS
D:$B FASTING
U:*W
D:$A CENA
D:$B DINNER
U:*W
D:$A -A -CION -SION -TAD -DAD -TUD -UMBRE -IE
D:$B LIST EIGHT FEMININE NOUN ENDINGS
U:*G
D:$A COMO MANOCISTES
D:$B HOW WAS YOUR AWAKENING
U:*P
D:$A ESTOY DESAYUNANDO
D:$B I'M HAVING BREAKFAST
U:*P
D:$A VENGA DE AYUNAS
D:$B COME WITHOUT EATING
U:*P
D:$A CLIMA DILEMA DRAMA ENIGMA ESQUEMA FANTASMA
D:$B LIST SIX MASCULINE NOUNS ENDING IN -A AND STARTING WITH
                                        C, D, E, OR F.
U:*G
D:$A PANORAMA TELEGRAMA TEMA TRAUMA PROBLEMA PROGRAMA
D:$B LIST SIX MASCUNINE NOUNS ENDING IN -A AND
                                    STARTING WITH LETTERS AFTER F.
U:*G
D:$A COMO MANOCIO
D:$B HOW WAS THE AWAKENING
D:$C COULD APPLY TO OTHERS
U:*P
D:$A NO, ESTA CERCA
D:$B NO, IT'S NEAR
U:*P
D:$A REPITA
D:$B REPEAT, SAY IT AGAIN
D:$C RREPI(!)TA
U:*W
D:$A POR FAVOR
D:$B PLEASE
U:*W
D:$A PERDON, COMO DICE USTED?
D:$B EXCUSE ME, WHAT DID YOU SAY?
D:$C PERDO(!)N KO(!)MODI(!)SEUSTE(!)D
U:*P
T:THAT'S ALL FOR LESSON TWO. THANKS!
```

A District Attorney Responds to Jon Taber

BY DONALD J. INGRAHAM
Senior Trial Deputy
Alameda County District Attorney's Office
1225 Fallon Street
Oakland, CA 94612

We published an article in DDJ *# 37. "On Computer Crime Bill S240," in which Jon Taber seriously criticized the bill now pending in Congress. The article brought an immediate response from Donald Ingraham, a rebuttal in theory and perspective, which we now publish for your further edification on the matter.* *–SR*

In *On Computer Crime Bill S240* (*DDJ* # 37), Jon Taber shrilled the clarion to resist the enactment of the Federal Computer Crime Bill, and before everyone abandons the keyboard for the barricades, I thought-as a prosecutor-it might be helpful to hear from the other side.

In one sense we are allies, Taber and I, because as an employee of a local D.A., living in and accountable to our community, I do not think a Federal statute is needed. The enactment of S240 would permit defense counsel to argue preemption, the legal doctrine which precludes States from enforcing copyright, postal fraud and some other laws, on the theory that the availability of the Feds excludes States and local relief. Particularly in computer-related crime, my experience is that Federal preemption would deny protection to the smaller victims, and that anything smaller than Rifkin would not be accepted for prosecution because of U.S. Attorney budget restraints. Besides, we local firms like to help on local problems.

Which brings Mr. Taber and I to a parting of the ways, because he does not believe computer-related crime requires new laws to control, and I do. Not because computers are inherently evil, or programmers are given to sin more than the rest of us, but because a new technology carries the spores of an invasion of personal rights which it is my job, as a prosecutor, to resist.

At the risk of appearing to interject a Bicentennial Minute, to understand this problem it is is necessary to reflect on what the customer, the citizen, is asking from the Justice system: freedom, and the security without which exercise of that freedom is at least discouraged, and too often rendered impossible. Part of the expectation of the community is that their employees will follow the rules, and neither unnecessarily invade privacy nor misuse the tax-provided tools entrusted to them.

Do computers jeopardize that freedom and security? I believe they do, just as would an unsafe highway, unprotected phone lines and unregulated food and drugs. The automobile had a somewhat analogous impact upon our communities, and the manufacturers and then largely upper-crust operators probably expressed similar dismay over the concept of legislators (who probably didn't even *have* cars) presuming to control their playthings. Lord Peter Whimsey was a notoriously reckless driver, and after all, demmit, one will pay for the demnation cow, what? But that attitude has become a relic, even though there were laws which addressed damage to cows then on the books, indeed going back to Hammurabi's tablets. The laws are not passed to feed ravenous and ambitious prosecutors, but to protect the public.

And so it is with computers. Like it or not, a whole lot of everyone's property and privacy is now in some computer or another, and to the extent that privacy and property are legitimate recipients of the State's protection, that protection should be extended to the computers. It may be that part of Mr. Taber's different perception may be the result of our different training, and that some translator is needed. Property, in law, is an extension of the person, and consists not of the constituent atoms and elements of the land, literary endeavor, or bank account, but of the right of a particular individual thereto, somewhat metaphysical notion, but one which underlies the society which programmers and prosecutors, popcorn distributors and p. r. people inhabit. When in the pre-computer era access to personal information was a matter of a curiously configured metal "key" working on coded tumblers beyond which lay marked paper, the property now resides in a matrix of impulses, perhaps vulnerable to a password. That this change in format in no way changes the legal nature of the interests is clear to every lawyer I have approached, but it seems to cloud the minds of some computer specialists to an extent heretofore reached only by the late Lamont Cranston. And this is unfortunate because it reduces the dialogue I called for in *Computerworld* a few years back to a monologue with heckling, and I make a poor substitute for Fozzi Bear.

It would be entertaining to me to do a point by point response to Mr. Taber's brief. Perhaps we could hire a hall. I do think it important, as a member of the Bar given an opportunity to reach you of the Byte, to ask for a reconsideration of what Mr. Taber calls "Common Practices". This involves the recreational (or high school political) use of computers, their time and printouts, by employees. This is wrong, unless it is considered by the employer to be a fringe benefit, and is apparently so widespread as to be regarded as the droit de seigneur, of elitist memory. I question whether your profession can much longer avoid, with a wink and a nudge, addressing the ethics of the droit de programmeur.

There have been crimes accomplished with computers, usually in an inactive role. They have not been easy to prosecute, and we have been proceeding with spit, tape, and simile to discharge our responsibilities. Prosecutors do not hate computers; in fact, we use them as we did in prosecuting Watergate. We assuredly are not lusting after higher office with your charges as stepping stones to glory. Computers are not that attractive, and we have murderers vying for our attention.

Although I hope, too, that S240 is aborted or at least the interstate commerce extension deleted (I don't see how the Feds can be denied protection of their own computers), some legal recognition of the computer as an instrumentality for crime, and an Achille's heel for the people, is inevitable. Mr. Taber has testified and written effectively and often persuasively on the shortcomings and overreachings of embryonic legislation, but an affirmative proposal might be even more successful. There is a time to debug, and a time to design. The customer wants a protection program, and wants it *now*.

Jon Taber Responds to Donald Ingraham

BY JON TABER
609 Meadow Avenue
Santa Clara, CA 95051

When we received Mr. Ingraham's article, we sent a copy to Jon Taber for his comments. They follow. —SR

I read Mr. Ingraham's response to my article with interest, especially since he is chairman of the California District Attorneys Association Computer Committee. I was disappointed that he chose not to reply point by point. A response such as this would be helpful. I am not trying to be unreasonable in opposing computer crime legislation; if I am in error, Mr. Ingraham is cordially invited to demonstrate it for everyone's benefit, including my own.

Mr. Ingraham does agree with me that Sen. Ribicoff's bill, S240, should be defeated, or at least narrowed, although his reasons are different from mine. Mr. Ingraham disagrees with me on other points, so I wanted to stress our area of agreement: S240 is an ill-conceived bill, and should be defeated. A prosecutor's evaluation may have more weight than a programmer's, so I want to thank Mr. Ingraham for his contribution.

I do believe that a state computer crime bill, such as California's Senate Bill S66, (or any of the state bills now in process in about 20 state legislatures) is unnecessary. There is no crime that can conceivably be committed with a computer that is not pretty directly covered by one or more statutes of the California Penal Code. I would be astonished if the Codes of the other states were any less comprehensive.

I hope we can all agree that crimes such as fraud and embezzlement are already covered by law, and that the alleged use of a computer in commission of these crimes is immaterial. Other crimes are covered by the following California laws:

1. Theft of trade secrets, Calif Penal Code 499c. This statute is directly applicable to theft of programs and data. There is no requirement to demonstrate *asportation* (taking of the stolen item in the legal sense. Theft, unlike trade secret theft, requires asportation – and merely "copying" something without "taking" it doesn't qualify.)
2. Theft of services (CPC 532). This statute directly covers theft of machine usage, for example, using dp resources without the permission of the owner.
3. False record entry or record ateration, CPC471.1. This statute directly covers cases of entering false records or altering records, whether or not maintained in a computer system.
4. Malicious mischief, CPC 594. This statute is directly applicable to cases of destruction of tampering with computer data or programs, or the operation of the computer system. And malice, by the way, is determined by the facts of the case.
5. Criminal trespass, CPC 602j. This statute covers interference with a computer system. The offense is a misdemeanor.
6. Forgery, CPC 470. Applicability of this statute may not be immediately apparent, but it is applicable, and may prove a powerful sanction.. This statute covers the case of using an account or password not one's own to make use of a computer system. The account and password are considered legally to be "signatures" thus, using a false one is forgery.
7. Burglary, CPC 459. This statute is indirectly applicable. The commission of a felony on someone else's premises (for example, a computer installation) triggers burglary. Any felony is the sufficient *mens rea*. The fact that a programmer is privileged to use the computer by the owner is no defense in California (it is in some other states, however). Thus, the "authorized" progammer who steals a program may be charged with burglary as well as theft of trade secrets.

In addition, other statutes in the CPC may apply depending on the facts of the case and the legal requiremnts to support the charges. These are theft, CPC 484a, if the requirement of asportation can be satisfied; credit card abuse CPC 484 d-i; telephone abuse, CPC 474.

It seems to me these statutes provide complete protection for any thinkable "computer crime." We simply don't need one more law covering the same ground.

Mr. Ingraham asserts that prosecutors have had difficulties prosecuting "computer crime" cases. At the state level, if a prosecutor has had difficulties, it was due to improperly charging the culprit with something like theft of computer time instead of the proper charge, theft of services. Speaking rigorously, time is a dimension by which we measure the value of computer services; so it is services which is the *res* stolen, not time. Some judges therefore will not allow a charge of theft of time, as a prosecutor may learn if he bungles his indictment (but some judges do allow it). At the Federal level, prosecutor "difficulties" are due to over-reaching, that is, prosecuting offenses where the Federal authorities have no jurisdiction. The reader should know that murder and theft of services violate state law, not Federal law, unless the crime

Continued on pg. 25

North Star Disk

STOIC AN INTRODUCTION

BY B.W. LEE
1385 Rifle Range Rd.
El Cerrito, CA 954530

The aim of this series of four articles is to introduce STOIC as a programming language and as a program development system. This and the next in the series will cover the basic structures, syntax, and operations of the language and provide examples of their usage. The third in the series will describe the features of a specific implementation of STOIC for 8080/Z80 microcomputers based on the North Star Disk as the mass storage medium, the operations of the STOIC display editor, and the STOIC file system. The fourth and closing article will cover the modular extension packages for double precision integers, floating points, and complex FFT, with examples to illustrate their usage and applications.

WHAT IS STOIC?

The STOIC is an acronym derived from: *ST*ack *O*riented *I*nteractive *C*ompiler. Depending on the context in which it is used, STOIC refers either to a program development system or to a programming language.

As a program development system, STOIC is actually much more than a compiler. It comprises within a single consistent architecture the resources of a compiler, editor, assembler, interpreter, monitor, debugger, loader, and a complete and self-contained disk operating system. This richness in resources is supported with remarkable memory economy (the basic system including the disk operating system is less than 14K) and high program execution speed.

As a programming language, STOIC is nearly identical to FORTH in structure and syntax, but where FORTH is lean and skimpy, STOIC is baroque in its explicit support of a large number of data types with lavish sets of intrinsic functions and predefined operations. In addition to basic integers and Booleans, STOIC also supports string literals, arrays, double precision integers, floating points, and complex numbers, with corresponding sets of comprehensive operators and functions. Through the use of string literals STOIC, in contrast to FORTH, adheres strictly to the reverse polish convention throughout, even inthe syntax of word definitions. The use of string literals also enables STOIC to support a fully developed disk operating system with named contiguous files, in contrast to the fragmented files based on "SCREENS" which FORTH supports.

Note: Stoic, Part II will appear in DDJ # *42.*

WHY STOIC?

In the midst of the current hyperbolic efforts of many to promote PASCAL as the heir to high level programming languages and even to the discipline of structured programming, one may question the justification for interest in another language such as STOIC. In response one could compare those elements that are similar and those that are different or one could compare the relative advantages and disadvantages with respect to a set of applications. However, one soon realizes that while such comparisons may be interesting or instructive, they can never be conclusive.

The most appropriate response to the question "Why STOIC?" may be "Why not?". For as with natural languages, each viable language is at once similar and unique. No one language could ever be "best"for all occasions or applications. The co-existence of languages serves the irrepressible purposes of offering choice and richness in the expressions of ideas.

As will be clear later, STOIC is a general purpose programming language which is powerful and yet simple in syntax. It is completely in harmony with the precepts of structured programming. It shares with FORTH the unique property of spanning the levels of programming from the low assembly level to the heights of high level symbolic programming and adaptive self-development. With respect to the latter attribute, STOIC not only supports, as in PASCAL, the user definition of new data types, but it also permits great flexibility in extensions of the language through new word definitions. This capability permits the user to develop a powerful working vocabulary tailored specifically to his or her applications.

BASIC STOIC STRUCTURES

Being a user-alterable language, STOIC in the general form is open ended. For the purpose of this introduction it is therefore necessary to limit our attention first to the basic core of the language, the point from which either standard or user-defined modular extensions begin. This basic core is referred to as BASIC STOIC.

The most prominent feature of STOIC is its principal data structure called the DICTIONARY. Nearly all of the language except for the short compact codes of the interpreter is a part of the DICTIONARY. This structure is an ordered list of entries called WORDS. Associated with each WORD entry is a NAME. A legal NAME is any string of up to 127 ASCII characters. Punctuations, numerics, and most other special characters are permitted within a NAME. Two legal NAMES are equivalent if, and only if, their first five characters are the same and they have *exactly* the same length. Illegal characters within a NAME are:

SPACE, RETURN, FORM FEED, LINE FEED,
RUBOUT, NULL.

Examples:

PLOT	is not equivalent to	PLOTA
(HEAD1)	is equivalent to	(HEAD2)

☆

☆

A literal is a sequence of characters which denote a constant. BASIC STOIC supports two types of literals: 16-bit integers and strings. (Extensions of the language support other data types such as double precision integers, floating points, etc.)

An integer literal is a sequence of digits in the current radix, optionally preceded by a plus or minus sign. The radix can be DECIMAL, HEX, or OCTAL. The range of the integer literal must be within –32768 to 32767 if signed or 0 to 65535 if unsigned.

Examples: –1234 is a legal integer literal; +–1234 is not; –AFBC is a legal integer literal only if the current radix is HEX.

Representation of a string literal may be in any one of three forms:

1. a string between two double quotes: "STRING"
2. a string between two backslashes: \STRING\
3. a string preceded by a single quote and terminated by a SPAC or TAB: 'STRING

For all three forms, the end of line character (RETURN or FORM FEED) may serve as the alternate terminator. Special characters such as RETURN, RUBOUT, etc., may be included in a string literal by enclosing the corresponding ASCII code in octal between two &'s.

Examples: "THIS IS A STRING LITERAL"
"THIS LITERAL CONTAINS A RETURN &15&"
'&177&&15&

STOIC SYNTAX AND PROGRAMMING

STOIC syntax is very simple. A legal program line is a sequence of literals and/or NAMES of WORDS separated by SPACES and terminated by a RETURN or FORM FEED.

Programming in STOIC consists primarily of defining a set of new WORDS already defined in the DICTIONARY. A WORD must be defined before it can be used in a program line. In Basic STOIC an initial set of over 100 commonly useful WORDS called the KERNEL enables the user to get started.

The principal medium of communication between WORDS is the PARAMETER STACK, often referred to simply as "the stack". Data on which a WORD operates are pushed onto the stack. The WORD upon execution pops the data from the stack and pushes the results of the operations onto the stack. Although less typical, communication with VARIABLES in fixed memory locations are also used.

STOIC syntax follows strictly the reverse polish convention. This means that in all operations the operands precede their operators. This may seem strange at first to those having some experience with algebraic languages like BASIC, but it should be familiar to some pocket calculator users. The advantage of the convention is in not ever having to use parentheses to designate the correct order of operations.

Examples: 1 2 + 3 * in algebraic notation is (1+2)*3
1 2 3 * – is 1–(1*3)

BASIC STOIC OPERATORS

Operators, like +, –, * are WORDS which operate on operands on the stack and perform specific elementary transformations. Since all WORDS operate on data deposited onto the stack and transform them in some manner, the distinction of operators from other WORDS is somewhat arbitrary. The key distinctions are "shortness" and "elementary".

Even when reduced to its basic core, STOIC still provides the user with unusually large sets of pre-defined fixed point operators. These are generally grouped by the number of operands but in some cases also by the type of transformation.

Examples of operators requiring no operand are: DECIMAL, OCTAL, HEX; these change the current radix to that specified by the operator. Unary and binary operators are quite numerous. Only some of the most commonly useful ones are listed in Tables I and II. Note that the convention for Booleans are –1 (FFF in hex) for TRUE and 0 for FALSE. FALSE.

Operators which reorganize the contents of the stack are also quite numerous; again only some of the mot useful ones are listed in Table III along with their effects on the stack. It is important to be aware that each element of the stack is a 16-bit number.

A group of WORDS which are useful for I/O operations with the user's console device is listed in Table IV. The mode of interchange is in the curent radix or in ASCII characters. Conversion to another radix will be described in a later part of this series.

Unlike other high level languages, STOIC permits operations on address as well as their contents. This feature enables STOIC to span the levels of programming. However, it is very important for the user to be clearly aware of the distinction between addresses and their contents at all times.

There are three types of WORDS which push data onto the stack: literals, constants, and variables. A reference to a literal or to a constant causes its corresponding value to be pushed onto the stack. A reference to a variable, however, causes its address in fixed memory to be pushed.

Constants and variables must be declared before they can be referenced. The syntax for declaring a constant is:

VALUE 'NAME CONSTANT

where VALUE is the integer value of the constant and NAME is the name to be assigned to the constant. When compiled, a new DICTIONARY entry NAME is created which when referenced will, upon execution, push the VALUE onto the stack. The syntax for declaring a variable is:

VALUE 'NAME VARIABLE

where VALUE is the number on the top of the stack and NAME is the name to be assigned to the variable. VALUE is used to set the initial value of the variable. Examples:

10 'NPTS CONSTANT This sets up a DICTIONARY entry NPTS. Referencing and executing, the word NPTS will cause a 10 to be pushed onto the stack.

10 'COLUMN VARIABLE This defines a variable COLUMN in memory with an initial value of 10; when referenced and executed, the address of the memory location assigned to the variable will be pushed onto the stack.

Memory reference operators like those listed in Table V are predefined WORDS for accessing and modifying the values (i.e., contents) of variables in fixed memory locations. The following are some examples of their usage:

Examples: (X, Y, Z are variables; A, B, C are constants)

100 X !	sets the value of X to 100.
X 100 !	store the address of X at location 100.
X 100 ↤	same as 100 X !
X @ Y @ + Z !	sets the value of Z to the sum of value of X and value of Y.
X A B + + @ Y !	sets value of Y to the content of location X+A+B.
X 0 ↤	zeros the content of X.
A X +!	adds A to the content of X.
X 1+!	increments the content of X.

Figure 1. Examples of STOIC Operations

```
0> is the STOIC nesting depth count and user prompt

0> 1 MINUS =
-1

0> -1 ABS =
1

0> -1 EQZ =
0

0> 0 EQZ =
-1

0> 1 1 + =
2

0> 1 1 1 + + =
3

0> 1 2 3 4 5 = = = = =
5 4 3 2 1

0> 1 2 SWAP = =
1 2

0> 1 2 DUP = = =
2 2 1

0> 1 2 1 - EQ =
-1

0> 100 X ! X ?
100

0> -1 5 MAX =
5

0> -1 5 MIN =
-1
```

Table I. Unary Opertors

```
Number on the top of the stack is called A

MINUS   -A
ABS     absolute value of A
NOT     logical complement of A
2*      A*2
2/      A/2  (signed)
U2/     A/2  (unsigned)
1+      A+1
1-      A-1
EQZ     -1 if A equal to 0, 0 otherwise
```

NEZ	-1 if A not equal to 0, 0 otherwise
LTZ	-1 if A<0, 0 otherwise
LEZ	-1 if A<=0, 0 otherwise
GEZ	-1 if A>=0, 0 otherwise
GTZ	-1 IF A>0, 0 otherwise

Table II. Binary Operators

Number on top of the stack is A, next to top is B

+	B+A
-	B-A
*	B*A
/	B/A
MAX	maximun (B,A) (signed)
MIN	minimum (B,A) (signed)
AND	logical and of B,A
OR	logical or of B,A
XOR	logical exclusive or of B,A
EQ	-1 if B=A, 0 otherwise
NE	-1 if B not equal to A, 0 otherwise
LT	-1 if B<A, 0 otherwise
LE	-1 if B<=A, 0 otherwise
GE	-1 if B>=A, 0 otherwise
GT	-1 if B>A, 0 otherwise

Table III. Stack Operators

Name	Stack Before	Stack After	Description
DUP	A	A A	duplicates top of stack
OVER	A B	B A B	duplicates top-1 of stack
2OVER	A B C	C A B C	duplicates top-2 of stack
3OVER	A B C D	D A B C D	duplicates top-3 of stack
UNDER	A B	A	store top at top-1 of stack
2UNDER	A B C	B A	store top at top-2 of stack
3UNDER	A B C D	B C A	store top at top-3 of stack
DROP	A B	B	discards top of stack
2DROP	A B C	C	discards top and top-1 of stack
3DROP	A B C D	D	discards top 3 stack entries
SWAP	A B	B A	exchanges top 2 stack entries
2SWAP	A B C	A C B	exchanges top-1 and top-2 of stack
FLIP	A B C	C B A	exchanges top and top-2 of stack
+ROT	A B C	B C A	roll top 3 stack entries up
-ROT	A B C	C A B	roll top 3 stack entries down
DDUP	A B	A B A B	duplicates top 2 stack entries

Table IV. I/O Words

Top of the stack is A, top-1 of stack is B

!	store B at adress A.
<-	store A at address B.
@	replace top with content at address A
B!	store rightmost byte of B at address A
B@	load top with byte at address A
0<-	store 0 at address A.
-1<-	store -1 (FFFF in hex) at address A.
+!	add B to content of address A.
1+!	increment content of address A.
1-!	decrement content of address A.
MOVE	copy content of address B to address A.
XCHG	exchange contents of addresses B and A.
MVBYTE	copies bytes sequentially from one address range to another; top of stack is byte count, top-1 is destination starting address, top-2 is source starting address.

Table V. Memory Reference Words

Number on top of the stack is denoted by N.

COLUMN	variable containing the current column count.
TYO	output the ASCII character in the rightmost byte of N; column is incremented unless character is RETURN, in which case it is zeroed.
CR	output a RETURN follow by LINE FEED; column is zeroed.
FRESH	CR if column is not zero.
SPACE	outputs a space character.
SPACES	output N spaces.
TAB	tab to column N; nothing if already at or beyond N.
TYI	input a character.
=	output N in current radix follow by space.
?	output content of address N follow by space.
TYPE	output N ASCII characters starting at byte pointer at top-1 of stack.

COLON DEFINITION

Most of the flexibility and power of STOIC is derived from the ease with which the user can adapt and extend the vocabulary to suit his or her purpose. STOIC permits the user to define a new WORD in terms of STOIC WORDS already defined in the DICTIONARY, in terms of new assembly codes, or in terms of a segregated combination of existing STOIC WORDS and new assembly codes. Of these options, the first one, called COLON DEFINITIONS, is the easiest and will be described here. The other options will be reserved for a later part of this series.

The syntax of a COLON DEFINITION is as follows:

'NEWWORD : WORD1 WORD2 . . . WORDn ;

This will create a new DICTIONARY entry called NEWWORD which, when referenced and upon execution, will in turn execute WORD1, WORD2, . . . , WORDn precisely in that sequence. After executing the last word WORDn, control will return for the next word of the program. The string NEWWORD must be a legal STOIC name, and the words WORD1, WORD2, . . . , WORDn must already exist as entries in the DICTIONARY. If the latter is not true, STOIC will give a fatal error message of the following form:

UNDEFINED
(NAME OF UNDEFINED WORD)

An existing word may be redefined at any time. In this case, all prior definitions which referenced that word will still execute in accordance with the old definition, i.e., no change. All subsequent WORD definitions which reference that word will execute in accordance with that word's most recent definition.

If the name of the word being redefined appears within the new definition, its old meaning will be effective in compiling the new definition for the word. This convention permits the possibility for recursive calls.

REDEFINING (NAME OF WORD)
(Redefinition of an existing word causes STOIC to generate this warning message)
Examples: (current radix is octal; A, B, C are constants)

'AVERAGE : + 2/ ; This defines a new DICTIONARY entry called AVERAGE which on execution will compute the average of the two top numbers on the stack and replace them with the result.
'SPACE : 40 TYO This new entry will output a space character to the user console device. (40 is the ASCII code for the space character).
'QUADRATIC : DUP A * B + * C + ; This new entry will compute the value of the second order polynomial A* (X**2) + B*X + C with the number on the top of the stack as the value of X. The top of the stack is replaced with the result.

ITERATIONS AND LOOPS

STOIC supports five different structures for interactive executions of a sequence of words. These are:

N (W1 W2 . . . Wn), which execute the sequence of words included between the parentheses N times.

BEGIN W1 W2 . . . Wn END The sequence of words between BEGIN and END is executed in order once; when END is reached the top of the stack is popped and tested. If it is TRUE (i.e. non-zero) control passes to the word following END; if it is FALSE (i.e. zero) control passes to the word following BEGIN (i.e., repeat execution of sequence).

BEGIN T1 T2 . . . Tm IF W1 W2 . . . Wn REPEAT This is similar to BEGIN . . . END except that the test is made at the beginning rather than the end. The sequence of words between BEGIN and IF is executed, followed by popping and testing the top of the stack. If tested TRUE, the sequence of words between IF and REPEAT is executed and control passes back to the following BEGIN; if tested FALSE, control passes to the word following REPEAT.

HIGH LOW DO W1 W2 . . . Wn LOOP The limits HIGH and LOW (the top two stack entries) are compared; if HIGH is less than or equal to LOW, control passes to the word following LOOP. Otherwise, the sequence of words between DO and LOOP is executed. When LOOP is reached, execution of LOOP causes the lower limit LOW to be incremented by 1 and then compared to the upper limit HIGH. If LOW is equal or greater than HIGH, the loop is terminated with control passing to the word following LOOP; otherwise, control passes to the first word of the sequence following DO for another execution of the sequence.

HIGH LOW DO W1 W2 . . . Wn INCR + LOOP This is identical to the previous "do loop" with the difference that the lower limit LOW is incremented by the number INCR on top of the stack. INCR is normally a positive number.

For both types of do loops, the current value of the loop index is available for programming use through the word I. If the loop is nested, the word I always contains the value of the innermost index. The next outer indices are available through words J and K. The word I' contains the current value of (HIGH + LOW – I – 1); this may be used to run an index backward from (HIGH – 1) to LOW. The words J' and K' are similarly defined. When the parentheses form of iteration is nested with do loops, the parentheses form count as one level of indexing; the word I used within the range of a parentheses iteration will return the current value of the iteration count which runs from its initial value N down to 1.

For cases where the limits of iterations must be considered as unsigned numbers, unsigned parentheses iterations or do loops are used. These are as follows:

N U (W1 W2 . . . Wn)	unsigned iteration
HIGH LOW UDO W1 W2 . . . Wn ULOOP	unsigned do loop.
HIGH LOW UDO W1 W2 . . . Wn INCR U+LOOP	unsigned do loop.

The word EXIT causes the innermost loop in which it is imbedded to terminate unconditionally on the next cycle. The same effect holds true if EXIT is used within a parentheses iteration. Examples:

'DINGDING : 2 (DING) ; This definition is equivalent to the definition: 'DINGDING : DING DING ;
In either case, executing DINGDING will cause the word DING to be excuted twice.

'SPACES : (SPACE) ; This defines the word SPACES. Execution of 20 SPACES will cause the word SPACE to be executed 20 times (i.e., send 20 spaces to the user's console device).

'EXAMPLE : BEGIN 1– DUP DUP = EQZ END DROP ; This defines the word EXAMPLE which may be used as follows:
0 > 5 EXAMPLE 4 3 2 1 0
Each time through the iteration, the top of the stack (initially the number 5) is decremented by 1, printed, and compared to zero. If it is not zero the iteration is repeated. When it reached zero the iteration is terminated.
BEGIN EOF NOT IF READ-RECORD REPEAT This program line may be used to read a file. EOF returns a–1 if end of file has been encountered; a 0 otherwise. READ-RECORD reads the next entry of the file. By testing at the beginning of the iteration, the case of a zero length file is handled correctly.
5 0 DO I = LOOP This program line causes the numbers from 0 to 4 inclusive to be printed at the console.
5 0 DO 5 0 DO J 5 * I + = LOOP CR LOOP This program line causes the numbers from 0 to 24 inclusive to be printed as 5 lines of 5 numbers each.
5 0 DO I' = LOOP This prints the numbers 4 to 0 inclusive.
0 21 1 DO I + DUP = 2 +LOOP DROP This prints the first 10 (radix DECIMAL) perfect squares.

When using the index I' (or J' or K') within a "+LOOP" type loop, the upper limit HIGH should be replaced by (HIGH – INCR + 1) if it is desired to produce the same set of indices as with I. For example:

```
0 > 24 0 DO I = 4 +LOOP
0  4  8  12  16  20

0 > 24 0 DO I' = 4 +LOOP
23  19  15  11  7  3  1

0 > 24 4 – 1 + 0 DO I' = 4 +LOOP
20  16  12  8  4  0
```

In the second example above, HIGH=24 and therefore the first I' = 24+0–0–1 = 23 which is different from the last index in the first example. In the last example, HIGH=24–4+1=21 and therefore the first I'=21+0–0–1=20. Note that the last I' = 21+0–20–1 = 0.

CONCLUSION

This, the first part of the series, covered mainly the structure, syntax, some of the words in the basic vocabulary, and basic operations in the interactive mode. The compiling mode, which was only touched upon above, will be expanded in the next part of the series to cover assembly code and data type definitions, along with descriptions of more advanced constructs and words in the basic vocabulary.

For users of 8080/Z80 microcomputers with either single or double density North Star Disk as the mass storage medium, a complete STOIC package including diskette, source documentation, and user's guide to STOIC programming is available from B. W. Lee, 1385 Rifle Range Rd., El Cerrito, Calif., 94530, for $75 post paid. Specify single or double density.

Continued from pg. 19

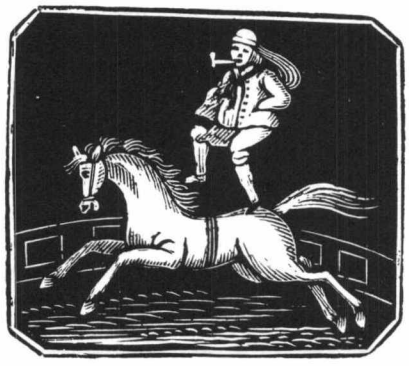
occurs on an Indian reservation, or other special circumstance. Some Federal prosecutors tend to forget the limits of their jurisdiction, and some local prosecutors mischarge; even so, they win "computer crime" cases anyhow. Not one, where real crime occurred, has been lost. It is not fair for prosecutors to blame their errors on the computer.

Mr. Ingraham invites me to leave off criticizing "embryonic" legislation and instead to help propose a better law. This I cannot do because the case for a computer crime bill has not been made.

I assure Mr. Ingraham that there is no such thing as a *droit de programmeur*. The machines are owned by our employers, and programmers do not have the right to make use of another's property that is not authorized by the owner. However, I know of only two companies that forbid incidental non-pecuniary personal use so that their employees know that it is forbidden; IBM and Aetna Life, and at that, only recently. No doubt there are others as well. Emphasis on my part may have misled Mr. Ingraham, but the truth is, neither the industry nor the profession has been aware of it as a problem consequently we haven't addressed it.

Programmers that I know, and I must have at least nodding acquaintance with several thousand now, are conscientious, intelligent, highly creative, and dedicated people. They deserve Mr. Ingraham's, and the U.S. Senate's, confidence.

Note: I have extracted these comments mostly from a paper by Susan Nycum, an attorney who has specialized in criminal law relating to computers. I remind the reader that I am a programmer, not competent in law. The reader should consult an attorney for any questions of law. – *JT*



DAL:
Data Acquisition Language

BY GREGG MARSHALL
P. O. Box 3282
Walnut Creek, CA 94598

Recent developments in the semiconductor and display industries have led to low cost, low power requirement components suitable for use in a portable data acquisition (capture) device. DAL is a language designed to control a portable data capture device, a portable terminal which allows the user to record data at the point of origin, later transmitting the data to a "host" computer.

We are presenting for your use and information the User's Manual. Due to the large size of the listings, we cannot publish them in Dr. Dobb's Journal *– they would take up more than one issue. However, Gregg Marshall has agreed to furnish them to anyone interested for $10, the cost to him of printing and mailing. Write to Marshall at the above address.*

INTRODUCTION

This manual is an introduction to the programming language DAL, which stands for a Data Acquisition Language. DAL is a very simple programming language. However, it provides everything needed for its application: data acquisition.

Gregg E. Marshall is a Scientific Programmer at Varian Aerograph in Walnut Creek, California. He is currently involved in developing microprocessor-based separation sciences instrumentation. Marshall holds an advanced degree in EE from the University of Colorado where his thesis topic was "A Floppy Disk Based File Management Peripheral."

We assume the reader has experience with computers, hopefully with programming. In particular, we assume the reader is familiar with data entry and has exposure to BASIC. If you lack sufficient background in data entry, we recommend chapter 12 in Mader and Hagin's *Information Systems.*

DAL is designed to control a portable Data Capture Device. A data capture device is a kind of portable terminal which allows its user to record data on the spot and later transmit that data to a "host" computer–in a form the computer understands. An example of a data capture device application is during a store's physical inventory. Generally a clerk counts the number of items actually present and records the part number plus the count. If this actual inventory count is to be utilized by a computer, it must be keypunched and possibly verified. Wouldn't it be nice if the actual inventory count could be directly entered into the computer? The problem with direct data entry is most computers and/or computer terminals are not that portable!

When using a data capture device, it is not only possible to record the part number and actual inventory count so the computer can "read" it, but also to reduce the number of errors. Reduce errors, you say: HOW? Since the data capture device is "intelligent," it can check and double check data values on the spot and ask for verification if a value seems odd. Interested in how this kind of data capture device can be programmed? Read on!

DAL is not intended to be used for extensive computations such as those required for data analysis, but rather to conveniently control the data acquisition process. Typical actions being performed utilizing DAL are the implementation of a simple command structure or simple checking of data input. It should be emphasized that DAL is designed only for data capture; no data analysis should be involved.

By recognizing that this is a limited application context, it seems reasonable to implement an extremely restricted

language. These restrictions should have several beneficial results. First, the size and development cost of the interpreter should be minimized because of the simplicity of the resulting language. Second, by only providing mechanisms appropriate to the task of data capture, it is hoped potential users will be discouraged from misusing the device for unsuitable purposes, namely online data analysis. Finally, the training of users will be easier since there are fewer constructs to learn.

The remainder of this manual is divided into a number of sections. The next section will present and discuss a sample DAL data acquisition program, in which each of the ten DAL statements will be briefly discussed. Following the example section, there will be sections presenting the DAL statements in greater detail.

A DAL EXAMPLE

For this example we will use the simple store physical inventory mentioned in the introduction. We assume the clerk counts the number of an item actually on the shelf and records the part number and number of items. In this case, however, the clerk will record the information in the memory of the data capture device instead of on paper.

While this may seem like an artificial example, it is similar to what might be done in the real world. While looking at this example you should be asking yourself what kind of data acquisition program is needed for your application?

In order to illustrate many of the capabilities of DAL and a data capture system, while still keeping this first example simple, we will make some assumptions. First, the user can either update the count for a given part number or the user can "dump" the recorded data to the host computer. Second, part numbers are alphanumeric values of ten characters or less and are unique in the first four characters. Third, the number of an item actually present on the shelf can never exceed several million. Fourth, to log onto the host computer requires only the following dialog:

ENTER USER NUMBER: xxxx

where xxxx is the number to be input by the data capture device to be recognized by the host computer. Fifth, to input the data into a host computer file only the following command is needed:

TAPE, DATA

Following this command, data is accepted until

END

is input. Sixth, to log off the host computer, the data capture sends

LOG

Operation of the system consists of turning on the data capture device and typing R for run. After initializing the variables and the array the device displays:

PART # (NEG TO DUMP)

If the user inputs a negative number, the program assumes the data capture device is connected to the host computer via a telephone. The data capture device then logs onto the host computer, transfers the data and logs off. If the user inputs a number, it is assumed to be a part number. The program looks in the array to see if the part has already been entered. If it has not, the program inserts the part number after the part numbers already entered and asks for a description and the initial quantity on hand. Otherwise, the device displays:

xxxx NOW yyyy CHANGE TO?

where xxxx is the part number and description entered by the user and yyyy is the part count already entered or zero. After entering the new count the program repeats the above sequence without initialization.

Having briefly described the overall logic of the program, let's take a look at the actual DAL program. Figure 2-2 is a listing of the program as it would be stored in the data capture device. But to make the program easier to read and understand, we have expanded it by spelling out the keywords and adding spaces as in Figure 2-3. Actually, Figure 2-3 is the form in which most DAL users will write their programs. Once the program is written it can be converted into the form of Figure 2-2 by a BASIC program running on the host computer.

The programs in figures 2-2 and 2-3 are equivalent and closely follow the flowcharts of Figure 2-1. The variables and array are initialized by setting the maximum array index to zero so that the first part number entered will not match anything in the array. This is done using the LET statement which assigns the value of 0 to the DAL variable M.

See how simple the program is? Using the details contained in the next sections, you should now be able to write your own data capture device programs with only a little practice.

VALUES, VARIABLES AND EXPRESSIONS

There are two kinds of values in DAL: numeric and alphanumeric (string). The numeric values are 32 bit signed integers in the range -2147483653 to 2147483653. Numeric values are most commonly used in calculations and when the input data is numeric. Generally numeric values should be used for numeric data since the internal representation of numeric values takes less space than the equivalent alphanumeric form. Alphanumeric values are strings of text that contain numbers, letters, and all the special characters on the keyboard except carriage returns and escapes. One of the powerful features of the data capture device that DAL runs on is the ability to input, display and store alphanumeric values. This avoids the error-prone practice of using numeric codes for inherently alphanumeric data.

The memory within the data capture device is divided into three main areas. The first area is contained in eraseable programmable read only memory (EPROM) and contains the DAL interpreter and the user's DAL program. The next two areas are contained in random access memory (RAM) which can be stored into and read out of. These two areas are the simple variables and primary data storage.

DAL has 25 simple variables denoted by the single letters B through Z. The first of these variables, B, has a special pre-

```
005F4,8,4
010T230
020LM=1
025LB=0
027D@
030D"PART # (NEG TO DUMP)"
040AT
045IO>T!G230
050D@
080LB=1
090IB=M!G122
100IA(1)=T!G170
110LB=B+1
120G090
122D"DESCRIPTION -- "
124A$A(2)
126LA(1)=T
130D@
132D"ON HAND -- "
134AA(3)
136LM=M+1
140D@
142G025
170D"# "
172DA(1)
173D$A(2)
174D" NOW "
176DA(3)
178D" CHANGE TO? "
180AA(3)
190D@
200G025
230D@
240R$C
250IC="ENTE"!G280
260D"LOG-ON ERROR  REDIAL AND TRY AGAIN"
270G030
280S"1345"
290RC
300S"TAPE,DATA"
305S@
310D"TRANSFER IN PROGRESS"
330LB=1
340IB=M!G390
350SA(1)
352S@
355S$A(2)
357S@
360SA(3)
365S@
370LB=B+1
380G340
390S"END"
395S@
400RC
410S"LOG"
415S@
420RC
430D"TRANSFER COMPLETE"
440Q
```

Figure 1. DAL Programming Example

```
5 FORMAT 4,8,4
9 ! INITIALIZE THE VARIABLE AND THE ARRAY
27 DISPLAY @
10 TRAP 230
20 LET M=1
25 LET B=0
30 DISPLAY "PART # (NEG TO DUMP) "
40 ACCEPT T
45 IF 0 > T GOTO 230
50 DISPLAY @
70 ! PART NUMBER, IN ARRAY ALREADY?
75 ! FIRST 4 CHARACTERS MUST BE UNIQUE
80 LET B=1
90 IF B=M GOTO 122
100 IF A(1)=T GOTO 160
110 LET B=B+1
120 GOTO 90
121 ! NEW PART, GET DESCRIPTION AND QUANTITY ON HAND
122 DISPLAY "DESCRIPTION -- "
124 ACCEPT $A(2)
126 LET A(1) = T
130 DISPLAY @
132 DISPLAY "ON HAND -- "
134 ACCEPT A(3)
136 LET M = M + 1
140 DISPLAY @
142 GOTO 25
160 ! NOW GET NEW COUNT AND LOOP BACK
170 DISPLAY "# "
172 DISPLAY A(1)
173 DISPLAY $A(2)
174 DISPLAY " NOW "
176 DISPLAY A(3)
178 DISPLAY " CHANGE TO? "
180 ACCEPT A(3)
190 DISPLAY @
200 GOTO 25
210 ! DUMP ROUTINE
220 ! ASSUME PHONE CONNECTION ALREADY MADE SO LOG ON
230 DISPLAY @
240 RECEIVE $C
250 IF C="ENTE" GOTO 280
260 DISPLAY "LOG-ON ERROR  REDIAL AND TRY AGAIN"
270 GOTO 30
280 SEND "1345"
290 RECEIVE C
300 SEND "TAPE,DATA"
310 DISPLAY "TRANSFER IN PROGRESS"
320 ! SEND DATA, 1 ITEM PER LINE
330 LET B=1
340 IF I=M GOTO 390
350 SEND A(1)
352 SEND @
355 SEND $A(2)
357 SEND @
360 SEND A(3)
365 SEND @
365 LET B=B+1
370 GOTO 340
380 ! DONE WITH DATA, LOG OFF
390 SEND "END"
395 SEND @
400 RECEIVE C
410 SEND "LOG"
415 SEND @
420 RECEIVE C
430 DISPLAY "TRANSFER COMPLETE"
440 QUIT
```

Figure 2. Extended DAL Programming Example

defined usage that will be discussed later in this section. The other 24 variables are entirely for the DAL programmer's use to contain either numeric values or alphanumeric values limited to 4 characters. Generally these variables will be used to control loops and to hold temporary numeric values during complex calculations.

Most of the RAM memory within the data capture device is dedicated to the primary data storage area. This large block of memory (also referred to as the array A) can be divided by the DAL programmer into a number of records. A record is a collection of numeric and alphanumeric fields. Each field holds one value. Thus in the first example, the physical inventory, each record consisted of two fields: a ten character alphanumeric field that contained the part "number" and a numeric field that contained the number of that part on the shelves. The structure of a record is defined by a FORMAT statement which is discussed in a later section.

There are some restrictions on how the records may be defined. First, every record within the primary data storage area is the same. It is not possible to simultaneously define two or more records structures. Second, each record must have at least one field and may not have more than 15 fields. Each of these fields may be numeric or alphanumeric. A field must be at least four bytes long but may not be more than 255 bytes long. A numeric field requires exactly four bytes. Alphanumeric fields may be any length between 4 and 255 bytes; each character requires one byte. For alphanumeric fields, the string value may be shorter than the defined length but may not exceed that length. Finally, the sum of the lengths of the fields within a record may not exceed 255.

Each simple variable is denoted by a single letter B through Z. The primary data storage area is denoted as the singly subscripted array A (the letter A followed by an index expression contained in parentheses). The value of the simple variable B selects the record of the primary data storage area to be used. The index value selects the field within that record. Note that the index selects the field (1-15) and not the byte offset within the record. Similarly the value of B selects the record (0 to m-1, where m is the primary data storage area size divided by the record size), and not the byte displacement within the primary data storage area.

Expressions in DAL are composed of operators, variables and constants. A DAL expression is evaluated from left to right without parentheses or operator precedence. An expression consists of alternating operands and operators. Expressions always begin and end with operands. Thus legal expression forms are: operand, operand operator operand, operand operator operand operator operand, etc. An operand is: a variable, a constant, or a variable or a constant preceded by a minus sign (-). Variable denotions have been described earlier in this section. Constants are either numeric (a string of digits) or alphanumeric (exactly four characters enclosed by quotes).

There are six DAL operators. These operators are: addition (+), subtraction (-), multiplication (*), division (/), test for greater than (>), and test for equality (=). All DAL operators assume signed twos complement integers as operands. Literal strings are converted to integers as described in the section on the IF statement. The two relational operands (>,=) return zero if the test is false, and one if the test is true. While unary negation is allowed, it is considered to be an operand modification and thus is not part of the DAL operator processing.

THE ACCEPT STATEMENT

Format:
```
Avariable
ACCEPT variable          (extended DAL)
```

Examples:
```
AR
ACCEPT R                 (extended DAL)

A$A(J)
ACCEPT $A(J)             (extended DAL)
```

This statement is used to read data from the data capture device's keyboard. This statement is DAL's way of listening to the user (the DISPLAY statement is DAL's way of talking). The data is stored in the variable specified.

The ACCEPT statement can be used to input either numeric or string values. Values typed in are generally terminated by a carriage return. Each string has a maximum length associated with it. If, during input, that string is filled, input is terminated. String variables are preceded by a dollar sign ($) value. Variables not preceded by a dollar sign are numeric variables. Only the digits 0 to 9 and an optional leading minus sign (-) may be input, and input is terminated by the first non-numeric character.

For either string or numeric input variables, if the first character is a carriage return, the value returned is - 2147483654. This allows the DAL programmer to check for null inputs. If the variable was a string variable, the first four bytes of the string contain this value. This feature allows the user to skip over certain fields in the input data and still distinguish between omitted data and zero values.

All input characters are echoed on the data capture device's display regardless of whether the source of the characters is the keyboard or the host computer interface. This allows the user to monitor host computer communications.

A limited input data editing facility is provided by DAL. If, during input, the user types a Control H, the last character input will be deleted. Control H may be repeated, as needed until the entire input value is deleted. If there is not input value when a Control H is typed, the Control H is ignored.

The DISPLAY Statement

Format:

```
Dprintlist
DISPLAY printlist                (extended DAL)
```

Examples:

```
D"ENTER NEXT COMMAND"
DISPLAY "ENTER NEXT COMMAND"     (extended DAL)

DV
DISPLAY V                        (extended DAL)

DE*F
DISPLAY E*F                      (extended DAL)

D$B
DISPLAY $B                       (extended DAL)
```

D@
DISPLAY @ (extended DAL)

The DISPLAY statement is one of the statements used to communicate with the user. This statement is used for output or communicating from the data capture device to the user. It does this by causing strings and data values to be transferred to the alphanumeric display.

Each DISPLAY statement can output only one of the following:

1. A literal string enclosed within quotes.
2. A string value of a variable.
3. A numeric value of an expression.
4. A carriage return/line feed (denoted by an @).

Literal strings are used to output prompts and other textual information to the alphanumeric display. A literal string begins with a quote and may contain any character except a carriage return or a quote. The string is terminated by a second quote. Unlike many programming languages, DAL makes no provision for imbedding quotes within literal strings.

DAL does not distinguish between numeric and string values (which are limited to 4 characters for simple variable) during any of its computations. The only place DAL distinguishes between the two types is during the input/output statements (ACCEPT, DISPLAY, RECEIVE and SEND). The input statements are discussed in another section. In output statements, DAL considers all values (even the value of "ABCD") to be numeric unless otherwise informed. The way to inform DAL that a value is alphanumeric is to precede it with a $. This is analogous to BASIC's representation of strings except that instead of flagging each variable with a $ and providing special operators, DAL only flags the result as being a string. In some respects this is dangerous; DAL would just as soon output a value as either a string or as a number—thus you can "calculate" many strings using only numbers and some knowledge of the ASCII codes for the characters.

DAL can be used to output numeric values too. For greater flexibility and ease in programming, expressions may be included in the DISPLAY statement. The expression is evaluated and the value is converted to an equivalent string of characters and output to the alphanumeric display. The value is output without leading zeros and if the value is negative, it is preceded by a minus sign (−). The DAL programmer is again cautioned that DAL makes no distinction between numeric and string values.

Normally, each DAL DISPLAY statement appends its output at the end of the text that has already been output. This way lines of text can be created using several DISPLAY statements allowing a mixture of strings, variable values, and user inputs. When a DAL DISPLAY statement contains an @ symbol instead of one of the previous printlist possiblities, a carriage return and a line feed will be output. This will cause the alphanumeric display to clear immediately and text to begin at the left hand side with the next DISPLAY statement.

The GOTO Statement

Format: Glinenumber
GOTO linenumber (extended DAL)

Example: GO10
GOTO 10 (extended DAL)

The GOTO statement is used to transfer control to a different part of the program. The line number must be exactly three digits except in the extended DAL form, where line numbers are from one to three digits.

A GOTO in extended DAL may jump to a comment statement. In that case, during the translation to standard DAL, the line number will be changed to reference the first executable statement following the comment.

The IF Statement

Format: Iexpression!Glinenumber
IF expression GOTO linenumber (extended DAL)

Examples: IJ > 3!GO20
IF J >3 GOTO 20 (extended DAL)

IR = "DUMP"!GOTO40
IF R = "DUMP" GOTO40 (extended DAL)

IB!G100
IF B GOTO 100 (extended DAL)

The IF statement is used for testing various conditions in DAL and altering the flow of control. The value of the expression is calculated. If that value is zero (false), execution of the DAL program continues with the next statement. If that value is nonzero and not a multiple of 256 (true), control is tranferred to the line number via a GOTO. Notice that the value must not be a multiple of 256 for the IF statement to work properly.

Note that a relational operator (>, =) may or may not be present in the expression and that strings may be included in the expression. The user is cautioned about comparing strings to numeric values. String comparisons are based on the ASCII collating sequence. When comparing the string to a numeric value, the string's value is:

CHR(character4)*16777216 + CHR(character3)*65536 + CHR(character2)*256 + CHR(character1)

where the characters are numbered from left to right and the function CHR gives the ASCII code representing the character.

The LET Statement

Format: Lvariable=expression
LET variable= expression (extended DAL)

Examples: LA(J)=B+C*D
LET A(J) = B+C*D (extended DAL)

```
LS = "ABCD"
LET S = "ABCD"          (extended DAL)

LS="AB  "
LET  S = "AB"           (extended DAL)

LC = S
LET  C = S              (extended DAL)
```

The LET statement is used to change the value of a DAL variable by assigning to it the result of a computation or the value of a string. The LET statement provides a way of arithmetically manipulating the input data during error checking and for maintaining indices into the array for data storage.

At this point we would like to again caution the user from trying the misuse the data capture device on which DAL is designed to run. DAL, and the data capture device, are designed for data acquisition, not data analysis. If you are manipulating the data before storing it, you are probably doing too much data analysis during data acquisition.

The RECEIVE Statement

Format:

```
Rvariable
RECEIVE variable        (extended DAL)
```

Examples:

```
RR
RECEIVE R               (extended DAL)

R$A(J)
RECEIVE $A(J)           (extended DAL)
```

This statement is used to read data from the data capture device's host computer interface, and is DAL's way of listening to the computer (the SEND statement is DAL's way of talking). The data is stored in the variable specified.

The RECEIVE statement can be used to input either numeric or string values. Values typed in are generally terminated by a carriage return. Each string has a maximum length associated with it. If, during input, that string is filled, input is terminated. String variables are preceded by a dollar sign ($) to indicate to the interpreter that input is to be a string value. Variables not preceded by a dollar sign are numeric variables. Only the digits 0 to 9 and an optional leading minus sign (–) may be input. Input is terminated by the first non-numeric character.

For either string or numeric input variables, if the first character is a carriage return, the value returned is –2147483654. This allows the DAL programmer to check for null inputs. If the variable was a string variable, the first four bytes of the string contain this value. This feature allows the user to skip certain fields in the input data and still distinguish between omitted data and zero values.

All input characters are echoed on the data capture device's display regardless of keyboard or host computer interface source of characters. This allows the user to monitor host computer communications.

A limited input data editing facility is provided by DAL. If the user types a Control H, the last character input will be deleted. Control H may be repeated until the entire input value is deleted. If there is not input value when a control H is typed, it is ignored.

The SEND Statement

Format:

```
Sprintlist
SEND Printlist          (extended DAL)
```

Examples:

```
S"ENTER NEXT COMMAND"
SEND "ENTER NEXT COMMAND"   (extended DAL)

SV
SEND V                  (extended DAL)

SE*F
SEND E*F                (extended DAL)

S$B
SEND $B                 (extended DAL)

S@
SEND @                  (extended DAL)
```

The SEND statement is one of the statements used to communicate with the host computer. This statement is used for output or communicating from the data capture device to the host computer. It does this by causing strings and data values to be transferred to the host computer interface.

Each SEND statement can output only one of the following:

1. A literal string enclosed within quotes.
2. A string value of a variable.
3. A numeric value of an expression.
4. A carriage return/line feed (denoted by an @).

Literal strings are used to output prompts and other textual information to the host computer interface. A literal string begins with a quote and may contain any character except a carriage return or quote. The string is terminated by a second quote. Unlike many programming languages, DAL makes no provision for imbedding quotes within literal strings.

DAL does not distinguish between numeric and string values (which are limited to 4 characters for simple variables) during any of its computations. The only place DAL distinguishes between the two types is during the input/output statements (ACCEPT, DISPLAY, RECEIVE and SEND). Input statements are discussed in another section. In output statements, DAL considers all values (even the value of "ABCD") to be numeric unless otherwise informed. The way to inform DAL a value is alphanumeric is to precede it with a $. This is analagous to BASIC's representation of strings, except instead of flagging each variable with a $ and providing special operators, DAL only flags the result as being a string. In some respects this is dangerous: DAL would just as soon output a value as either a string or a number. You can thus "calculate" many strings using only numbers and some knowledge of the ASCII codes for the characters.

DAL can also be used to output numeric values. For greater flexibility and ease in programming, expressions may be included in the SEND statement. The expression is

evaluated and the value is converted to an equivalent string of characters and output to the host computer interface. The value is output without leading zeros and if the value is negative, it is preceded by a minus sign (-). The DAL programmer is again cautioned that DAL makes no distinction between numeric and string values.

Each DAL SEND statement normally appends its output at the end of text already output. In this way, lines of text can be created using several SEND statements which allow a mixture of strings, variable values, and user inputs. When a DAL SEND statement contains an @ symbol instead of one of the previous printlist possibilities, a carriage return and a line feed will be output.

The FORMAT Statement

Format:	Fdeflist	
	FORMAT deflist	(extended DAL)
Examples:	F4, 4, 20	
	FORMAT 4, 4, 20	(extended DAL)
	F4, 10, 15, 4	
	FORMAT 4, 10, 15, 4	(extended DAL)

The FORMAT statement is used to define the structure of the records within the primary data storage area. As discussed in the values, variables and expressions section, the primary data storage area is divided into records which are divided into fields. The DAL programmer does not specify the type of the fields (numeric or alphanumeric), but the length of the fields. A given field might be used as a numeric field in one part of a DAL program and as an alphanumeric field elsewhere.

The FORMAT statement enumerates lengths of field within a record. The number of lengths in the definition list defines the number of fields; the sum of lengths defines record size. The second example defines a record with four fields that is 33 bytes long. Most likely, the field first is numeric, the second two are alphanumeric, and the third is again numeric. However, there is no restriction on the use of fields. By convention, numeric fields are four bytes long and alphanumeric fields are greater than four bytes.

The lengths within the FORMAT statement may be any valid DAL expression. This feature makes it possible to dynamically allocate the record at execution time by inputting the field lengths before the FORMAT statement. This feature should be used with caution since the resulting structures may be undocumented and easily forgotten.

The QUIT Statement

Format:	Q	
	QUIT	(extended DAL)
Example:	Q	
	QUIT	(extended DAL)

The QUIT statement is used to terminate the execution of a DAL program and return control to the keyboard monitor. Every DAL program should end with a QUIT statement. A DAL program may contain more than one QUIT statement.

The TRAP Statement

Format:	Tlinenumber	
	TRAP linenumber	(extended DAL)
Example:	T120	
	TRAP 120	(extended DAL)

The TRAP statement is a specialized conditional control statement somewhat related to the IF statement. The TRAP statement is used to transfer control to a different part of the program when an escape is encountered during input. The primary use for trapping escapes is to provide a "help key" for the ultimate user. This key might, for example, list the possible input values or a description of the value to be input. The line number must be exactly three digits (except in the extended DAL form, where line numbers are from one to three digits).

A TRAP in extended DAL may jump to a comment statement. In that case, during translation to standard DAL, the line number will be changed to reference the first executable statement following the comment.

The BACK Statement

Format:	B	
	BACK	(extended DAL)
Example:	B	
	BACK	(extended DAL)

The BACK statement is used to resume processing after a trap has been processed. During the execution of a trap, after encountering an escape on input, the line number of the input statement is saved. This allows the DAL programmer to return to the input statement after, for example, outputting a desired description. If a BACK statement is encountered without executing a trap, the statement is ignored and execution continues with the next DAL statement.

REPRESENTATION OF A DAL PROGRAM WITHIN THE DATA CAPTURE DEVICE

The user's DAL program, once translated from extended DAL into compact DAL form, is stored within the data capture device in erasable programmable read only memory (EPROM). The DAL interpreter, which implements the DAL functions, also resides in EPROM. The data capture device has a capacity for over six thousand characters of EPROM storage. Two thousand characters of this capacity is dedicated to the DAL interpreter, leaving four thousand characters for the user's DAL program. As an indication of the capabilities of DAL and the current data capture device, the example program in this manual requires about seven hundred bytes of storage. The first EPROM address for the user's DAL program is E800 (hex). DAL programs must always be stored in compact form starting at this address. The user's DAL program is stored as a sequence of ASCII characters. These characters must have zero parity (the most significant bit is always zero). Each line is terminated by an ASCII carriage return. In addition, the first character in a DAL program is a carriage return.

Complex Pseudorandom Sequences from Interlaced Simple Generators

BY H. T. GORDON
College of Natural Resources
University of California
Berkeley, CA 94720

© 1979, H. T. Gordon

My first effort at generation of a sequence of pseudo-random numbers (*DDJ* # 32) was primarily aimed at minimizing the *time* required to generate successive 8-bit binary numbers. The basic logic outputs a non-repeating sequence (NRS for short) of all 256 possible numbers. This "core" sequence can be transformed into a limited number of variants by simple mathematical operations. Such variations can be *concatenated* to yield a longer NRS. However, this logic entails a great sacrifice of the *quality* of the numerical sequence, that is easily recognizable as non-random and is not difficult to decipher.

Interest in random-number generation has been increasing in the microcomputing community. Bungay and Martin (*KILOBAUD* # 28, p. 46) designed a *hardware* generator controlled by a "noisy" clock, that is said to yield a "truly" random sequence of 8-bit numbers. Timing was not given, but was implied to be slow although fast enough for use with a BASIC interpreter. The unpredictability of the next number in the sequence, however, disqualifies such generators for some important applications, e.g. "hashing" algorithms or cryptography.

All this has tempted me to create more complex generator logic, so that the output will be a closer approximation to "true" randomness. Since the instruction sets of 8-bit microprocessors are inefficient for "number-crunching", complexity is achieved by *interlacing* the output of n independent simple generators. The concept is probably not novel, although its value for enhancing the quality of a pseudorandom sequence is not common knowledge (I did not find it in the elaborate discussion of random numbers in Knuth's *Seminumerical Algorithms*, perhaps because it's too simple.) An interlacing algorithm is easy to implement in microprocessor code, and makes possible a planned *synthesis* of very diverse numerical outputs with extremely long NRS.

Simulation of "True" Randomness

Although I have no expertise in probability theory, a few principles are demonstrable at an elementary level:

1. The occurrence of any sequence of 256 all-different numbers is wildly improbable, even though the number of possible sequences of this type is 256 factorial, for which Stirling's approximation gives the astronomical number 2.43×10^{525}. This is dwarfed by the total number of all possible sequences, 256^{256} or 3.23×10^{616}. The probability of an all-different sequence is an infinitesimal $7.5 \times^{-92}$, and no self-respecting simulation should ever generate such a sequence! Other *kinds* of sequences have immensely higher probability, that can be calculated from formulas for permutations and combinations. There are so many types of sequences that it is impractical to do such calculations for any significant fraction of them. For the type consisting of 192 different numbers, of which 64 are duplicated, there exist 1.03×10^{572} possible sequences, and the probability of occurrence (immeasurably higher than that of the all-different sequence!) is still a ridiculously low 4.2×10^{-45}. The maximum probability (still abysmally low) for this type is near 187 different numbers, with 1.6×10^{574} possible sequences. This analysis, inadequate though it is, suggests that generator design should aim at a zero-order heterogeneity (defined as H_0 in my previous *DDJ* article) near 187.
2. The probability that any given 8-bit number will *not* be the next number in a random sequence is 255/256, roughly 0.99609, and that it will not occur in the next n numbers is $(99609)^n$. In a one-page sequence, the probability that a given number will not occur is 0.3672; therefore the probability that it *will* occur is only 0.6328 – very far from certainty! The probability of its occurrence rises to 0.8652 for 2 pages (512 consecutive numbers), 0.9505 for 3 pages,

and 0.9818 for 4 pages. I am unable to imagine program logic that would even approximate these specifications, even if it were inordinately complex. One can guess that realistic logic should try *not* to include every possible 8-bit number in any 3-page (or even longer) output sequence. This accords with an H_0 value of 187, that would exclude 69 of the possible numbers from any one sequence page.

3. One could extend the above line of reasoning to the 16-bit level, viewing 2 successive outputs as one number. The probability of occurrence of any one of the 65536 possible numbers is of course very low, and the calculations would be tedious. One can guess that only a fraction of the possible numbers will occur even if the NRS exceeds any realistic limits. On the other hand, frequence recurrence of the same number is improbable. The proper strategy may be to *try* to generate as many different numbers as one can, and generate as long an NRS as one is likely ever to require.

An Experimental Generator Algorithm

Logical complexity is a perilous sea. More program and working memory, worse timing, diminishing returns, poor predictability, formidable testing problems, and the many other evils that complexity so generously bestows!

My program MIXSIM (cf. listings, 6502-coded) presents one way of interlacing the output of *n* simple generators (each one yielding a different sequence of the 256 different 8-bit binary numbers) into a synthesized sequence with an NRS of $n(256)^n$, or more than 50 million if $n = 3$. The interlacing logic is not very complex. Location MEMEX "remembers" the value of the X index register that selected the generator used in the previous subroutine call. This is reloaded into X and decremented to shift control to the next lower generator in the sequence. If MEMEX was zero, however, the X register is RESET. In the example X becomes 02, and generator G_2 always outputs its "next number", resetting MEMEX to 02 just before exit. The *next* call will use X = 01. It first loads the previous G_2 output into the accumulator, and if this was *not* zero branches to the normal generator logic to output the "next number" of the G_1 sequence (resetting MEMEX to 01). If the previous G_2 was zero, normal G_1 generator logic will be *bypassed* and the output will be created by a "pseudo-generator". The LDA RND+2,X instruction is an "antibug" that guards against a *double-bypass* of normal generator logic. It loads the content of the *next higher* location to test whether it is also zero. When G_1 is being tested, it is trying to load the "previous output" of the non-existent G_3, that will in reality be a non-zero constant used by the logic of G_0. The BEQ is therefore not executed, and a "pseudo" number is fabricated by loading the previous G_1 output and forced-branching to DEJUM, that complements the 7 low-order bits. Since the G_1 "seed" is unchanged, the effect is to *offset* (or desynchronize) the G_2 and G_1 generators. The frequency of normal G_1 output is 255/256 that of the always-normal G_2 output. Therefore G_2 and G_1 will not *both* re-initialize until G_2 has output 256^2 numbers, and G_1 has output 255 X 256 normal and 256 pseudo numbers.

The logic also desynchronizes the G_0 generator, in a more complex way. The guard instruction ensures that all pseudo G_1 outputs will be followed by a normal G_0. Since 1/256 of the normal G_1 outputs will be 00, 255 pseudo G_0 outputs will occur for every 256^2 G_2 outputs, together with (255 X 256) + 1 normal G_0 outputs. Simultaneous re-initialization of all 3 generators therefore will not occur until 256^3 G_2 numbers have been output. The other generators will have output as many numbers, but 256^2 G_1 and 255 X 256 G_0 outputs will have been "pseudo".

The Logic of the Simple Generators

Although MIXSIM uses an X-indexed version of my earlier SIMRND generator, because I feel it has a near-optimal balance of speed, simplicity, and output heterogeneity, other types of generators might interlace as well. One requirement of interlacing is that each of the generators yield a distinctly different sequence of the 256 different numbers; otherwise the same number will show recurrence at regular intervals. The variant sequences are produced in MIXSIM by appending my earlier ADDRND module. The base location for the 3 addends is RND+3. Since all the variants have low heterogeneity in the low-order bits, this is improved by appending the "jumbling" module SIMJUM, with the added advantage that its DEJUM instruction can be used by the pseudo-generator logic.

Many distinct versions of MIXSIM are possible. The G_0 addend location (RND+3) can be any one of 255 numbers, since 00 is excluded to ensure unconditional operation of the G_1 logic. The G_1 addend can be any one of 255 numbers, since 00 is allowed, and the G_2 addend can be any one of 254 numbers. Since every permutation of addends yields a distinct long sequence, more than 16.5 million versions are possible. Perhaps some of these will yield a "better-looking" long sequence than others. I instinctively feel that addends with quite different bit-patterns are preferable, but instinct is far from an infallible guide in the jungles of complexity!

The Possibility of Greater Complexity

As one seeks to improve sequence quality by more complex logic, the number of *possible* ways becomes even larger and the difficulty of foreseeing flaws becomes greater. I see *no* way of creating a sequence that will satisfy the specifications of "true" randomness that I lightly touched on earlier in this note. What *may* be fabricable is a sequence that would be extremely difficult to decipher. MIXSIM is not. Its G_2 output appears regularly as the first of every sequence-of-three, and repeats after every 3 X 256 outputs. One fairly simple way of correcting this would be to add (following the RESET instruction) my earlier INCRND module, to increment the G_2 seed location at the start of each full G_2 sequence. This will increase the NRS of G_2 to 256^2 and (if there are no pitfalls!) the long-sequence NRS to billions of numbers. Checking this out would be tedious, even if the INCRND logic is not applied to any of the other generators, to avoid augmenting the number of initialization states beyond 256^4.

A less interesting approach would be to modify MIXSIM to use 4 (or more) different generators. This would retain the transparency of the G_2 output, but (if problem-free) greatly lengthen the NRS. One could of course go to *higher* levels of interlacing, with additional logic that would interlace the output of generators of the MIXSIM type. I remain skeptical that the resulting output could withstand even moderate analysis without revealing its pseudorandomness, even if the generating logic becomes undecipherable.

Some Rough Tests of the MIXSIM Output Sequence

If the initial state of MEMEX is 02, with seeds (G_0, G_1, G_2) of 00, 01, 02 and addends of $59, $A6, 00, and 10 successive sequences of 256 numbers are run, the zero-order heterogeneity (H_0) ranges from 176 to 184, with a mean of 180.7 and standard deviation of 3.13. Numbers that occur only once (in a set of 256) range from 106 to 120, mean 114.8 and s.d. 5.96. Numbers that are duplicated range from 52 to 60, mean 56.5, s.d. 3.10. Numbers that are triplicated range from 8 to 11, mean 9.4, s.d. 1.07. Even an analysis as simple as this is sufficient to prove pseudorandomness (or so it seems to me!)

A different analysis involved running MIXSIM continuously, counting the number of each of the 256 possible 8-bit outputs until one of them was output 256 times. With the same initial states used in the previous test, this test routine terminated after outputting the number $4C 256 times. All other numbers occurred from 190 to 194 times. The egregious surplus of a unique number among the over 49000 numbers in the long sequence is glaringly non-random. This odd result led me to change the seeds to 11, 22, 33 (with the same addends). This run yielded 256 $C8 values, with all other numbers occurring from 213 to 217 times. The "preferred number" is therefore determined by the values selected for the seeds. Changing the addends to 57, A6, 00 had no notable effect, $C8 still being preferred. Since the fundamental logic cannot cause such a preference in the entire NRS, I presume that the "preferred" number *changes* in various short sequences, but have not bothered to test this thoroughly. I did run MIXSIM in an endless loop for about 25 minutes, using the same initial conditions as in the first paragraph of this section, to get it about half-way through its NRS. When halted, MEMEX was 02 and the seeds had become 06, E3, 8D. Running the second test routine from that point yielded $10 as the "surplus" number, with the others occurring from 198 to 201 times. The inherent monotony of the output was revealed by one run of the first test routine, giving an H_0 of 183 with 119 once-occurring numbers, 55 duplicates, and 9 triplicates. So even an amateur cryptanalyst would know a lot about MIXSIM even from a sequence as short as 512 numbers! It's not that this *type* of sequence is so improbable. There are 4.8×10^{579} possible sequences of 118 singles, 57 duplicates, and 8 triplicates This type is therefore 10000 times more probable than the duplicates-only sequences I dealt with at the beginning of this note. What is incredible is that even 2 of them should occur in a truly random numerical sequence, let alone that *all* sequences should be of the same type! The simplicity of the underlying logic is transparent. A clever analyst (and any large organization is likely to have at least one brilliant analyst) would have no trouble deciphering it. Mere *length* of the NRS represents no genuine challenge. As with the simple generators I discussed in *DDJ* #32, the problem is the creation of *heterogeneity*.

The interlacing logic module is too simple to be undecipherable, but that was not its goal. It is fast, adding only about 16 cycles to the mean timing per output (that totals about 45 logic-only cycles). The cost in memory locations is greater: 38 program-logic and 7 working bytes. The output does represent a closer approximation to true randomness, since it consists of nearly 200,000 successive one-page sequences that are all different and of a type that is 10^{54} times more probable than the 256-different-number sequences of its underlying simple generators.

The next step may be much more costly and yield diminishing returns. It ought to be based on a much more detailed analysis of sequence probability, perhaps beyond the 256-byte level, so one will know what kinds of sequences need to be synthesized. Merely avoiding monotony of single-page sequences is not enough. The question is: what kind of variety is most probable?

Some Thoughts About the Role of *DDJ*

Few journals would dream of publishing this kind of disquisition on tiny algorithms. Computer scientists will deem it "trivial", a favorite pejorative computerese word. Mathematicians would (quite rightly) deprecate the lack of generality and abstract thinking. On the other hand, many amateurs might find neither the purpose nor the presentation crystal-clear. *DDJ* is for in-between types, like me. Feedback in response to some earlier articles has revealed a fascinating variety of human personalities, of the inventive type, most of them, not surprisingly, interested in useful code, often with a practical goal and a concern for optimal timing, re-enforcing my belief that it really matters, and that small modules have their own uses. This restores morale that sometimes erodes when I compare my *haikus* with the Homeric software epics I see everywhere!

A few correspondents are more interested in concepts than code, and some even empathize with the traces of idealism that at times surface in whatever I write. I suspect that latent idealism is at the heart of *DDJ* and sustains its open-to-all, unstructured and unhierarchic outlook, that contrasts so vividly with the *Weltanschauung* of the commercial or subsidized technical journals. These are run by some elite person or group, whose expertise enables them to detect and exclude obvious errors. That's a plus, but the expert human mind also tends to reject the unfamiliar and the unconventional. Domination by experts or an establishment tends to crush novel thought patterns (that will indeed usually be imperfect at birth). This is not yet a problem for the editors of *DDJ* (though they have lots of *other* problems!), who don't seem to know what heresy is. Still, the second law of thermodynamics is inexorably at work. Fires cool, and (if, as I hope, *DDJ* survives) one cannot but wonder into what pattern *DDJ* will crystallize.

Continued on pg. 41

Dumping Files Under FLEX

BY FREEMAN L. MOORE
Department of Computer Sciences
Mt. Pleasant, MI 48859

While memory dumps may be helpful in some debugging applications, there are occasional instances where one must use a file dump to ensure the contents are as expected. The following program, written for the SWTP 6800 system, is such a dump. It uses the FLEX operating system. The layout of the dump is similar to those produced on most large IBM systems, hexadecimal followed by the ASCII equivalent. Where a particular hexadecimal pattern does not correspond to a valid ASCII printable character, it is replaced by the period (.) prior to printing.

The needed entry points in FLEX are identified by EQU statements for which the equivalent routines should be available with other operating systems. The file is read as a data file, a sector at a time, and edited for output. Note that the variable LINE is set to the number of hexadecimal digits to print on a line. This may be changed depending upon the width of the output device. Using a value of 12 is ideal for a 64 character output device such as the SWTP CT-64. A value of 8 works for the SWTP PR-40 tiny printer. The correct value may be computed by:

4 x LINE = WIDTH

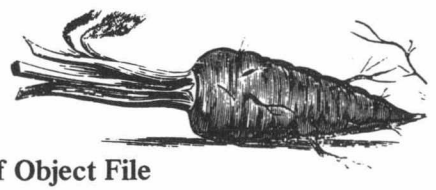
where WIDTH is the maximum width of the output device. It should be noted that FLEX assigns files in a linked-list fashion and that the user is never really aware where a file is physically stored. The program prints out the track and sector numbers kept by FLEX for each file in the first four bytes of a 128 byte sector. Once the actual track and sector is known, "disk-patching" is a relatively minor task, but a task not encouraged!

The ORG statement in the program defines the program to occupy storage beginning at location $7600, which is the utility command space available in FLEX. The program also takes advantage of the system file control block (FCB) found at location $7740.

Listing

```
+++DISKDUMP,DUMPTEST.TXT
DISK ADDRESS IS 1A 0D          This is a fi
54 68 69 73 20 69 73 20 61 20 66 69
6C 65 20 77 68 69 63 68 20 77 69 6C
6C 20 62 65 20 64 75 6D 70 65 64.
75 73 69 6E 67 20 74 68 65 20 73 79
73 74 65 6D 20 75 74 69 6C 69 74 79
20 44 49 53 4B 44 55 4D 50 2C 41 61
6E 64 20 69 73 20 63 61 70 61 62 6C
65 20 6F 66 20 64 75 6D 70 69 6E 67
20 61 6E 79 20 6B 69 6E 64 20 6F 66
0D 66 69 6C 65 20 61 6E 64 20 70 72
69 6E 74 69
DISK ADDRESS IS 1A 0E          ng its disk
6E 67 20 69 74 73 20 64 69 73 6B 20
61 64 64 72 65 73 73 0D 61 73 20 77
65 6C 6C 2E 0D 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
+++
*ASM, LP:DDJ*,WIDTH=80,NOSEQUENCE
*6800 RELOCATABLE CROSS-ASSEMBLER VER 2.5
PAGE 0001

                                    OPT     SYMBOL,MEMORY
                                    DISKDUMP
                                *
                                *   THIS PROGRAM WILL READ A FLEX FILE AND PRINT
                                *   ITS CONTENTS IN HEXADECIMAL AND ASCII ON THE
                                *   PRINCIPAL OUTPUT DEVICE.  NON-ASCII VALUES
                                *   ARE PRINTED AS THE PERIOD (.).
                                *
                                *   THE FILENAME IS SPECIFIED AS THE ARGUMENT
                                *   ON THE PROGRAM CALL LINE AND FOLLOWS STANDARD
                                *   FLEX NOTATION.
                                *
                                *   THE NUMBER OF PRINT POSITIONS IS DEFINED BY
                                *   THE FOLLOWING EQU STATEMENT WHICH MAY BE CHANGED
                                *   FOR DIFFERENT SIZED OUTPUT DEVICES.
                                *
0000C                   LINE    EQU     12      NUMBER OF HEX CHARACTERS PER LINE
                                *
                                *   THIS PROGRAM WAS WRITTEN ON A SWTP 6800
                                *   SYSTEM USING FLEX 1.0
                                *
                                *   FREEMAN L. MOORE
                                *   CENTRAL MICHIGAN UNIVERSITY
                                *
                        RPTERR  EQU     $713C   REPORT DISK ERROR
                        DOS     EQU     $7103   RETURN INTO FLEX
                        PUTCHR  EQU     $7112   OUTPUT 1 CHAR
                        OUTHEX  EQU     $7139   OUTPUT 2-DIGIT HEX
                        FMS     EQU     $7D06   FILE MANAGEMENT SYSTEM ENTRY
                        GETFIL  EQU     $7D22   PROCESS FILE SPEC
                        FCB     EQU     $7740   USE SYSTEM FCB
                                *
7600                    START   ORG     $7600   USE COMMAND SPACE
7600 20 03              NOPEN   BRA     ST1     BRANCH TO START OF PROGRAM
7602 03                         FCB     3       VERSION OF PROGRAM
7603 BD 711C                    JSR     PCRLF   START ON NEXT LINE
7606 CE 7706                    LDX     #ERR1   PRINT ERROR IN FILE
7609 BD 7103                    JSR     DOS     OPENING PROCESS
760C 7E                         JMP             THEN STOP.
                                *
                                *   START OF PROGRAM HERE
                                *
760D 86 00              ST1     LDA     A#FCB   PROCESS FILE SPEC
760F BD 7740                    JSR     GETFIL
7612 CE 7727                    LDX     #FCB    OPEN FOR READ
7615 C6 01                      LDA     B#1     IN FILL FCB
7617 A7 00                      STA     A0,X
7619 BD 7D06                    JSR     FMS
761C 26 F2                      BNE     NOPEN
761E 86 FF                      LDA     A#$FF
```

```
00055  0  7623  A7  3B          LOOP1   STA A   59,X        CHANGE TO BINARY FILE
00056                          *
00057                          *  START OF PROCESSING SECTOR
00058                          *
00059  0  7625  FE  7786          LDX     FCB+64      GET CURRENT SECTOR INFO
00060  1  7628  FF  76F0          STX     SECTOR
00061  0  762B  CE  7740          LDX     #FCB
00062  0  762E  BD  7806          JSR     FMS         GET ONE BYTE
00063  0  7631  26  0E            BNE     ERROR
00064  1  7633  B7  76F5          STA A   HEX         SAVE CHARACTER FOR LATER
00065  0  7636  BD  711E          JSR     PCRLF       OUTPUT CR/LF
00066  1  7639  CE  76F7          LDX     #MESS1      *
00067  1  763C  BD  76C9          JSR     PSTRNG      *  OUTPUT DISK ADDRESS
00068  1  763F  CE  76F0          LDX     #SECTOR     *  TRACK
00069  0  7642  BD  7139          JSR     OUTHEX
00070  0  7645  86  00            LDA A   #'          OUTPUT SPACE
00071  0  7647  BD  7112          JSR     PUTCHR      *
00072  1  764A  CE  76F1          LDX     #SECTOR+1   * AND SECTOR
00073  0  764D  BD  7139          JSR     OUTHEX      *
00074  0  7650  BD  711E          JSR     PCRLF
00075  0  7653  86  7C            LDA A   #124        SET FOR NUMBER OF BYTES / SECTOR
00076  1  7655  B7  76F6          STA A   COUNT
00077  1  7658  BD  76E4          JSR     RESET       RESET COUNTERS
00078  0  765B  20  0B            BRA     LOOP3
00079                          *
00080                          *  MAIN LOOP BODY STARTS HERE
00081                          *
00082  0  765D  CE  7740  LOOP2   LDX     #FCB
00083  0  7660  BD  7806          JSR     FMS         GET ONE BYTE
00084  0  7663  26  3C            BNE     ERROR
00085  1  7665  B7  76F5          STA A   HEX         SAVE CHAR
00086  1  7668  CE  76F5  LOOP3   LDX     #HEX        ADDRESS OF HEX VALUE
00087  0  766B  BD  7139          JSR     OUTHEX      OUTPUT IN HEX
00088  0  766E  86  00            LDA A   #'            THEN OUTPUT SPACE
00089  0  7670  BD  7112          JSR     PUTCHR
00090  1  7673  B6  76F5          LDA A   HEX         GET CHAR AGAIN
00091  0  7676  BD  45            BSR     VALID       TEST IF ASCII CHAR
00092  1  7678  FE  76F3          LDX     CHRPTR      THEN SAVE IN BUFFER
00093  0  767B  A7  00            STA A   0,X
00094  0  767D  08                INX     ADVANCE     TO NEXT POSITION IN BUFFER
00095  1  767E  FF  76F3          STX     CHRPTR      SAVE NEW POINTER
00096  1  7681  7A  76F2          DEC     LCTR        DECREASE LINE COUNT
00097  0  7684  26  04            BNE     LOOP4       SKIP IF NOT E-O-L
00098                          *
00099                          *  END OF LINE PROCESSING
00100                          *
00101  0  7686  8D  4E            BSR     PBUFF       PRINT BUFFER
00102  0  7688  8D  5A            BSR     RESET       RESET COUNTERS
00103  1  768A  7A  76F6  LOOP4   DEC     COUNT       DECREASE SECTOR COUNT
00104  0  768D  26  CE            BNE     LOOP2       BRANCH IF NOT END OF SECTOR
00105                          *
00106                          *  END OF SECTOR PROCESSING
00107                          *
00108  1  768F  CE  771E  LOOP4A  LDX     #SPAC3      OUTPUT THREE BLANKS
00109  0  7692  8D  35            BSR     PSTRNG
00110  1  7694  7A  76F2          DEC     LCTR        FOR EACH HEX CHAR BYPASSED
00111  0  7697  26  F6            BNE     LOOP4A
00112  1  7699  FE  76F3          LDX     CHRPTR      RESTORE XR
00113  0  769C  8D  38            BSR     PBUFF       PRINT BUFFER
00114  1  769E  7E  7625          JMP     LOOP1
00115                          *
00116                          *     PROCESS DISK ERRORS FROM FMS
00117                          *
00118  0  76A1  B6  7741  ERROR   LDA A   FCB+1       GET ERROR CODE
00119  0  76A4  81  08            CMP A   #8          TEST FOR EOF
00120  0  76A6  26  0D            BNE     OVER        ERROR IF NOT EOF
00121  0  76A8  CE  7740  CLOSE   LDX     #FCB        PREPARE TO CLOSE
00122  0  76AB  86  04            LDA A   #4          THE FILE
00123  0  76AD  A7  00            STA A   0,X         4 IS FILE CODE
00124  0  76AF  BD  7806          JSR     FMS
00125  0  76B2  7E  7103          JMP     DOS         RETURN TO FLEX
00126                          *
00127                          *
00128  0  76B5  CE  7740  OVER    LDX     #FCB
00129  0  76B8  BD  713C          JSR     RPTERR      OUTPUT ERROR MESSAGE
00130  0  76BB  20  EB            BRA     CLOSE       THEN CLOSE THE FILE
00131                          *
00132                          *  SUBROUTINE TO RETURN IF VALID ASCII CHAR
00133                          *  ELSE CHANGE TO PERIOD (.)
00134                          *
00135  0  76BD  81  20  VALID     CMP A   #$20        COMPARE TO SPACE
00136  0  76BF  2B  05            BMI     VAL10       BR IF LESS
00137  0  76C1  81  7F            CMP A   #$7F        UPPER BOUND
00138  0  76C3  2E  01            BGT     VAL10       BR IF GREATER
00139  0  76C5  39                RTS     ELSE        RETURN OK
00140  0  76C6  86  2E  VAL10     LDA A   #'.         ELSE GET PERIOD
00141  0  76C8  39                RTS     TO          RETURN
00142                          *
00143                          *  INTERNAL SUBROUTINE TO ACT LIKE 'PSTRNG'
00144                          *  BUT WITHOUT INITIAL CR/LF
00145                          *
00146  0  76C9  A6  00  PSTRNG   LDA A   0,X         GET CHAR
00147  0  76CB  81  04            CMP A   #4          TEST IF EOT
00148  0  76CD  27  06            BEQ     PST20       DONE IF YES
00149  0  76CF  BD  7112          JSR     PUTCHR      ELSE OUTPUT CHARACTER
00150  0  76D2  08                INX     THEN        ADVANCE TO NEXT CHAR
00151  0  76D3  20  F4            BRA     PSTRNG      AND REPEAT PROCESS
00152  0  76D5  39                PST20 RTS
00153                          *
00154                          *  INTERNAL SUBPROGRAM TO PRINT CHARACTER BUFFER
00155                          *
00156  0  76D6  86  04  PBUFF     LDA A   #4
00157  0  76D8  A7  00            STA A   0,X         INSERT EOT CHAR
00158  1  76DA  CE  7726          LDX     #CHAR
00159  1  76DD  BD  76C9          JSR     PSTRNG      THEN OUTPUT BUFFER
00160  0  76E0  BD  711E          JSR     PCRLF       THEN CR / LF
00161  0  76E3  39                RTS
00162                          *
00163                          *  INTERNAL SUBPROGRAM TO RESET LINE COUNTERS
00164                          *
00165  0  76E4  86  0C  RESET     LDA A   #LINE       NUMBER OF HEX PER LINE
00166  1  76E6  B7  76F2          STA A   LCTR
00167  1  76E9  CE  7725          LDX     #CHAR+3     ADDRESS OF OUTPUT BUFFER
00168  1  76EC  FF  76F3          STX     CHRPTR
00169  0  76EF  39                RTS
00170                          *
00171                          *     DATA AND MESSAGES USED
00172                          *
00173  0  76F0  0002  SECTOR    RMB     2
00174  0  76F2  0001  LCTR      RMB     1
00175  0  76F3  0002  CHRPTR    RMB     2
00176  0  76F5  0001  HEX       RMB     1
00177  0  76F6  0001  COUNT     RMB     1
00178  0  76F7  44    MESS1     FCC     'DISK ADDRESS IS '
       0  76F8  49
       0  76F9  53
       0  76FA  4B
       0  76FB  20
       0  76FC  41
       0  76FD  44
       0  76FE  44
       0  76FF  52
       0  7700  45
       0  7701  53
       0  7702  53
       0  7703  20
       0  7704  49
       0  7705  53
       0  7706  20
       0  7707  04              FCB     4
00179  0  7708  45    ERR1      FCC     'ERROR IN FILE OPENING'
00180  0  7709  52
       0  770A  52
       0  770B  4F
       0  770C  52
       0  770D  20
       0  770E  49
       0  770F  4E
       0  7710  20
       0  7711  46
       0  7712  49
       0  7713  4C
       0  7714  45
       0  7715  20
       0  7716  4F
       0  7717  50
       0  7718  45
       0  7719  4E
       0  771A  49
       0  771B  4E
       0  771C  47
00181  0  771D  04              FCB     4
00182  0  771E  20    SPAC3     FCC     '   '
       0  771F  20
       0  7720  20
00183  0  7721  04              FCB     4
00184  0  7722  20    CHAR      FCC     '   '
       0  7723  20
       0  7724  20
       0  7725  20
00185  0  7726  000C            RMB     LINE        NOTE VARIABLE SIZE
00186  0  7732  04              FCB     4
00187                          END     START

TOTAL ERRORS - NONE
STARTING ADDRESS 7600
PROGRAM REQUIRES 0120 BYTES OF STORAGE
```



Listing of Object File

```
7600  26 0D 03 8D 71 1E CE 77 08 8D 7B C9 7E 71 03 CE *
7610  77 40 8D 71 27 CE 77 40 86 01 A7 00 BD 78 06 26 *
7620  E2 86 FF A7 3B FE 77 80 FF 76 F0 CE 77 40 BD 78 *
7630  06 26 6E B7 76 F5 8D 71 1E CE 76 F7 BD 76 C9 CE *
7640  76 F0 BD 71 39 86 00 8D 71 12 CE 76 F1 BD 71 39 *
7650  8D 71 1E 86 7C B7 76 F6 8D 76 E4 20 08 CE 77 40 *
7660  8D 78 06 26 3C B7 76 F5 CE 76 F5 BD 71 39 86 00 *
7670  8D 71 12 B6 76 F5 8D 45 FE 76 F3 A7 00 08 FF 76 *
7680  F3 7A 76 F2 26 04 8D 4E 8D 5A 7A 76 F6 26 CE CE *
7690  77 1E 8D 35 7A 76 F2 26 F6 FE 76 F3 8D 38 7E 76 *
76A0  25 B6 77 41 81 08 26 0D CE 77 40 86 04 A7 00 BD *
76B0  78 06 7E 71 03 CE 77 40 BD 71 3C 20 EB 81 20 2B *
76C0  05 81 7F 2E 01 39 86 2E 39 A6 00 81 04 27 06 BD *
76D0  71 12 08 20 F4 39 86 04 A7 00 CE 77 22 BD 76 C9 *
76E0  BD 71 1E 39 86 0C B7 76 F2 CE 77 25 FF 76 F3 39 *
76F0  44 49 53 4B 20 41 44 44 52 45 53 53 20 49 53 20 *
7700  44 45 52 52 52 4F 52 20 49 4E 20 46 49 4C 45 20 4F *
7717  50 45 4E 49 4E 47 04 20 20 20 04 20 20 20 20 *
7732  04 *
```

*** Symbol Table ***

```
CHAR    1  7722
CHRPTR  1  76F3
CLOSE   1  76A8
COUNT   1  76F6
DOS     0  7103
ERR1    1  7708
ERROR   1  76A1
FCB     0  7740
FMS     0  7806
GETFIL  0  7127
HEX     1  76F5
LCTR    1  76F2
LINE    1  000C
LOOP1   1  7625
LOOP2   1  765D
LOOP3   1  7668
LOOP4   1  768A
LOOP4A  1  768F
MESS1   1  76F7
NOPEN   1  7603
OUTHEX  0  7139
OVER    1  76B5
PBUFF   1  76D6
PCRLF   1  711E
PST20   1  76D5
PSTRNG  1  76C9
PUTCHR  0  7112
RESET   1  76E4
RPTERR  0  713C
SECTOR  1  76F0
SPAC3   1  771E
ST1     1  760F
START   1  7600
VAL10   1  76C6
VALID   1  76BD
```

PRELIM PROGRAMMING SPECS VDM-2/GRAPHIC DISPLAY

BY LEE FELSENSTEIN
Golemics, Inc.
1407 Addison St.
Berkeley, CA 94702

Dear Dr. Dobb's:

I am trying to find a home for an orphan design. The VDM-2 currently exists as a prototype S-100 printed circuit built partly to specifications set by Processor Technology, which so recently made a meteoric attempt to achieve greatness by liquidating. Being but a simple engineer, I stolidly completed the design (with a few small changes) and displayed the result at SigGraph recently. Now it remains to find a manufacturer, especially one who will agree to make a version available as an S-100 Sol retrofit.

From my own experience, the best designs and systems are the result of playing around, rather than "serious work." I have almost completed my own hardware-related playing around—now it's time for the software fun. Accordingly, I am submitting for publication the complete set of software specifications for the current version of the VDM-2.

Note that the final version remains to be laid out. I am committed to making an S-100 version available somehow, but the design can be adapted for almost any bus.

Readers: Please *write me your comments!*

The VDM-2 would have existed six months ago it I had had the good sense to stop waiting for the front office turkeys to make up their minds. As it was, I finally told them what I would and would not design. Now they've lost their front office, but maybe we don't need them anyway. — *Lee Felsenstein*

Introduction

The VDM-2 was designed as a sophisticated 80 X 24 text display for use in Sol and other S-100 computers. Of prime importance in setting the specifications has been the desire to allow for the fullest use of the graphics-related capabilities inherent in the memory-mapped type of display. In addition, provision has been made in the hardware for use of the VDM-2 in an extended video environment.

Graphics-related capabilities include the alternate writeable character font, row-column screen addressing, contiguous accessible video in both dimensions across the screen and "glitch-free" CPU access.

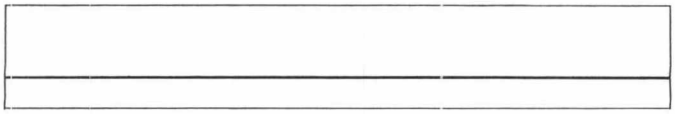
Alphanumeric features include extended attributes such as underline, half intensity, blinking and font select, ROM resident character set using EPROM-compatible character generator, selectable character or cursor blink, selectable text line modulus and smooth scrolling in either direction.

Video-related features include video lock capability among several units, memory disable for efficient multi-unit operation, and gen-lock capability to external video.

The VDM-2 is intended to be used in an interactive system mode and provides for the possibility of quote windowing, mixture with live and animated video, and joystick screen control. Programming for the VDM-2 should be done with the enjoyment of the user in mind.

Screen Format

The VDM-2 occupies an address space of 4096 bytes, organized as 32 rows of 128 columns each. Since the screen displays only 1920 characters at most, columns 81 through 128 (50h through 7Fh) are not displayed and must not be written by the CPU. Similarly, rows 25 through 32 (18h through 1Fh) must not be written.



This is because the board contains memory for only 2048 bytes in its base page, and the mapping necessary to compact the memory space results in non-monotonic relationships between address and physical memory location. In simpler language, anything written into forbidden memory areas will turn up at another apparent location within the visible area.

There is, however, an area of memory which may be utilized by the CPU without any interference with the display. This area exists as a series of eight groups of sixteen contiguous addresses each, starting at C30h and recurring at intervals of 128 bytes (the next group starts at CB0h and runs through CBFh—the lowest three hex digits of the memory address are specified because the highest digit is the base address of the unit). These locations may be used for temporary storage of parameters specific to the operation of the display. Together with the memory disable feature, this allows the use of re-entrant code for the display driver in many applications involving multiple displays.

In such an arrangement, all of the VDM-2's may be addressed to the same 4K base address. Bit 0 of the control output register will prevent the memory decoder of the device from responding when set = 1. I/O references are unaffected by this "DISABLE" bit.

All memory and I/O references to the VDM-2 cause no interruption in the screen display.

Screen Memory Definition

The screen memory stores one 12-bit word for each character location on the screen. These words are accessed by the CPU as determined by the status of the PAGE bit (bit 2) of the control register. A bit value of 0 gives the CPU access to the low-order eight bits of each word. In this case, the low-order seven bits of the CPU data word access the character number—equivalent to the ASCII value if the character of an ASCII character font is used. The high order bit of the CPU word controls the cursor for that character—video will be inverted in that character cell if this bit is set = 1.

With PAGE set = 1 in the control register, the CPU accesses the high-order four bits of the character word. These bits are accessed as the four least significant bits of the CPU data word and their functions are:

bit 0 — underline: the cursor is inverted on scans nine and ten of the character
bit 1 — half intensity:
bit 2 — blink: blinks video at 512 msec period if the BLINK bit of the control register is set = 0; blinks the cursor if a cursor is set at that character location and the BLINK bit of the control register is set = 1
bit 3 — writeable font.

These bits are active when set = 1 in the CPU data byte, and affect only the character location where written. Both pages may be written and read by the CPU, and the PAGE bit of the control register remains at its previous value until changed by a CPU reference or reset to 0 by a bus initialization.

Writeable Font

In addition to the standard 128 ASCII character dot patterns stored in the ROM font memeory, the VDM-2 stores another 128 character dot patterns in a 2048 byte random-access memory. These may be called up on a character-by-character basis by the high order bit of the character data word as described above.

While the raster of the VDM-2 is interlaced, the same video is fed to both fields. This results in a duplication of the video on each pair of interlaced scans and a finer apparent display in the vertical dimension. The video uses 240 scans to display either 24 or 20 character lines.

In the 20 line mode, the character font consists of a dot matrix eight dots wide by twelve scans high. In the 24 line mode the lower two scans of the matrix are truncated to produce a matrix ten scans high. The display mode is selected by the status of bit 4 (24) of the control register. 20 line mode will be selected when this bit is set = 0 or after initialization.

The writeable font memory is accessed by the CPU when the FONT bit of the control register (bit 1) is set = 1. In this mode, the low order seven bits of the address specify the character number to be accessed. The eighth bit of address is ignored, and the ninth through twelfth bits of the address specifies the scan number accessed. The scans are numbered top to bottom, zero through eleven (Oh through Ah). The CPU data byte will be accessed to or from the scan specified. The least significant bit of the data byte will correspond to the right hand dot of the matrix, and the most significant bit to the left hand dot. A bit set = 1 will turn on video in that dot location.

Some persons familiar with graphics programming may quarrel with this assignment of bits horizontally, pointing out that screen co-ordination has historically had its origin point at the upper left hand corner. This scheme has been chosen to make life easier for the inexperienced programmer trying to convert a visual pattern into memory contents. With least significant bit to the right, the pattern becomes a graphic representation of two hexadecimal digits commonly used to represent memory contents. The user need only memorize the hex bit patterns and enter the desired two hex bytes to the memory.

The deletion of the eighth memory address bit puts the scan number fully in the high byte of the memory address, which is manipulated in the H register of most 8 bit microprocessors. A font created and stored in the writeable font memory may be transferred to a 2716 type EPROM through a bus-resident programmer, providing the address is "closed up" by shifting the four scan bits right one place.

Sixteen scans per character pattern may be stored and read by the CPU, but only the top twelve will be visible.

Scrolling

"Scrolling" here refers to the mapping of the text line in memory to the character line on the screen. Mapping of a given character line to a given scan line is considered below under "crawling."

Upon power-up initialization, the memory text line having line address 0 appears as the first character line at the top of the screen. An output to the screen port with data bit 4

set = 0 will increment the memory text line number displayed on the top character line of the screen. The memory text line formerly at the top of the screen will appear as the bottom line on the screen. This occurs regardless of the display mode (20 or 24 line screen).

It is important to note that in the 20 line mode, the highest four memory text lines (20 through 23) do not participate in the scrolling, although they are accessible by the CPU. These lines are used by the "subtext" command, which is described below.

Scrolling down is accomplished by scrolling up N-1 times, where N is the number of character lines displayed on the screen. Since the scrolling hardware is read internally at the leading edge of the vertical display signal VDISP, scrolling should be held off until immediately after this event. VDISP is made available to the CPU on bit 1 of the status port. The CPU should store the last value of this bit and test the new value to detect a change from 0 to 1.

The CPU may reset the scrolling hardware by outputting to the screen port with bit 5 set = 0. This will override any data on bit 4 and will cause the top line of the display to be memory text line 0.

Crawling

The VDM-2 includes the hardware necessary for the movement of the display vertically in increments of one scan. When synchronized with the scrolling, this feature allows for smooth text movement up or down the screen as if the screen were a window gliding over an unbroken column of text.

In such crawling it is ideal if the top displayed line is gradually eclipsed by the top of the screen while the new bottom line emerges gradually at the same time. Since in the 24 line mode of the VDM-2 the partially obscured top and bottom lines are in fact the same text line in memory, this ideal condition is not realized. Instead, a one-line "preblank" curtain descends over the top line at such strategic times.

Crawling is controlled by bits 0 through 3 of the screen port. Bit 0 sets the direction (0 = up, 1 = down), and bits 1 and 2 select one of four crawl rates. Three of these are generated in the hardware as submultiples of the 60 Hz vertical rate. One (both bits = 1) provides for an externally generated crawl clock, which may be, for example, an oscillator whose frequency is controlled by a joystick. Bit 3 is the "go" bit which initiates a crawl sequence when set = 0 during a screen port output.

Upon issuance of a "crawl up" sequence, the display will wait until the onset of VDISP and will then appear to "jump down" one character line. The old top line will still appear at the top of the display, but it will be displaced down by one character line. The display will gradually crawl back up to its original state, advancing one scan for each crawl clock. When the starting point is reached, bit 0 of the status port (CRAWL DONE) will set = 1 (it sets = 0 immediately upon issuance of a crawl sequence command).

It should be obvious that if, immediately following the issuance of a "crawl up" command and before the onset of VDISP, the new bottom line is written into the old top line and the display is scrolled up one line, the result will be the disappearance of the top line and a slow scroll up, with the old top line appearing at the bottom of the screen with its new contents.

When a "crawl down'. sequence is initiated, the subsequent onset of VDISP will cause the display to move down one scan, with the bottom scan of the bottom line obscured. This downward movement will continue, one scan for each crawl clock, until the bottom line has been completely obscured. The CRAWL DONE bit will now return t-o = 1. At this point the new top line should be written into the old bottom line and the screen should be scrolled down one line. The newly written line will not become visible until after a new "crawl down" sequence is initiated.

If the newly-written top line is that last of a sequence and it is desired that the crawling stop, issue the crawl down command and follow it immediately with an output to the screen port having the direction bit = 0 (up) and the GO bit = 1. The new line will appear at the top of the screen and on the next crawl clock will move up one scan. CRAWL DONE will then set = 1.

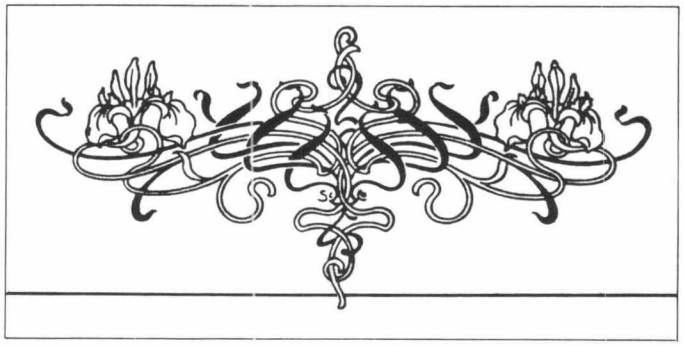
When the external crawl clock is selected it is useful to have an externally provided "go request" and "direction request" set of signals so that text movement may be controlled without use of the standard input channel such as the keyboard. Bits 4 and 5 of the status port are connected to the external device connector and are read by the VDM-2 when the status port is read. These bits are advisory to the CPU and have no function in the VDM-2 hardware. They will show a default "1" value when no external device is connected.

Subtext

In the section on scrolling it was mentioned that four memory text lines remain unseen when the VDM-2 is in the 20 line mode. These text lines may still be accessed by the CPU, and constitute a "subtext" area for information of secondary utility. Upon command from the CPU these lines can be made to appear at the bottom of the screen obscuring one or more of the text lines normally displayed.

Bit 5 of the control register (SUBTEXT 0) will cause the display of one or more of these lines when set = 1. The number of subtext lines to be displayed is determined by the binary value of bits 6 and 7 of the control register (SUBTEXT 1 and 2). Taking bit 7 as the most significant bit, values 0, 1, 2, and 3 represent 1, 2, 3, and 4 displayed subtext lines. These lines always end at the bottom of the screen and the topmost line is always memory text line 20. The subtext lines do not respond to crawl or scroll commands.

Subtext lines are intended for uses such as prompting, system status information display, menus, "jack-in-the-box" quick comebacks, etc. Use of the subtext should be avoided in the 24 line mode, since the subtext lines will be duplicates of lines visible elsewhere on the screen.



Interrupts

Interrupts have been left unimplemented in the current prototype S-100 version of the VDM-2. Since interrupts are a system consideration and since the VDM-2 is currently without a system, many configurations are possible.

As a minimum, the CRAWL DONE bit could cause an interrupt by its onset when interrupts are enabled. This would allow smooth scrolling with a minimum of CPU attention.

Bit 7 of the screen port has been tentatvely designated as INTERRUPT ENABLE. Its exact mode of functioning awaits the final design of the device.

BIT SUMMARY -- OUTPUT PORTS

Control port

bit no.	Name	Function
0	DISABLE	- Prevents memory decoder from responding when set = 1
1	FONT	- Causes font memory (writeable) to respond to CPU memory references when set = 1
2	PAGE	- Causes high order four bits of screen memory character word to respond to CPU memory references when set = 1
3	BLINK	- Determines whether cursor or video blinks on characters having page 1 bit 2 set = 1. 0 causes video blink, 1 causes cursor blink.
4	24	- Determines number of character lines per screen. 0 causes 20 lines of 12 scan each. 1 causes 24 line display of 10 scans each.
5	SUBTEXT 0	- Controls display of subtext lines at bottom of screen. 1 = subtext displayed.
6	SUBTEXT 1	- Least significant bit pair
7	SUBTEXT 2	- Most significant bit of pair Bit pair value determines number of subtext lines to be displayed. Number of lines is one plus binary value of bit pair.

Screen port

bit no.	Name	Function
0	DIRECTION	- Sets direction of crawl. 0 = up.
1 2	Speed 0 SPEED 1	- As a binary pair (bit 1 least significant) sets one of four crawl speeds: value speed 0 60 scans/ sec. 1 30 scans/sec. 2 15 scans/ sec. 3 externally clocked
3	GO	- Intitiates crawl operation when set = 0
4	SCROLL	- Increments the number of the memory text line displayed at the top of the screen. 0 = increment
5	RESET	- When set = 0 causes memory text line 0 to be displayed at top of screen. Overrides effect of bit 4.
7	INTERRUPT ENABLE	- reserved for use in interrupt version

BIT SUMMARY -- INPUT PORT

Status port

bit no.	name	function
0	CRAWL DONE	Sets = 0 when crawl is in progress.
1	VDISP	Sets = 1 during vertical display time (240 scans)
2	CRAWL CLOCK	Sets = 1 during crawl clock signal. Crawl advances one scan at onset of VDISP following leading edge of crawl clock.
3	unused	
4	EXT. CRAWL RQST	Sets = 0 when external crawl-control hardware is requesting text movement
5	EXT CRAWL DIRECTION	sets = 0 when external crawl control hardawre requests crawl down.

(note- bits 4,5 are set = 1 when no external device is connected.)

Continued from pg. 35

Copyrights

As usual, I assert the free-diffusion-clause copyright defined in *DDJ* #32 and earlier articles. I recommend this to algorithmaniacs of my ilk, since it encourages users to let you know of unexpected applications *they* have found for your creations.

Hal Gordon is 60 years old, has a Ph.D (Biology, Harvard, 1947), and has been at UC – Berkeley since 1947. He is a low-level programmer in more ways than one, having learned how to use a CPU directly on an early KIM-1 and only recently upgraded to a SYM-1. His (still unrealized) dream is to equip these boards with sensors and effectors so that they can control real-time biological experiments. He sympathizes with the anti-AI views of Joseph Weizenbaum on the limits of human-language-oriented programs, believing that it's more instructive to interact with books and the right kind of human mind.

(listing of subroutine MIXSIM, without addresses since it is fully relocatable)

```
A6 DF   MIXSIM LDX MEMEX   (load prev. G-index into X)
CA             DEX         (decrement X for next G)
30 0D          BMI RESET   (if = $FF, go reset X to 02)
B5 E1          LDA RND+1,X (load G+1 seed into Acc.)
D0 0B          BNE SIMRND  (if ≠ 0, run normal G logic)
B5 E2          LDA RND+2,X (load G+2 seed into Acc.)
F0 07          BEQ SIMRND  (if = 0, run normal G)
B5 E0          LDA RND,X   (load G seed into Acc.)
18             CLC         (clear carry for branch)
90 10          BCC DEJUM   (output pseudo-G number)
A2 02   RESET  LDX #$02    (reset X to 02)

B5 E0   SIMRND LDA RND,X   (load G seed into Acc.)
0A             ASL A       (X 2)
0A             ASL A       (X 4)
38             SEC         (set carry to add 1)
75 E0          ADC RND,X   (X 5, + 1)
95 E0          STA RND,X   (store new seed)

18             CLC         (clear carry for addition)
75 E3          ADC RND+3,X (add addend for G variant)

10 02          BPL STOREX  (bypass DEJUM if bit 7 = 0)
49 7F   DEJUM  EOR #$7F    (complement lowest 7 bits)

86 DF   STOREX STX MEMEX   (store G-index value)
60             RTS         (exit subroutine)
```

Zero-page locations used (all must be initialized, cf. text for restrictions):

```
DF   MEMEX  (must be 02 or 01 or 00)

E0   RND    (seed of G0)
E1   RND+1  (seed of G1)
E2   RND+2  (seed of G2)

E3   RND+3  (addend of G0, must not be = 00)
E4   RND+4  (addend of G1, ≠ RND+3 and ≠ RND+5)
E5   RND+5  (addend of G2, ≠ RND+3 and ≠ RND+4)
```

An Interactive Heath H-8 Disassembler

BY RONALD T. BOROCHOFF, Ph.D.
3822 Murworth Drive
Houston, Texas 77025

The following program is a modification of the 8080 disassembler which originally appeared in the March, 1977 issue of *DDJ*, Page 25. The program has been adapted to fit a Heath H-8 using Benton Harbor Extended Basic. Capabilities have also been added for listing the numerical opcodes and for an optional continuous disassembly mode.

It is unfortunate that the original contributor wished to remain anonymous, since the program is a superb example of how a well-written program should be devised. The original deserves attention even if you aren't interested in a disassembler.

In converting the program to my H-8, I have maintained all of the original comments and line numbers and attempted to maintain the original spirit of the program; that is, to keep the code and intent of the program as transparent as possible. However, B-H Basic does not allow variables with more than one alphabetic character; the original variable names, therefore, had to be replaced with something less inspired but more workable.

Heath's split level octal format has been substituted for the original hex. If you prefer hex, it is easy enough to return to that format with insertions from the original listing. In fact, I maintain a hex version just to enable communication with friends of that persuasion.

I had added two functions which I like and which were not included in the original program: First, the numerical opcodes are now printed (in octal) to accompany the mnemonics; and second, an option has been added to enable continuous disassembly down to a predetermined address; the original allowed only line by line disassembly by tapping the space bar.

After loading the disassembler and program of interest and typing "RUN", you will receive a prompt, ">". You may then type in the split octal address of where disassembly is to start. The program will check the address for validity and return the mnemonic and numerical opcode. It is not necessary to type "RETURN". To continue disassembly, tap the space bar line by line. Each subsequent address, nmemonic and opcode will be printed out. If you wish to go to another address, then type in that address instead of hitting the space bar; continue from there.

An attractive feature of this program is the ability to branch with the program when such a branch command is encountered. Arriving at any program statement containing a jump or call, you have the option to continue to the next address in sequence or to branch disassembly to the address of the command. If you choose to follow the branch, type "J", the program will complete the word "JUMP" and print the beginning address of the jump or call. The mnemonic and numeric code will be printed and disassembly may continue from this point. You may return to the branch point any time by typing in that address. This "JUMP" capability is useful in that it allows you to follow the actual run time computer path, skirting around such things as data storage or subroutine areas which may confuse the disassembler.

At any time, if you wish to continuously disassemble a program or a portion of a program down to some address, simply type "C". The machine will complete the command "CONTINUE TO >"; you then input the address where you wish disassembly to stop. The machine will then print out each successive address, mnemonic and numerical opcode until it reaches the stop address. Once at the stop address, you may continue line by line as before. The "CONTINUE TO" capability is handy for disassembling long subroutines or programs that you don't expect to be troublesome. A default stop address, at Line 155, is reverted to at the end of a "CONTINUE TO" mode. This is the maximum address that any program is expected to have. For my computer, this is decimal 32,765; which happens to correspond with

Listing


```
5 REM ********* INTERACTIVE 8080 DISASSEMBLER *********
7 REM
10 REM MODIFIED FOR HEATH H-8 AND ASR-33 BY R. T. EDROCHOFF, 3/11/79
20 REM ORIGINAL PROGRAM FROM "DR. DOBB'S JOURNAL" 3/77, ANONYMOUSLY.
25 REM ADDED OPCODE PRINTING AND CONTINUOUS RUN MODE, 3/30/79
30 REM
40 REM REMOVE 'REM' STATEMENTS TO SAVE MEMORY, BUT BEWARE OF
50 REM BRANCHES TO SOME OF THESE STATEMENTS. IN SUCH A CASE,
60 REM LEAVE 'REM' OUT REMOVE TEXT.
100 GOSUB 9000:REM INITALIZE ARRAYS, CONSTANTS
110 PRINT "    8 0 8 0      D I S A S S E M B L E R"
120 PRINT
140 J=F:REM JUMP FLAG
145 C1=F
150 P=0
155 M1=32765:REM MEMORY LIMIT; MAY BE CHANGED
200 REM *** MAIN LOOP ***
210 PRINT " >";
220 GOSUB 5600:REM GET A CHARACTER AND ECHO IT
222 IF C1=T THEN IF C$=" " OR C$=CHR$(13) GOTO 120:REM EXIT CONT. MODE
225 IF C$="C" THEN C1=T:GOTO 550
230 IF C$=" "GOTO 500
235 IF J=T THEN IF C$="J" GOTO 450
240 REM LOOK FOR AN ADDRESS (MUST START WITH 0-3)
245 W=0
250 IF C$>="0" AND C$<="3"GOTO 300
252 IF W=1 THEN IF C=127 GOTO 340
255 IF C$=CHR$(13) THEN PRINT :GOTO 210
260 IF P2=T THEN GOSUB 6200:REM ERASE CHAR.
270 GOSUB 5600:GOTO 250
300 T1=C-ASC("0"):N=1
310 GOSUB 5600:REM GET A CHAR.
320 IF C$>="0" AND C$<="7" THEN D=C-ASC("0"):GOTO 400
340 IF C=127 THEN PRINT CHR$(127);CHR$(13);CHR$(10);:GOTO 200
350 IF P2=T THEN GOSUB 6200:REM ERASE CHAR.
360 GOTO 310
400 T1=D+8*T1
405 N=N+1
410 IF N<3 GOTO 310
415 IF W=1 GOTO 425
420 U=T1:W=1:GOSUB 5600:GOTO 250
425 T1=U*256+T1
427 IF C1=T THEN M1=T1:PRINT :GOTO 520:REM T1 IS LIMIT OF CONT. RUN
430 P=T1
440 GOTO 1020
450 REM JUMP TO ADDRESS IN LAST CALL OR JUMP INSTR.
460 PRINT "JMP";PRINT " >";
470 P=X
480 GOTO 1000
500 REM SPACE MEANS - DO THE NEXT INSTRUCTION
510 PRINT CHR$(8);:REM GET RID OF THE SPACE
520 IF P<=M1 GOTO 1000
530 PRINT "ADDRESS GREATER THAN LIMIT";C1=F:M1=32765:REM REINITIATE
540 GOTO 200
550 PRINT "CONTINUE TO";:GOTO 200
1000 REM
1010 X=P:GOSUB 5450:REM PRINT ADDRESS
1020 PRINT ";  ";
1030 GOSUB 5000:D=X:REM FETCH OPCODE
```

the top address of my available RAM. This simply prevents me from inadvertently stepping out of RAM. This value may be changed to suit the user's own needs.

I should explain that the terminal I am presently using is a Teletype ASR-33. Therefore, all terminal I/O considerations have been based on this. I have thus deleted all the original commands dealing with an "intelligent" terminal. You may want to reinsert these if you have such a device. Also, when an address is input, the program performs a validity check to insure that the address fits the split octal format. Any character that does not fit is automatically deleted and the "cursor" is backspaced (ASCII 8). Since my teletype does not backspace, a "left arrow" is printed to denote that a deletion and backspace have been inserted. If you have a backspacing terminal, you will not need the arrow; therefore, delete the CHR$(95) from line 6210.

A short word about the commenting of the program. If all the "REMARK" statements are removed, the required program storage will be reduced to about half. However, be advised that there are several branches to these "REMARK" statements. If you are not certain about branches to a "RE–MARK" you are deleting, delete the command and retain the "REM" statement. This is simpler than trying to find and change all the involved "GOTO" and "GOSUB" statements.

I think you will find this disassembler is a handy tool that will help you disassemble all sorts of "obfuscated" code, even Benton Harbor Basic itself. And, if you enjoy writing short machine code programs from the front panel, as I do, this little disassembler is a good way to get a nice, neat listing for your program.

```
1040 G0SUB 2000:PRINT TAB(31);M$;:M$="":REM DEC0DE IT
1045 PRINT
1047 IF C1=T G0T0 520
1050 G0T0 200
2000 REM DEC0DE "0" AS AN INSTRUCTI0N AND PRINT ITS MNEM0NIC
2010 REM
2020 J=F
2030 REM GENERAL N0TE AB0UT "0N N ..." STATEMENTS:
2040 REM IF N IS 0, C0NTR0L DR0PS THR0UGH T0 THE
2050 REM STATEMENT F0LL0WING THE "0N" STATEMENT.
2055 REM LLMMMKKK   EXTRACT THE FIELDS SH0WN
2060 L=INT(0/B(6))        :REM BITS 6-7
2062 M=INT(0/B(3)) AND 7 :REM BITS 3-5
2064 K=0 AND 7            :REM BITS 0-2
2070 0N L G0T0 3000,3200,3500
2099 REM
2100 REM 00MMMKKK
2110 REM DISPATCH 0N K (BITS 0-2)
2120 0N K G0T0 2200,2400,2500,2600,2600,2700,2800
2129 REM
2130 REM 00MMM000
2140 IF M=0 THEN S$="N0P"
2150 IF M>0 THEN S$="- -":REM UNIMPLEMENTED INSTRUCTI0N
2160 G0SUB 6000:RETURN
2199 REM
2200 REM 00MMM001
2210 K=M/B(1):REM K IS N0W BITS 4-5 0F 0PC0DE
2220 IF 0 AND B(3)=T G0T0 2300
2230 REM 00KK0001
2240 S$="LXI ":G0SUB 6000
2250 S$=R2$(K):G0SUB 6000
2260 S$=", ":G0SUB 6000
2270 G0SUB 5400:REM FETCH NEXT TW0 BYTES
2280 RETURN
2300 REM 00KK1001
2310 S$="DAD ":G0SUB 6000
2320 S$=R2$(K):G0SUB 6000
2330 RETURN
2399 REM
2400 REM 00MMM010
2410 S$=02$(M):G0SUB 6000
2420 IF M<4 THEN RETURN :REM STAX 0R LDAX, 1 BYTE
2430 REM SHLD,LHLD,STA,LDA - FETCH NEXT 2 BYTES
2440 G0SUB 6100
2450 G0SUB 5400
2460 RETURN
2499 REM
2500 REM 00MMM011
2510 K=INT(M/B(1)):REM K IS N0W BITS 4-5 0F 0PC0DE
2520 M=M AND B(0):REM M IS N0W BIT 3 0F 0PC0DE
2530 IF M=0 THEN S$="INX "
2540 IF M=1 THEN S$="DCX "
2550 G0SUB 6000
2560 S$=R2$(K):G0SUB 6000
2570 RETURN
2599 REM
2600 REM 00MMM100  INR
2602 REM 00MMM101  DCR
2610 IF K=4 THEN S$="INR "
2620 IF K=5 THEN S$="DCR "
2630 G0SUB 6000
2640 S$=R1$(M):G0SUB 6000
2650 RETURN
2699 REM
2700 REM 00MMM110  MVI
2710 S$="MVI ":G0SUB 6000
2720 S$=R1$(M):G0SUB 6000
2730 S$=", ":G0SUB 6000
2740 G0SUB 5200
2750 RETURN
2799 REM
2800 REM 00MMM111
2810 S$=07$(M):G0SUB 6000
2820 RETURN
3000 REM 01MMMKKK   M0V INSTRUCTI0N
3010 IF M=6 AND K=6 G0T0 3150:REM M0V M,M IS REALLY HLT
3020 S$="M0V ":G0SUB 6000
3030 S$=R1$(M):G0SUB 6000:REM DESTINATI0N REGISTER
3040 S$=", ":G0SUB 6000
3050 S$=R1$(K):G0SUB 6000:REM S0URCE REGISTER
3060 RETURN
3199 REM
3200 REM 10MMMKKK   ACCUMULAT0R ARITHMETIC INSTR.
3210 S$=A1$(M):G0SUB 6000:REM M DETERMINES 0PERATI0N
3220 G0SUB 6100
3230 S$=R1$(K):G0SUB 6000:REM K DETERMINES 0PERAND
3240 RETURN
3499 REM
3500 REM 11MMMKKK
3510 0N K G0T0 3560,3600,3800,3620,4000,4100,4200
3520 REM 11MMM000  RETURN C0NDITI0NAL
3530 S$="R":G0SUB 6000
3540 S$=C8$(M):G0SUB 6000:REM M IS THE C0NDITI0N
3550 RETURN
3560 REM 11MMM001
3570 S$=C1$(M):G0SUB 6000
3580 RETURN
3599 REM
3600 REM 11MMM010  JUMP C0NDITI0NAL
3610 S$="J":G0T0 3650
3620 REM 11MMM100  CALL C0NDITI0NAL
3630 S$="C"
3650 G0SUB 6000
3660 S$=C8$(M):G0SUB 6000:REM M DETERMINES THE C0NDITI0N
3680 G0SUB 6100
3690 G0SUB 5400:REM FETCH THE ADDRESS
3695 J=T
3700 RETURN
3800 REM 11MMM011
3810 S$=C3$(M):G0SUB 6000:REM PRINT 0PC0DE
3820 IF M=0 G0T0 3850:REM JMP
3830 IF M=2 0R M=3 G0T0 3900:REM 0UT,IN
3840 RETURN
3850 G0SUB 6100
3860 G0SUB 5400:REM FETCH THE ADDRESS
3865 J=T
3870 RETURN
3900 G0SUB 6100
3910 G0SUB 5200:REM FETCH THE DEVICE C0DE
3920 RETURN
4000 REM 11MMM101
4010 S$=C5$(M):G0SUB 6000
4020 IF M<>1 THEN RETURN
4030 REM CALL INSTRUCTI0N
4040 G0SUB 6100
4050 G0SUB 5400:REM FETCH THE ADDRESS
4055 J=T
4060 RETURN
4100 REM 11MMM110
4110 REM IMMEDIATE ACCUMULAT0R ARITHMETIC INSTR.'S
4120 S$=C6$(M):G0SUB 6000:REM PRINT 0PC0DE
4130 S$=" ":G0SUB 6000
```

```
4140 GØSUB 5200:REM FETCH ØPERAND
4150 RETURN
4200 REM 11MMM111  RST
4210 S$="RST ":GØSUB 6000
4220 X=8*M:GØSUB 5100:REM PRINT ADDRESS CALLED
4230 J=T
4240 RETURN
4999 REM
5000 REM FETCH THE NEXT BYTE AND INCREMENT THE "PC"
5010 REM THE BYTE IS RETURNED IN X
5020 REM P IS THE "PC".
5030 X=PEEK(P):P=P+1:GØSUB 5050:RETURN
5035 REM LØAD NUMERICAL ØPCØDES INTØ M$
5050 X1=INT(X/64):N$=Q$(X1)+Q$(INT((X-X1*64)/8))+Q$(X AND 7)
5060 M$=M$+N$+CHR$(32):RETURN
5099 REM
5100 REM PRINT THE NUMBER X AS THREE ØCTAL DIGITS
5110 X1=INT(X/64):PRINT Q$(X1);Q$(INT((X-X1*64)/8));Q$(X AND 7);:RETURN
5199 REM
5200 REM FETCH THE NEXT BYTE AND PRINT IT IN ØCTAL
5210 REM THE NEW BYTE IS RETURNED IN X
5220 GØSUB 5000:GØSUB 5100:RETURN
5299 REM
5300 REM FETCH THE NEXT TWØ BYTES
5310 REM RETURN THE FIRST IN L1, THE SECØND IN H1,
5320 REM AND THE 16-BIT NUMBER (H1,L1) IN X
5330 GØSUB 5000:L1=X
5340 GØSUB 5000:H1=X
5350 X=L1+256*H1:RETURN
5399 REM
5400 REM FETCH THE NEXT TWØ BYTES, AS ABØVE,
5410 REM AND PRINT X AS A 6-DIGIT ØCTAL NUMBER
5420 GØSUB 5300:GØSUB 5450
5430 RETURN
5450 REM PRINT X AS A SIX DIGIT ØCTAL NUMBER
5460 T1=X:X=INT(T1/256):GØSUB 5100
5470 X=T1-256*X:GØSUB 5100
5480 X=T1:RETURN
5499 REM
5500 REM GET A SINGLE CHARACTER FRØM THE KEYBØARD,
5510 REM RETURN THE NUMBER IN C, THE CHARACTER IN C$.
5520 PAUSE
5523 C=PIN(250) AND 127
5525 IF C>=97AND C<=122THEN C=C-32
5530 C$=CHR$(C):RETURN
5600 REM GET A CHARACTER FRØM THE KEYBØARD,
5610 REM ECHØ IT ØN THE SCREEN IF IT'S A PRINTING CHARACTER;
5620 REM P2 IS TRUE IF THE CHARACTER WAS PRINTED.
5630 GØSUB 5500
5640 IF C>=32 AND C<=126 THEN P2=T:GØTØ 5650
5645 P2=F
5650 IF P2=T THEN PRINT C$;
5660 RETURN
5999 REM
6000 REM PRINT THE STRING IN S$
6010 REM UNLESS THE PRINT FLAG, P1, IS FALSE
6020 PRINT S$;
6030 RETURN
6100 REM PRINT A BLANK UNLESS P1 IS FALSE
6110 PRINT " ";
6120 RETURN
6200 REM DELETE LAST CHAR. FRØM THE SCREEN
6210 PRINT CHR$(8);CHR$(95);:REM PRINT BACKSPACE AND ←
6220 RETURN
8999 REM
9000 REM INITIALIZATIØN
9010 REM
9020 REM AN ARRAY ØF ØCTAL DIGITS
9030 DIM Q$(7)
9040 FØR I=0 TØ 7:READ Q$(I):NEXT I
9050 DATA "0","1","2","3","4","5","6","7"
9060 REM
9070 REM AN ARRAY ØF BITS (PØWERS ØF 2)
9080 DIM B(7)
9090 FØR I=0 TØ 7:READ B(I):NEXT I
9100 DATA 1,2,4,8,16,32,64,128
9110 REM
9120 REM THE NAMES ØF THE 8-BIT REGISTERS
9130 DIM R1$(7)
9140 FØR I=0 TØ 7:READ R1$(I):NEXT I
9150 DATA "B","C","D","E","H","L","M","A"
9160 REM
9170 REM THE NAMES ØF THE REGISTER PAIRS
9180 DIM R2$(3)
9190 FØR I=0 TØ 3:READ R2$(I):NEXT I
9200 DATA "B","D","H","SP"
9210 REM
9220 REM THE ACCUMULATØR ARITHMETIC INSTRUCTIØNS
9230 DIM A1$(7)
9240 FØR I=0 TØ 7:READ A1$(I):NEXT I
9250 DATA "ADD","ADC","SUB","SBB","ANA","XRA","ØRA","CMP"
9260 REM
9270 REM IMMEDIATE INSTRUCTIØNS (SERIES STARTING WITH C6H)
9280 DIM C6$(7)
9290 FØR I=0 TØ 7:READ C6$(I):NEXT I
9300 DATA "ADI","ACI","SUI","SBI","ANI","XRI","ØRI","CPI"
9310 REM
9320 REM CØNDITIØN CØDES FØR JUMP, CALL, RETURN
9330 DIM C8$(7)
9340 FØR I=0 TØ 7:READ C8$(I):NEXT I
9350 DATA "NZ","Z","NC","C","PØ","PE","P","M"
9360 REM
9370 REM THE SERIES ØF INSTRUCTIØNS STARTING WITH C1H
9380 REM AND SPACED 8 APART
9390 DIM C1$(7)
9400 FØR I=0 TØ 7:READ C1$(I):NEXT I
9410 DATA "PØP B","RET","PØP D","- -","PØP H","PCHL","PØP PSW","SPHL"
9420 REM
9430 REM THE SERIES STARTING WITH C3H
9440 DIM C3$(7)
9450 FØR I=0 TØ 7:READ C3$(I):NEXT I
9460 DATA "JMP","- -","ØUT","IN","XTHL","XCHG","DI","EI"
9470 REM
9480 REM THE C5H SERIES
9490 DIM C5$(7)
9500 FØR I=0 TØ 7:READ C5$(I):NEXT I
9510 DATA "PUSH B","CALL","PUSH D","- -","PUSH H","- -","PUSH PSW","- -
"
9520 REM
9530 REM THE SERIES STARTING WITH 07H
9540 DIM Ø7$(7)
9550 FØR I=0 TØ 7:READ Ø7$(I):NEXT I
9560 DATA "RLC","RRC","RAL","RAR","DAA","CMA","SYC","CMC"
9570 REM
9580 REM THE SERIES STARTING WITH 02H
9590 DIM Ø2$(7)
9600 FØR I=0 TØ 7:READ Ø2$(I):NEXT I
9610 DATA "STAX B","LDAX B","STAX D","LDAX D","SHLD","LHLD","STA","LDA"
9800 REM
9810 REM TRUE/FALSE CØNSTANTS
9820 T=1:F=0
9999 RETURN
*
```

Interfacing My Home Computer To A Large Scale System

BY GLENN STORY
363 Shasta Drive
Palo Alto, CA 94306

© 1979 Glenn Story

In this article I will tell how I went about connecting my home computer, a SOL-20, to the large-scale computer located where I work, which is the Itel Technical Support Center in Palo Alto, California. I will deal primarily with the software I wrote for my SOL, but also touch upon other software and hardware involved, as well as the predefined interfaces with which I worked. – *GS*

MY HARDWARE

As I said above, I have a SOL-20 computer from Processor Technology. It contains 24K of RAM and uses a Northstar disk for external storage. When I set about this project I acquired a Tec-Com Modem with acoustic coupler. This device, which plugs into the serial port on my SOL, converts the digital signals from the SOL into an audible sound transmitted over a phone line, and translates the incoming sound-signal back to digital form. The former process is called modulation, and the latter demodulation; thus, modulator-demodulator, or modem for short. The interface from the modem to the telephone is done acoustically rather than electronically: a small speaker and microphone on the modem are attached to rubber sleeves into which the telephone handset is placed. This is what is meant by "acoustic coupler."

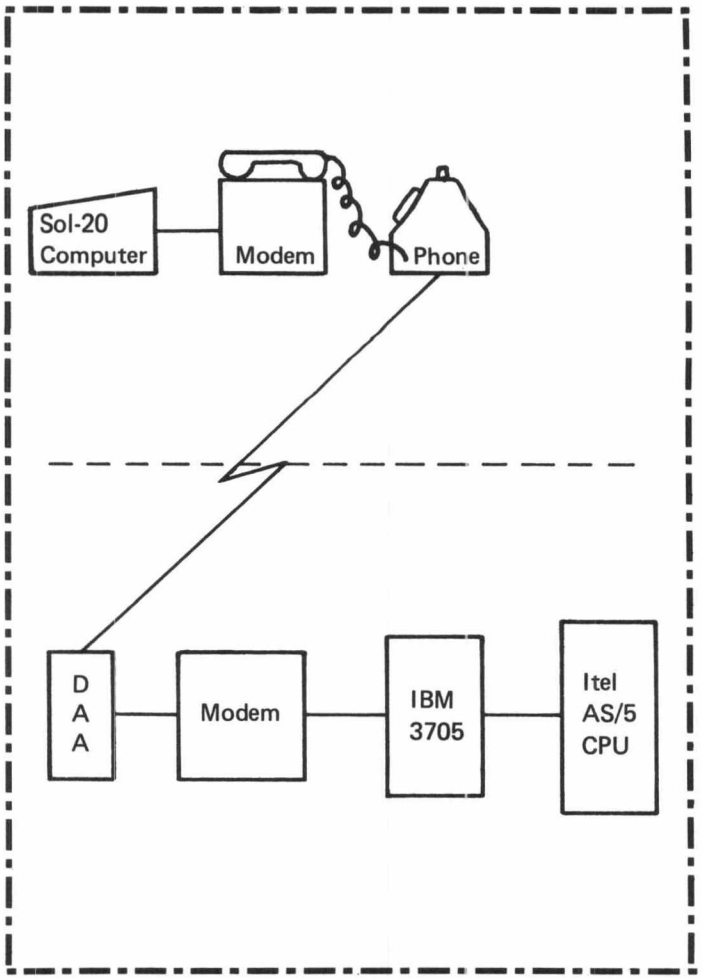
HARDWARE AT THE OTHER END

The computer to be hooked up to is an Itel AS/5 (or other equivalent machine). Itel's Advanced System computers are hardware and software compatible with IBM's System/370 series. The AS/5 is attached through an IBM 3705 programmable teleprocessing control unit to a bank of auto-answer modems, which in turn are connected to the phone network via a phone-company supplied data access arrangement (DAA).

The job of the IBM 3705 is to take over some of the burden of controlling the interface to remote terminals, so the AS/5 is freed to perform its primary job of running people's programs.

The term "auto-answer", describing the modems, indicates they have the ability to automatically answer an incoming call. Thus, when I want to establish communication between my computer and the one at Itel, I dial the phone number of this bank of modems; one of them answers my call and starts sending a high-pitched tone called a carrier or data tone. When I hear the tone, I place my phone's handset into my acoustic coupler. When my modem "hears" the carrier from the other end, it starts sending its own carrier (at a different pitch), and communication is established.

The DAA or Data Access Arrangement, which Itel uses, is an alternative to the acoustic coupler (which is what I use) as a means of interfacing data-processing equipment to telephone equipment. The phone company has always been nervous about other people hooking their electronics into the phone system. The acoustic coupler avoids this problem, since there is no direct electrical connection between the computer equipment and the phone. However, you can make a direct connection, provided it is done through a DAA, which contains circuits guarding against unacceptable signals, voltages, etc., being introducted into the phone system. Until recently, DAAs could only be obtained from the phone company. (See Figure 1 for a diagram of the hook-up from my CPU to Itel's).



MY SOFTWARE

Supplied with my SOL was a routine in ROM which would allow my computer to behave like a TTY (teletype) terminal. However, I quickly discovered a number of shortcomings in this software and set out to design my own. The bulk of this article is a discussion of that software.

SOFTWARE AT THE OTHER END

The software running at the other end which is of primary interest is called VM/370 (Virtual Machine facility/370). This IBM software is designed to let a number of users, sitting at terminals, have the impression that they are each running their own dedicated System/370 machine. This "virtual machine" has virtually (!) all of the features and facilities of a real machine, time-sharing to the nth degree. (Because of the compatability which exists between IBM and Itel machines, this software can run unmodified on our AS/5.)

There are two features of VM/370 which I would like to discuss, since they played a part in the design of the software I wrote to run in my SOL.

When I dial into the VM system from my computer at home (or from some other terminal), I must first identify myself to VM via a "userid" supplied to me by the VM system programmer at Itel. This userid is also known to my coworkers so we may communicate amongst one another over the VM system. I must also supply a "password" to "prove" that I am who I say I am. This password, also given to me by the system programmer, is known to no one else, and it's my job to keep it that way. I may also have to supply additional passwords to gain access to certain data stored in the VM system. One of the requirements of the software for my SOL is that it protect my password by inhibiting its display on the screen.

The other feature of VM/370 which I want to point out is a set of characters called "line-editing" characters. For TTY terminals, each character is transmitted to the computer as it is keyed in. One cannot simply back up the cursor to correct mistakes. The three primary characters are shown in Table 1.

Table 1. VM Line-editing Characters.

Name	Purpose	Normal Char.	My Char.
Char. delete	Delete preceding character	@	
Line delete	Delete preceding line	¢	E
Line end	Separate multiple lines	#	#

As an example of their use:

abcd@e#xyz¢#123

would be interpreted as:

abce
123

LINE PROTOCOL

The term "line protocol" refers to the exchange of special control characters between computers or between a computer and a terminal, which ensures the two ends of the conversation are "in sync" with one another. They are analogous to saying "over" when talking on a two-way radio, so the person at the other end knows it's his/her turn to talk. Other terms for line protocol are "line discipline", and "data link control". Of the three protocols currently in wide use, asynchronous (the one used with TTY-type terminals) is the simplest. As implemented with VM/370 its essentials are as follows:

1. The terminal user indicates that s/he is through entering a line by pressing either carriage-return (ODH) or control-S (13H). (If you are unfamiliar with micro-computer assembly-language conventions, 13H means "13 hexadecimal".)
2. When the computer receives either of the characters indicated above, it responds by sending: carriage-return (ODH), line-feed (OAH), direct-control-3 (or DC3) (13H). (Note that DC3 is the same as control-S.)
3. When the computer wishes to invite the remote terminal to send, it sends a prompting character. The prompting character may be selected as either a period (.) (2EH) or an end-of-transmission (EOT) (04H). The EOT is generally selected for equipment which recognizes the character in the hardware or software and unlocks the keyboard or some such. The period is used on other devices as a visual cue to the terminal user to enter input. The prompting character is followed by a direct-control-1 (DC1) (11H).

4. Although its use is not specifically defined under VM/370, the escape character (ESC) (1BH) is used to mean that a special device control character follows.

These rules are summarized in Table 2, and an example of their use is shown in Figure 2.

Figure 2. Sample terminal/computer Exchange, showing control characters.

```
Computer:  Hello (CR) (LF) (DC3)
Computer:  Enter your name. (CR) (LF) (DC3)
Computer:  (EOT or ".") (DC1)
Terminal:  Glenn (DC3, or CR)
Computer:  (CR) (LF) (DC3)
```

Table 2. Line Control Characters.

Name	Abrev.	Hex Code	Meaning
End of transmission	EOT	04	Invite terminal to send
Carriage return	CR	0D	Move carrier or cursor to left margin. On input from terminal signal end of input
Line feed	LF	0A	Move paper up one line or cursor down one line
Direct control 1	DC1	11	Invite terminal to send input
Direct control 3	DC3	13	End of line
Escape	ESC	1B	Special device-control character(s) follow(s)

DETAILS OF THE SERIAL-PORT INTERFACE

Before I begin the detailed description of my program, I should describe in more detail the interface to the serial port, which is my connection to the modem, and thus the world. There are two levels of interface: software and hardware.

SOFTWARE INTERFACE: This is via routines in ROM. There are two such routines, one for input and one for output. AINP is used for input. Before calling, A is set to 1 to indicate the serial port. Upon return, the zero flag is set if no character has been received, or cleared if a character is available. In this latter case, the character is in the A register. AOUT is used for output. Again, A is set to 1 for serial port. The character to be sent is placed in the B register. Upon return from AOUT, the character has been transmitted. **HARDWARE INTERFACE**: This is via two I/O ports; F8H is a status port, and F9H is a data port. To determine the status of the serial port, the instruction IN 0F8H is issued. The values of the bits which are returned from this instruction are as follows: (See Figure 3 for a summary)

Figure 3. Status Port Bit Values.

```
0 - Carrier is being received if 1
1 - The modem is not ready if 1
2 - A parity error was detected if 1
3 - A framing error if 1
4 - An overrun error if 1
5 - Not clear to send if 1
6 - An input character is waiting to be read if 1
7 - The SOL hardware is ready to send a character if 1
```

0. A zero here indicates that the connection to the remote computer has been lost. This can be caused by a break in the phone connection, a VM/370 decision to hang up (too many bad passwords or I logged off), or a VM/370 software failure.
1. This bit is one if I forget to turn on my modem.
2. This bit is one if a character is received with bad parity (i.e., an incorrect number of one bits).
3. A framing error is caused by incorrect start and/or stop bits. The format of a valid character is, one start bit (always zero), seven ASCII data bits, one parity bit, and two stop bits (always one). If this format is not adhered to, a framing error results.
4. An overrun occurs if my software does not read a data character from the serial data port before the next character is received. (In this case, only the last character sent will be available.)
5. This is a control signal from the modem (as are bits 0 and 1). So far as I can tell, my modem always has this bit the opposite of bit 0.
6. A one in this bit means that there is a character waiting to be read in the serial input data port.
7. A one in this bit means that the SOL hardware is ready to receive a character in the serial output data port.

I have chosen to use the software interface for output in my routine. I also use the software interface for reading data from the modem, but I supplement this by directly reading the status port to look for errors.

LIMITATIONS OF SUPPLIED SOL SOFTWARE

When I first started using my SOL as a VM/370 terminal, I discovered the following limitations of the terminal software which came with the computer. (This is the TERM command of SOLOS).

1. The control characters DC1 and DC3 display on the screen.
2. If I use carriage return to terminate an input line, it erases the entire input line (due to peculiarities in the SOL software for driving the video display). Thus all my input lines were disappearing.
3. There is no way of suppressing the display of passwords on the screen. (At least they disappeared when I hit carriage-return!)
4. It does not check for errors from the serial port hardware.

Because of these limitations, I decided to write my own software to allow my SOL to function as a terminal to VM/370.

GOALS OF MY SOFTWARE

1. It must overcome the limitations above.
   - DC1 and DC3, when received, will not be sent to the video screen.
   - When I press carriage-return, a DC3 will be transmitted instead. (Remember, that this is the alternate way of terminating an input line; this prevents my input lines from disappearing.)
   - When I want to enter a password, I first press the ESCAPE key. This causes my software to send spaces to the screen for the characters I key in. This continues until I press carriage-return.

• I check for errors before reading a character from the serial port. I display messages on the screen as appropriate.

2. If the MODE SELECT key is pressed at any time, the program terminates and gives control to Northstar DOS.
3. If I strike a key (other than MODE SELECT) at a time when VM/370 is not expecting input, the character is not transmitted.
4. I display on the screen the effects of the VM line-editing characters, line delete, and character delete.
5. All messages generated by my software are displayed in reverse letters (i.e., white on black instead of black on white) to distinguish from information sent from VM.
6. If I press a SOL screen-control key, it is sent directly to the SOL video software, not the modem, e.g., pressing CLEAR clears the screen.

What follows is a detailed description of the logic of the program which I wrote to interface my SOL to the VM/370 computer. The reader should reference the source listing, supplied, while reading this section.

The program begins by testing to see if the modem is ready. If it is not, a message is issued and the program loops until the modem is powered on. This same process is then repeated looking for a carrier, which, when detected, means that the connection is made to the equipment at Itel.

The logic then enters the main loop of the program, beginning at label, TOP. A check is made for error conditions, and a message is issued if any are found. I then check to see if a data byte has been received from the modem. If not, I jump to CKIN to look for a key pressed on the keyboard.

When a character is received from the modem, it is checked for several specific values. If it is a DC1, a switch is turned on to allow keyboard input, and the character is dropped. (i.e., it is not sent to the screen.) If it is a DC3 it is also dropped. If the character is an ESC, control is transferred to a special routine, described later.

Next a write switch is tested. This switch is normally on, in which case the character is written to the screen. If it is off (which is caused by my pressing the ESCAPE key to enter a password), a blank is sent to the screen instead of the input character. Finally the ROM routine SOUT is called to write the character (or blank) to the screen. Subroutine SOUT will handle such control characters as carriage-return, DEL, and line-feed. We then jump back to TOP to check for another character from the modem.

The code beginning at label CKIN handles keyboard input. The SOL ROM routine SINP is called to look for a character. If none is found, control returns to TOP to check the modem again. Keys with the high-order bit (80H) on are screen control keys (such as CLEAR) and are sent directly to the screen software via subroutine SOUT. The exception is MODE SELECT (80H) which causes the program to terminate. Any other character is only accepted if a read switch is on. This switch is turned on by the receipt of DC1 from the serial port. If the switch is off, control returns to the TOP of the loop; if it's on, the character is checked to see if it is a carriage-return (0DH). If it is, the read switch is turned off (so no more keyboard characters will be processed until another DC1 is received from VM), and the write switch is turned on, terminating password suppression. A carriage return is changed to a DC3 (13H) for transmission to the serial port. If the character is not a carriage return it is checked to see if it is an ESCAPE. If so, the write switch is turned off so what follows is not displayed on the screen. The escape key will not be sent to the modem. If the key pressed was a DEL (7FH), it is changed to "_" (5FH) which (1) is defined as the character-delete character to VM; and, (2) causes the SOL video software to back up the cursor one position. If the key pressed was a [, which is the VM line-delete character, then a routine is entered which will blank out everything on the screen back to either a line-end character (#) or the beginning of the line on the screen. This routine must update the field NCHAR in the SOL work area to reflect the new cursor position. Finally, the character is written to the modem by calling SOL subroutine AOUT.

The next routine is the "escape sequence handler." Basically, if VM sends the sequence, ESC (1BH), 1 (31H), my program responds with a predefined reply. The routine, ESCAPE, deletes the remainder of the incoming message which began with ESC 1, and then sends the reply. The purpose of this is to automatically set up terminal characteristics of my terminal. Here we have an example of two programs talking to each other. The program at the other end, which is written in a special interpretive language, looks like this, in part:

```
&TYPE e1 ENTER TERMINAL BRAND
&READ VARS &BRAND &LL &LD &CD
&IF &INDEX EQ 4 &GOTO –SOL
&TYPE ENTER LINE LENGTH
&READ VARS &LL
. . . etc. . . .
```

where e1 is actually ESC 1.

When my program in the SOL sees the ESC1, it "swallows" "ENTER TERMINAL BRAND." It then transmits "SOL 63 [_" where _ is the delete character, 5FH. If the above VM program is used with some other terminal, the ESC 1 will be ignored, and the message "ENTER TERMINAL BRAND" will appear on the terminal. The &IF statement in the above coding says, if all four values were entered go to the SOL routine. Otherwise, fall through and ask for the remaining values to be entered from the terminal.

The next routine in my SOL program, GETM, is a subroutine which gets a byte from the serial port. This subroutine is used by ESCAPE, the routine described above.

Next is ERROR which issues an error message and jumps to the very beginning of the program, so that if the error involved loss of the carrier, we can loop and wait for the carrier to be restored. (This generally entails my hanging up the phone and redialing.)

The final subroutine in this program is the message writer. This routine passes the characters, one at a time, to the SOL routine OCHAR. I am actually entering the SOL routine, SOUT, in the middle of its processing. I turn on the high-order bit (ORI 80H) to reverse the character. If I called SOUT, that bit would be stripped off.

FUTURE PLANS

I have several specific plans for future software interfacing between my SOL and the "big machines":

1. I want to write a set of SOL and VM programs to interchange disk files between the two environments.

2. I want to write similar programs so that I can send printed output from the VM machine to my selectric device. (The problem here is that VM transmits at 30 cps (characters per second), whereas my selectric runs at 14.5 cps. So VM must wait at the end of each line for my typewriter to catch up.)
3. I want to rewrite the program described in this article to take better advantage of the video screen: I want to divide the screen into an output area, input area, and status area. Then error messages would not interfere with the display of output. I also want to be able to key in data before it's requested and transmit the entire line when the DC1 is received.

CONCLUSION

By writing my own software routine, I was able to improve the way in which I can interface my home computer to a large-scale time-sharing computer. In the process I gained a good deal of insight into the hardware and software requirements for such an interface. Moreover, my experience has shown me the value of having programability at both ends of the line.

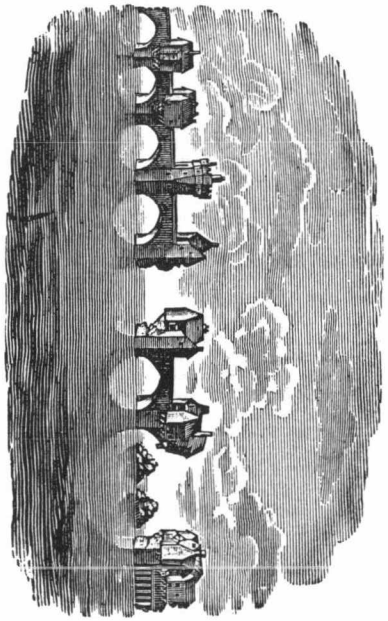
ACKNOWLEDGEMENTS

I truly appreciate the willingness of Itel management to allow me to hook up my "toy" with their multimillion dollar equipment. I also want to acknowledge Doug Martin, my co-worker on the Itel software staff, whose technical information and advice regarding teleprocessing hardware and software were invaluable.

Listing

```
                        LIST
                        ENDIF
                ;  TERM - PROGRAM TO INTERFACE SOL AS A
                ;  TERMINAL TO VM/370
                ;  WRITTEN 11/03/78 BY GLENN STORY
                ;
                ;  THIS PROGRAM READS FROM THE CURRENT
                ;  INPUT PORT AND WRITES TO PORT 1 (MODEM)
                ;  IT READS FROM PORT 1 (MODEM) AND WRITES
                ;  TO THE CURRENT OUTPUT PORT
                ;
                ;  IT IMPROVES OVER THE SOLOS 'TERM' COMMAND IN THE
                ;  FOLLOWING WAYS -
                ;
                ;  1.  FOR INPUT (FROM THE TERMINAL TO THE MODEM)
                ;          A.  CARRIAGE RETURN IS TRANSLATED TO DC3
                ;          B.  ESCAPE DISABLES OUTPUT TO SCREEN UNTIL
                ;                  A CR IS PRESSED
                ;          C.  DEL IS TRANSLATED TO @
                ;          D.  MODE SELECT RETURNS TO DOS
                ;          E.  ESCAPE SEQUENCE FEEDS CONSTANT MESSAGE
                ;  2.  FOR OUTPUT (FROM MODEM TO TERMINAL)
                ;          A.  DC1 AND DC3 ARE DROPPED FROM DATA
                ;          B.  ERRORS ARE DETECTED FROM THE SERIAL PORT
                ;
0000                    URG     2A00H
                ;
2A00            ;  ISSUE INITIAL MESSAGE
2A00            ;
2A00 21992B             LXI     H,GOMSG;  GET STARTING MESSAGE
2A03 CD832B             CALL    MSG;  WRITE IT
2A06            ;
2A06            ;  CHECK FOR A CARRIER
2A06            ;WAIT FOR CARRIER IF NECESSARY
2A06            ; EXIT FROM WAIT WHEN CARRIER DETECTED
2A06            ;OR MODE-SELECT PRESSED ON KEYBOARD
2A06            ;
2A06 DBF8       NEWC    IN      0F8H    ;GET SERIAL PORT STATUS
2A08 E602               ANI     2       ;IS THE DATA SET READY?
2A0A CA202A             JZ      NC2     ;JUMP IF SO
2A0D 213B2C             LXI     H,WD    ;INDICATE THE MESSAGE
2A10 CD832B             CALL    MSG
2A13 DBF8       DLUP    IN      0F8H    ;GET STATUS
2A15 E602               ANI     2       ;IS DATA SET READY
2A17 C2132A             JNZ     DLUP
2A1A 214F2C             LXI     H,DSR
2A1D CD832B             CALL    MSG
2A20 DBF8       NC2     IN      0F8H    ;GET STATUS
2A22 E601               ANI     1       ;IS THERE A CARRIER?
2A24 CA482A             JZ      TOP     ;JUMP IF SO
2A27 21FC2B             LXI     H,WC    ;INDICATE MESSAGE
2A2A CD832B             CALL    MSG
2A2D DBF8       CLUP    IN      0F8H    ;GET SERIAL PORT STATUS
2A2F E601               ANI     1       ;IS THERE A CARRIER?
2A31 CA422A             JZ      GOTC    ;JUMP IF SO
2A34 CD1FC0             CALL    SINP    ;GET KEYBOARD STATUS
2A37 CA2D2A             JZ      CLUP    ;JUMP IF NO KEY PRESSED
2A3A FE80               CPI     80H     ;WAS MODE SELECT PRESSED?
2A3C C22D2A             JNZ     CLUP    ;JUMP IF NOT
2A3F C3D72A             JMP     MODE    ;JUMP IF SO
2A42 21122C     GOTC    LXI     H,GOTMSG
2A45 CD832B             CALL    MSG     ;WRITE MESSAGE
2A48            ;
2A48            ;  CHECK SERIAL STATUS PORT FOR ERROR CONDITIONS
2A48            ;
2A48 DBF8       TOP     IN      0F8H    ;GET SERIAL PORT STATUS
2A4A 329A2C             STA     LASTS   ;SAVE STATUS
2A4D 47                 MOV     B,A     ;SAVE IT IN B
2A4E 21CD2B             LXI     H,NOC   ;POINT TO AN ERROR MESSAGE
2A51 E601               ANI     01H     ;IS THERE A CARRIER?
2A53 C27D2B             JNZ     ERROR   ;JUMP IF NOT
2A56 78                 MOV     A,B     ;RESTORE
2A57 21DC2B             LXI     H,NOL
2A5A E602               ANI     02H     ;IS THE LINK READY?
2A5C C27D2B             JNZ     ERROR   ;JUMP IF NOT
2A5F 78                 MOV     A,B     ;RESTORE
2A60 21ED2B             LXI     H,PE
2A63 E604               ANI     04H     ;IS THERE A PARITY ERROR?
2A65 C27D2B             JNZ     ERROR   ;JUMP IF SO
2A68            ;
2A68            ;  READ A DATA BYTE FROM THE MODEM
2A68            ;  AND WRITE IT TO STANDARD OUTPUT PORT
2A68            ;
2A68 3E01               MVI     A,1     ;INDICATE MODEM INPUT
2A6A CD22C0             CALL    AINP    ;LOOK FOR A BYTE FROM THE LINE
2A6D CABE2A             JZ      CKIN    ;JUMP TO CHECK FOR INPUT IF NO DATA
2A70 32992C             STA     LASTC
2A73 E67F               ANI     7FH     ;TURN OFF PARITY BIT
2A75 FE07               CPI     7;  IS IT BELL?
2A77 C28B2A             JNZ     UCK;  JUMP IF NOT
2A7A DBFA               IN      0FAH;  GET STATUS
2A7C E604               ANI     4;  IS THE HARD-COPY DEVICE READY
2A7E C2482A             JNZ     TOP;  JUMP IF NOT
2A81 3E03               MVI     A,3;  INDICATE HARDCOPY DEVICE
2A83 0620               MVI     B,' ';  INDICATE SPACE CHAR
2A85 CD1CC0             CALL    AOUT;  WRITE TO HARDCOPY
2A88 C3482A             JMP     TOP;  GO DO IT ALL AGAIN
2A8B FE5F       UCK     CPI     5FH;  IS IT AN UNDERSCORE
2A8D C29A2A             JNZ     DC1CK;  JUMP IF NOT
2A90 47                 MOV     B,A;  SAVE CHAR IN B
2A91 3A08C8             LDA     NCHAR;  GET CURRENT CHARACTER POSITION
```

```
2A94  B7              ORA   A;  SEE IF IT'S ZERO
2A95  78              MOV   A,B;  RESTORE CHAR
2A96  C29A2A          JNZ   DC1CK;  JUMP IF NOT
2A99  AF              XRA   A;  SET CHAR TO NULL
2A9A  FE11    DC1CK   CPI   11H  ;IS IT A DC1?
2A9C  C2A82A          JNZ   CKDC3  ;JUMP IF NOT
2A9F  21E32A          LXI   H,RDON  ;TURN ON READ
2AA2  22E12A          SHLD  RDSW+1
2AA5  C3482A          JMP   TOP
2AA8  FE13    CKDC3   CPI   13H  ;IS IT DC3?
2AAA  CA482A          JZ    TOP  ;JUMP IF SO (DROP THE CHAR)
2AAD  FE1B            CPI   1BH  ;IS IT ESCAPE?
2AAF  CA3C2B          JZ    ESCAPE  ;JUMP IF SO
2AB2  C3B72A  WRSW    JMP   WRON  ;WRITE SWITCH
2AB5  3E20    WROFF   MVI   A,20H  ;SET OUTPUT TO SPACE IF OFF
2AB7  47      WRON    MOV   B,A  ;MOVE THE CHAR TO B REG
2AB8  CD19C0          CALL  SOUT  ;WRITE IT OUT
2ABB  C3482A          JMP   TOP  ;AND GO POLL AGAIN
2ABE  CD1FC0  CKIN    CALL  SINP  ;LOOK FOR A CHAR FROM STANDARD PORT
2AC1  CA482A          JZ    TOP  ;START OVER IF NO CHAR
2AC4  47              MOV   B,A  ;SAVE IT
2AC5  E680            ANI   80H  ;CHECK FOR SCREEN CONTROL KEY
2AC7  78              MOV   A,B
2AC8  CAE02A          JZ    RDSW  ;JUMP IF NOT
2ACB  78              MOV   A,B  ;RESTORE CHAR
2ACC  EE80            CPI   80H  ;IS IT MODE SELECT?
2ACE  CAD72A          JZ    MODE  ;  JUMP IF SO
2AD1  CD19C0          CALL  SOUT  ;WRITE IT DIRECT TO SCREEN
2AD4  C3482A          JMP   TOP  ;AND START OVER
2AD7  21B42B  MODE    LXI   H,ENDMSG  ;POINT TO END MESSAGE
2ADA  CD832B          CALL  MSG  ; CALL MESSAGE SUBROUTINE
2ADD  C32820          JMP   DOS  ;RETURN TO DOS
2AE0  C3E32A  RDSW    JMP   RDON  ; READ SWITCH
2A48          RDOFF   EQU   TOP  ;RETURN TO POLL IF OFF
2AE3  FE0D    RDON    CPI   0DH  ;IS IT A CARRIAGE RETURN
2AE5  C2F92A          JNZ   CKESC  ;JUMP IF NOT
2AE8  3E13            MVI   A,13H  ;CHANGE IT TO DC3
2AEA  21482A          LXI   H,RDOFF  ;  TURN READ SWITCH OFF
2AED  22E12A          SHLD  RDSW+1
2AF0  21B72A          LXI   H,WRON  ;TURN ON WRITE SWITCH
2AF3  22B32A          SHLD  WRSW+1
2AF6  C3332B          JMP   ENDCK
2AF9  FE1B    CKESC   CPI   1BH  ;CHECK FOR ESCAPE
2AFB  C2072B          JNZ   LDCK;  JUMP IF NOT
2AFE  21B52A          LXI   H,WROFF  ;TURN WRITE SWITCH OFF
2B01  22B32A          SHLD  WRSW+1
2B04  C3482A          JMP   TOP  ;DON'T WRITE THIS CHAR
2B07  FE5B    LDCK    CPI   '[';  IS IT A LINE DELETE?
2B09  C22C2B          JNZ   DELCK;  JUMP IF NOT
2B0C  CD1CC1          CALL  VDADD;  GET VIDEO BUFFER ADDRESS IN HL
2B0F  7E      LDLUP   MOV   A,M;  GET THE CHAR
2B10  3620            MVI   M,' ';  REPLACE IT WITH A SPACE
2B12  FE23            CPI   '#';  IS IT A LINE SEPARATOR?
2B14  CA212B          JZ    ENDLD;  JUMP IF SO
2B17  7D              MOV   A,L;  GET LOW-ORDER BYTE OF BUFFER ADDRESS
2B18  E63F            ANI   3FH;  IS IT BEGINNING OF LINE?
2B1A  CA212B          JZ    ENDLD;  JUMP IF SO
2B1D  2B              DCX   H;  DECREMENT THE ADDRESS
2B1E  C30F2B          JMP   LDLUP;  AND LOOP BACK
2B21  7D      ENDLD   MOV   A,L;  GET LOW-ORDER BYTE OF BUFFER ADDRESS
2B22  E63F            ANI   3FH;  TURN OFF HIGH ORDER BITS
2B24  3208C8          STA   NCHAR;  STORE IT IN SOL SYSTEM RAM
2B27  3E5B            MVI   A,'[';  RESTORE THE ORIGINAL CHARACTER
2B29  C3332B          JMP   ENDCK;  AND JUMP
2B2C  FE7F    DELCK   CPI   7FH  ;IS IT A DEL?
2B2E  C2332B          JNZ   ENDCK  ;JUMP IF NOT
2B31  3E5F            MVI   A,5FH  ;CHANGE IT TO VM'S DELETE CHAR
2B33  47      ENDCK   MOV   B,A  ;MOVE THE CHAR TO B REG
2B34  3E01            MVI   A,1  ;INDICATE MODEM OUTPUT
2B36  CD1CC0          CALL  AOUT  ;AND WRITE A CHAR
2B39  C3482A          JMP   TOP  ;AND START ALL OVER AGAIN
2B3C                ;
2B3C                ;  ESCAPE SEQUENCE HANDLER
2B3C                ;
2B3C  CD682B  ESCAP   CALL  GETM  ;GO GET A BYTE FROM MODEM
2B3F  21232C          LXI   H,EE
2B42  FE31            CPI   '1'  ;IS NEXT CHAR A 1
2B44  C27D2B          JNZ   ERROR  ;JUMP IF NOT
2B47  CD682B  ESCLU   CALL  GETM
2B4A  FE11            CPI   11H  ;IS IT DC1?
2B4C  C2472B          JNZ   ESCLUP  ;LOOP UNTIL SO
2B4F  21602C          LXI   H,EMSG;  GET ESCAPE MESSAGE
2B52  CD832B          CALL  MSG;  AND WRITE IT
2B55  21872C          LXI   H,CMDSEQ  ;GET COMMAND SEQUENCE
2B58  7E      RDLUP   MOV   A,M  ;GET A CHAR FROM CANNED MESSAGE
2B59  FE0D            CPI   0DH  ;IS IT THE END?
2B5B  CA482A          JZ    TOP  ;JUMP IF SO
2B5E  47              MOV   B,A
2B5F  3E01            MVI   A,1  ;INDICATE MODEM OUTPUT
2B61  CD1CC0          CALL  AOUT  ;WRITE A CHAR OUT
2B64  23              INX   H;BUMP TO NEXT CHAR
2B65  C3582B          JMP   RDLUP  ;AND LOOP AGAIN
2B68                ;
2B68                ; SUBROUTINE GETM
2B68                ; GETS A BYTE FROM MODEM
2B68  DBF8    GETM    IN    0F8H  ;GET STATUS OF MODEM
2B6A  21CD2B          LXI   H,NOC
2B6D  E601            ANI   1  ;IS THERE A CARRIER
2B6F  C27D2B          JNZ   ERROR  ;JUMP IF NOT
2B72  3E01    GMLUP   MVI   A,1  ;INDICATE MODEM INPUT
2B74  CD22C0          CALL  AINP  ;GET A BYTE
2B77  CA722B          JZ    GMLUP
2B7A  E67F            ANI   7FH  ;REMOVE PARITY
2B7C  C9              RET
2B7D                ;
2B7D                ;  ; WRITE ERROR MESSAGES
2B7D                ;
2B7D  CD832B  ERROR   CALL  MSG
2B80  C3062A          JMP   NEWC
2B83                ;
2B83                ; SUBROUTINE MSG - WRITES MESSAGE TO SCREEN
2B83                ;
2B83  E5      MSG     PUSH  H  ;SAVE MESSAGE POINTER
2B84  CDF9C2          CALL  CRLF  ; RETURN THE CARRIAGE
2B87  E1      MSGLU   POP   H  ;RESTORE MESSAGE POINTER
2B88  7E              MOV   A,M  ;GET A BYTE OF THE MESSAGE
2B89  FE0D            CPI   0DH  ;IS IT A CARRIAGE RETURN?
2B8B  CAF9C2          JZ    CRLF  ;  ; JUMP IF SO
2B8E  F680            ORI   80H  ; REVERSE IT
2B90  47              MOV   B,A  ; AND MOVE IT TO B
2B91  23              INX   H  ;BUMP THE COUNTER
2B92  E5              PUSH  H  ;SAVE IT
2B93  CD98C0          CALL  OCHAR; WRITE IT OUT
2B96  C3872B          JMP   MSGLUP
2B99                ;
2B99                ;  ;MESSAGES
2B99                ;
2B99  20564D20  GOMSG   DB    ' VM Terminal mode entered '
      5465726D
      696E616C
      206D6F64
      6520656E
      74657265
      6420
2BB3  0D                DB    13
2BB4  20564D20  ENDMS   DB    ' VM Terminal mode ended '
      5465726D
      696E616C
      206D6F64
      6520656E
      64656420
2BCC  0D                DB    13
2BCD  204C6F73  NOC     DB    ' Lost carrier '
      74206361
      72726965
      7220
```



```
2BDB  0D              DB      13
2BDC  204C696E        NOL     DB      ' Link not ready '
      6B206E6F
      74207265
      61647920
2BEC  0D              DB      13
2BED  20506172        PE      DB      ' Parity error '
      69747920
      6572726F
      7220
2BFB  0D              DB      13
2BFC  20576169        WC      DB      ' Waiting for carrier '
      74696E67
      20666F72
      20636172
      72696572
      20
2C11  0D              DB      13
2C12  20436172        GOTMS   DB      ' Carrier detect '
      72696572
      20646574
      65637420
2C22  0D              DB      13
2C23  20457363        EE      DB      ' Escape sequence error '
      61706520
      73657175
      656E6365
      20657272
      6F7220
2C3A  0D              DB      13
2C3B  20547572        WD      DB      ' Turn the modem on '
      6E207468
      65206D6F
      64656D20
      6F6E20
2C4E  0D              DB      13
2C4F  20446174        DSR     DB      ' Data set ready '
      61207365
      74207265
      61647920
2C5F  0D              DB      13
2C60  20546572        EMSG    DB      ' Terminal characteristics transmitted ',0DH
      6D696E61
      6C206368
      61726163
      74657269
      73746963
      73207472
      616E736D
      69747465
      64200D
2C87
2C87  534F4C20        CMDSE   DB      'SOL 63 [ ',5FH,'
      3633205B
      205F2020
      20202020
2C97  13                      DB      13H  ;DC3 TO TERMINATE DATA TO VM
2C98  0D                      DB      0DH  ;CARRIAGE RETURN TO TERMINATE TO SOL
2C99  00              LASTC   DS      1
2C9A  00              LASTS   DS      1
C098                  OCHAR   EQU     0C098H  ;ROUTINE IN SOLOS
C11C                  VDADD   EQU     0C11CH;  SOLOS ROUTINE TO FIND
2C9B                  ;               CURRENT VIDEO BUFFER ADDRESS
C808                  NCHAR   EQU     0C808H;  CURRENT CHAR POSITION IN BUFFER
2C9B                          PRTSYM
```

End

BY L. H. REID
22, Greystones Crescent
Sheffield S1 1GP
England

Dear *Dr. Dobb's*,

I herewith enclose two programs which I have found very useful to me when used on my system.

I have been subscribing to *Doctor Dobb's* now for two years and had to go out and buy the bound copy of *Volume One*. I also subscribe to *Byte*, *Interface Age* and *Kilobaud* although it is because the *Doctor* has supplied me with such valuable information and software that I am sending you these. I have got the 8080 Emulator up and running despite a listing error (line 186 should be 'CLR BREG' not 'CLR B') and have used the Palo Alto BASIC you published ages ago in *Volume One.*

I hope you think these are worth publishing as I was prompted to send these after seeing Gary Gaugler's 'Quolst' program in a recent issue (*DDJ* #33).

My own system comprises an MSI 6800 system with a Dual drive BFD-68, Teletype 43, GRI keyboard and home-brew VDU.

We are alive and well over this side of the Atlantic and can write software !!!

Best Wishes to *Dr.*,

Lindsay Reid.

PRTECT YOUR SPACE

space

'SPACE' is a transient command for use with the Smoke Signal BFD-68 Disk system for 6800 processors. I had found it annoying to have to list a whole disk catalogue just to find the space available left on a drive. This program saves all that and gives it to you more quickly than "QUOLST" by Gary Gaugler, published recently—only because it does not have to sort out all the file types. It is one of those commands whereby one wonders why nobody did it before—perhaps they did and I missed it !!

To use the command one simply enters:—

SPACE (,optional drive number)

It really is that easy !

```
                NAM     SPACE

                ******************************************************
                *
                *                  SPACE
                *
                *  THIS IS TRANSIENT COMMAND FOR USE WITH THE
                *  6800 AND SMOKE SIGNAL BFD-68 DISK SYSTEM.
                *  IT SHOWS AVAILABLE SPACE LEFT ON THE SPECIFIED
                *  DRIVE (DEFAULT TO ZERO). IF AN ILLEGAL DRIVE IS
                *  SPECIFIED, SPACE ON DRIVE ZERO IS GIVEN.
                *
                *  (C) 1979 BY LINDSAY REID (ENGLAND).
                *  ALL RIGHTS RESERVED. NO REPRODUCTION FOR
                *  COMMERCIAL PURPOSES MAY BE MADE WITHOUT
                *  WRITTEN PERMISSION FROM THE ABOVE AUTHOR.
                *
                ******************************************************

                * EXTERNAL EQUATES ***********
7283            ZWARMS EQU    $7283       SSB DOS WARM START
72AF            ZOUTHA EQU    $72AF       OUTPUT A HEX ADDRESS
72A9            ZTYPDE EQU    $72A9       SHOW DISC ERROR
72A6            ZOUTST EQU    $72A6       OUTPUT STRING TILL A NULL
7297            ZGNCHR EQU    $7297       GET NEXT BYTE FROM I/P BUFFER
E1D9            OUTCH2 EQU    $E1D9       OUTPUT THE BYTE IN A REG
7786            DFM    EQU    $7786       CALL DISC FILE MANAGEMENT

7080                   ORG    $7080       TRANSIENT COMMAND AREA
                       OPT    NOG

7080 CE 70 E9          LDX    #FCB        GET FCB ADDRESS INTO IX
7083 BD 72 97          JSR    ZGNCHR      GET DRIVE NUMBER (IF THERE)
7086 81 30             CMP A  #'0         IS IT ZERO ?
7088 2F 0A             BLE    ZERO        IF LESS THAN ASCII 0 FORCE 0
708A 81 33             CMP A  #'3         IS IT THREE OR MORE
708C 2C 06             BGE    ZERO        IF SO FORCE DRIVE 0
708E 80 30             SUB A  #'0         MAKE INTO HEX
7090 A7 02             STA A  2,X         STORE IN FCB XUN
7092 20 02             BRA    CON         ALL SET SO BRANCH
7094 6F 02      ZERO   CLR    2,X         FORCE DRIVE 0
7096 6F 00      CON    CLR    0,X         SET UP FOR QFREE
7098 BD 77 86          JSR    DFM         CALL DFM
709B 24 06             BCC    ALLOK       IF NO ERROR BRANCH
709D BD 72 A9          JSR    ZTYPDE      SHOW ERROR
70A0 7E 72 83          JMP    ZWARMS      AND WARM START INTO DOS
70A3 B7 70 E7   ALLOK  STA A  TEMP        SAVE HI SPACE
70A6 F7 70 E8          STA B  TEMP+1      SAVE LO SPACE
70A9 CE 70 C6          LDX    #MESS1      GET ADDRESS OF MESSAGE
70AC BD 72 A6          JSR    ZOUTST      AND OUTPUT
70AF B6 70 EB          LDA A  FCB+2       GET DRIVE NUMBER INTO A
70B2 8A 30             ORA A  #'0         MAKE HEX INTO ASCII
70B4 BD E1 D9          JSR    OUTCH2      AND OUTPUT DRIVE NO.
70B7 CE 70 E3          LDX    #MESS2      LOAD ADDRESS OF MESSAGE 2
70BA BD 72 A6          JSR    ZOUTST      AND OUTPUT
70BD CE 70 E7          LDX    #TEMP       LOAD ADDRESS OF SPACE
70C0 BD 72 AF          JSR    ZOUTHA      AND SHOW THE SPACE AVAILABLE
70C3 7E 72 83          JMP    ZWARMS      AND WARM START DOS
70C6 0D         MESS1  FCB    $D,$A
70C8 53                FCC    'SPACE AVAILABLE ON DRIVE #'
70E2 00                FCB    $0
70E3 20         MESS2  FCC    ' = '
70E6 00                FCB    $0
70E7            TEMP   RMB    2           TEMP STORE FOR SPACE
70E9            FCB    RMB    166
                       END
      NO ERROR(S) DETECTED


   SYMBOL TABLE:
ALLOK   70A3      CON     7096      DFM     7786      FCB     70E9
MESS1   70C6      MESS2   70E3      OUTCH2  E1D9      TEMP    70E7
ZERO    7094      ZGNCHR  7297      ZOUTHA  72AF      ZOUTST  72A6
ZTYPDE  72A9      ZWARMS  7283
```

prtect

I am one of those people who just has to try everything, and when I had unwrapped my 'Air-Freighted across the Atlantic' Smoke Signal BFD-68 Disk system and got it up and running, I tried everything—well, almost!

The thing I could not do was to get two of the error codes —no matter how hard I tried! The error codes are for trying to write to a write-protected file (Error 12) and trying to delete a delete protected file (Error 13). Despite a trans-Atlantic telephone call to a very helpful 'Smoke Signal,' I still couldn't find out EXACTLY how to protect my files. In my 'try everything' fashion I managed it with the tips SSB had given me and the 'PRTECT' program is the result.

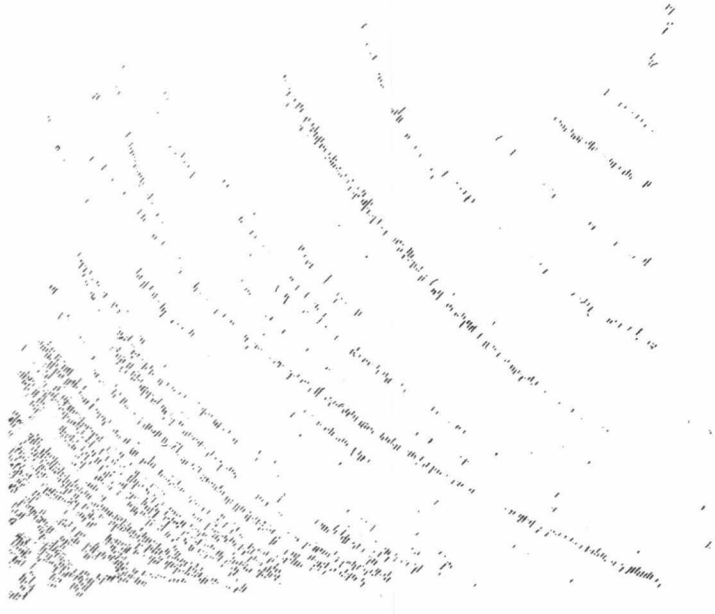
In the 'wee small hours' it saves pulling your hair out as you accidentally delete or write all over your database. It has saved me several embarrassing moments. It does not stop you from formatting all over your Data or Single Sector.

The command is used as follows:—

PRTECT, (filename), (Protection code), ($ if CMD file)

The codes are D=Delete protect. W=Write protect. F= Write & Delete Protect. In order to 'de-protect' just miss out the protection code.

```
                        NAM     PRTECT
                        OPT     NOG
*******************************************************
*
*               PRTECT
*
* THIS IS A TRANSIENT COMMAND FOR USE WITH THE
* 6800 AND SMOKE SIGNAL BFD-68 DISK SYSTEM.
*
* (C) COPYRIGHT 1979 BY LINDSAY REID (ENGLAND)
* ALL RIGHTS RESERVED. NO REPRODUCTION FOR
* COMMERCIAL PURPOSES MAY BE MADE WITHOUT THE
* WRITTEN PERMISSION FROM THE ABOVE AUTHOR.
*
*******************************************************

* EXTERNAL EQUATES IN SSB DOS

7780            ODFM    EQU     $7780       OPEN DFM
72AF            ZOUTHA  EQU     $72AF       OUTPUT HEX ADDRESS
72A6            ZOUTST  EQU     $72A6       OUTPUT STRING TILL A NULL
72B5            ZLINE1  EQU     $72B5       INPUT EDITED LINE FROM BUFFER
7297            ZGNCHR  EQU     $7297       GET NEXT BYTE FROM BUFFER
7291            ZFLSPC  EQU     $7291       GET A FILE SPECIFICATION
7786            DFM     EQU     $7786       CALL DFM
7783            CDFM    EQU     $7783       CLOSE DFM
7283            ZWARMS  EQU     $7283       DOS WARM START
72A9            ZTYPDE  EQU     $72A9       SHOW DISC ERROR
7294            ZGCHAR  EQU     $7294       GET CURRENT BYTE FROM BUFFER

7080                    ORG     $7080       TRANSIENT COMMAND AREA
7080 BD 77 80   INEAR   JSR     ODFM        OPEN DFM
7083 CE 00 01           LDX     #$0001      FIRST DIRECTORY TRACK-SECTOR
7086 FF 71 8F           STX     TRASEC      STORE IN SECTOR POINTER
7089 CE 71 C0           LDX     #XSOD+4     SET POINTER TO FIRST FILENAME
708C FF 71 8C           STX     POINTB      AND STORE IN POINTER B
708F CE 71 92           LDX     #FCB        GET FCB ADDRESS INTO IX
7092 BD 72 91           JSR     ZFLSPC      GET FILENAME FROM BUFFER
7095 CE 71 92           LDX     #FCB        RESTORE FCB ADDRESS
7098 86 12              LDA A   #$12        SET UP FOR SINGLE SECTOR READ
709A A7 00              STA A   0,X         STORE QSSR IN XFC
709C BD 72 97           JSR     ZGNCHR      WHAT COMMAND ?
709F 81 57              CMP A   #'W         WRITE PROTECT ?
70A1 26 04              BNE     DTEST       IF NOT 'W' TEST FOR 'D'
70A3 86 10              LDA A   #%00010000  SET UP WRITE PROTECT
70A5 20 12              BRA     SAVEIT      SAVE MASK
70A7 81 44      DTEST   CMP A   #'D         DELETE PROTECT ?
70A9 26 04              BNE     FTEST       IF NOT 'D' TEST FOR 'F'
70AB 86 20              LDA A   #%00100000  SET UP DELETE PROTECT
70AD 20 0A              BRA     SAVEIT      SAVE MASK
70AF 81 46      FTEST   CMP A   #'F         FULL PROTECT ?
70B1 26 04              BNE     PRTCLR      IF NOT D,W OR F THEN WE CLEAR
70B3 86 30              LDA A   #%00110000  SET UP FULL PROTECT
70B5 20 02              BRA     SAVEIT      SAVE MASK
70B7 86 80      PRTCLR  LDA A   #%10000000  SET UP TO CLEAR PROTECTION
70B9 B7 71 91   SAVEIT  STA A   NEWXFT      STORE MASK
70BC BD 72 94           JSR     ZGCHAR      CHECK FOR NO PARAMETER
70BF 81 2C              CMP A   #',         WAS THERE NO PARAMETER ?
70C1 27 03              BEQ     CLRIT       IF NOT BRANCH TO DE-PROTECT
70C3 BD 72 97           JSR     ZGNCHR      GET NEXT CHARACTER (COMMA ?)
70C6 BD 72 97   CLRIT   JSR     ZGNCHR      GET NEXT CHARACTER ($ ?)
70C9 81 24              CMP A   #'$         IS IT A COMMAND FILE ?
70CB 26 03              BNE     GETSEC      IF NOT - GET SECTOR
70CD B7 71 9B           STA A   XFE         STORE '$' IN XFN EXTENSION
70D0 FE 71 8F   GETSEC  LDX     TRASEC      GET TRACK AND SECTOR
70D3 FF 71 B0           STX     XCT         STORE IN CURRENT TRACK
70D6 C6 05              LDA B   #5          LOAD B WITH COUNT
70D8 F7 71 8E           STA B   COUNT       AND STORE IT
70DB CE 71 92           LDX     #FCB        GET FCB ADDRESS
70DE BD 77 86           JSR     DFM         READ THIS SECTOR
70E1 24 06              BCC     SEARCH      IF NO ERROR BRANCH
70E3 BD 72 A9   ERRJ    JSR     ZTYPDE      SHOW ERROR
70E6 7E 72 83           JMP     ZWARMS      AND WARMSTART DOS

70E9 CE 71 95   SEARCH  LDX     #XFN        POINT TO XFN WE WANT
70EC FF 71 89           STX     POINTA      AND STORE ADDRESS
70EF FE 71 8C           LDX     POINTB      POINT TO FIRST XFN FROM READ
70F2 C6 09              LDA B   #9          LOAD B WITH COUNT
70F4 F7 71 8B           STA B   OFFSET      AND STORE FOR REFERENCE
70F7 A6 00      SRCH2   LDA A   0,X         GET CHARACTER TO MATCH
70F9 FE 71 89           LDX     POINTA      GET BUFFER POINTER
70FC A1 00              CMP A   0,X         DO THEY MATCH ?
70FE 26 12              BNE     NXTFIB      IF NOT - DO THE NEXT FIB
7100 08                 INX                 MOVE TO NEXT CHARACTER
7101 FF 71 89           STX     POINTA      STORE NEW POINTER
7104 FE 71 8C           LDX     POINTB      GET OLD BUFFER PONTER
7107 08                 INX                 AND UPDATE IT
7108 FF 71 8C           STX     POINTB      AND STORE IT
710B 7A 71 8B           DEC     OFFSET      REDUCE OFFSET REFERENCE
710E 27 3F              BEQ     MATCH       IF ALL 9 MATCH - BRANCH
7110 20 E5              BRA     SRCH2       WE HAVE PARTIAL MATCH

7112 7A 71 8E   NXTFIB  DEC     COUNT       REDUCE FIB COUNTER
7115 26 26              BNE     SETFIB      IF BUFFER NOT DONE BRANCH
7117 B6 71 B8           LDA A   XNT         GET ADDRESS OF NEXT DIR BLOCK
711A 27 15              BEQ     NMATCH      IF ZERO FILE NOT FOUND
711C 84 7F              AND A   #$7F        MASK OFF THE 8
711E B7 71 8F           STA A   TRASEC      AND STORE TRACK NUMBER
7121 B6 71 B9           LDA A   XNT+1       GET NEXT SECTOR NUMBER
7124 84 3F              AND A   #$3F        MASK OFF THE 4
7126 B7 71 90           STA A   TRASEC+1    STORE NEXT SECTOR NUMBER
7129 CE 71 C0           LDX     #XSOD+4     GET BUFFER START ADDRESS
712C FF 71 8C           STX     POINTB      STORE ADDRESS IN POINTER B
712F 20 9F              BRA     GETSEC      GO AND GET THE SECTOR

7131 CE 71 75   NMATCH  LDX     #NOFIND     WE DIDNT FIND THE FILE
7134 BD 72 A6           JSR     ZOUTST      OUTPUT THE MESSAGE
7137 BD 77 83   END     JSR     CDFM        CLOSE DFM
713A 7E 72 83           JMP     ZWARMS      AND WARMSTART DOS68

713D B6 71 8B   SETFIB  LDA A   OFFSET      GET OFFSET INTO A
7140 8B 0F              ADD A   #15         ADD MIN COUNT TO NEXT FIB
7142 BB 71 8D           ADD A   POINTB+1    ADD POINTB+1 ADDRESS LO
7145 B7 71 8D           STA A   POINTB+1    AND STORE NEW VALUE
7148 24 9F              BCC     SEARCH      DID WE HAVE A CARRY ?
714A 7C 71 8C           INC     POINTB      IF SO INCREMENT POINTB HI
714D 20 9A              BRA     SEARCH      CARRY ON LOOKING

714F A6 00      MATCH   LDA A   0,X         GET CURRENT XFT INTO A
7151 F6 71 91           LDA B   NEWXFT      GET NEW FILE TYPE WANTED
7154 2A 05              BPL     SETXFT      BRANCH IF NOT TO CLEAR
7156 84 0F              AND A   #$0F        CLEAR PROTECTION
7158 7F 71 91           CLR     NEWXFT      CLEAR NEW FILE TYPE WANTED
715B BA 71 91   SETXFT  ORA A   NEWXFT      SET ONLY NEW PARAMETERS
715E A7 00              STA A   0,X         STORE IN BUFFER
7160 CE 71 92           LDX     #FCB        GET FCB ADDRESS
7163 86 13              LDA A   #19         SET UP FOR QSSW
7165 A7 00              STA A   0,X         STORE IN XFC
7167 BD 77 86           JSR     DFM         CALL DFM
716A 25 06              BCS     JMPERR      IF ERROR THEN BRANCH
716C BD 77 83           JSR     CDFM        CLOSE DFM ALL OK AND DONE
716F 7E 72 83           JMP     ZWARMS      AND WARMSTART BACK TO DOS
7172 7E 70 E3   JMPERR  JMP     ERRJ        JUMP TO DO ERROR
7175 0D         NOFIND  FCB     $D,$A
7177 46                 FCC     'FILE NOT FOUND'
7185 00                 FCB     0
7186 0D         CRLF    FCB     $D,$A,$0

                * VARIABLES AND FCB TABLE *

7189            POINTA  RMB     2           POINTER TO XFN WE WANT
718B            OFFSET  RMB     1           9 MINUS MATCHED BYTES
718C            POINTB  RMB     2           POINTER TO XFN IN BUFFER
718E            COUNT   RMB     1           5 MINUS FIBS WE'VE TRIED
718F            TRASEC  RMB     2           TRACK+SECTOR TO READ OR WRITE
7191            NEWXFT  RMB     1           MASK FOR PROTECTION CHANGE
7192            FCB     RMB     1           FCB STARTS HERE
7193            XES     RMB     1           ERROR CODE
7194            XUN     RMB     1           UNIT NUMBER
7195            XFN     RMB     6           FILENAME (MAIN)
719B            XFE     RMB     3           FILENAME (EXTENSION)
719E            XFT     RMB     1           FILE TYPE
719F            XFS     RMB     1           FILE STATUS
71A0            XFSU    RMB     2           FIRST SECTOR USED
71A2            XLSU    RMB     2           LAST SECTOR USED
71A4            XSUC    RMB     2           SECTORS USED COUNT
71A6            XRFS    RMB     7           RANDOM FILE SIZE + 4 RESERVED
71AD            XNFP    RMB     2           NEXT FCB ACTIVE
71AF            XBI     RMB     1           INDEX INTO BUFFER
71B0            XCT     RMB     1           CURRENT TRACK
71B1            XCS     RMB     7           CURRENT SECTOR + 6 RESERVE
71B8            XNT     RMB     1           NEXT TRACK
71B9            XNS     RMB     1           NEXT SECTOR
71BA            XPT     RMB     1           PREVIOUS TRACK
71BB            XPS     RMB     1           PREVIOUS SECTOR
71BC            XSOD    EQU     *           START OF DATA BUFFER
                        END
      NO ERROR(S) DETECTED


   SYMBOL TABLE:
CDFM    7783    CLRIT   70C6    COUNT   718E    CRLF    7186
DFM     7786    DTEST   70A7    END     7137    ERRJ    70E3
FCB     7192    FTEST   70AF    GETSEC  70D0    INEAR   7080
JMPERR  7172    MATCH   714F    NEWXFT  7191    NMATCH  7131
NOFIND  7175    NXTFIB  7112    ODFM    7780    OFFSET  718B
POINTA  7189    POINTB  718C    PRTCLR  70B7    SAVEIT  70B9
SEARCH  70E9    SETFIB  713D    SETXFT  715B    SRCH2   70F7
TRASEC  718F    XBI     71AF    XCS     71B1    XCT     71B0
XES     7193    XFE     719B    XFN     7195    XFS     719F
XFSU    71A0    XFT     719E    XLSU    71A2    XNFP    71AD
XNS     71B9    XNT     71B8    XPS     71BB    XPT     71BA
XRFS    71A6    XSOD    71BC    XSUC    71A4    XUN     7194
ZFLSPC  7291    ZGCHAR  7294    ZGNCHR  7297    ZLINE1  72B5
ZOUTHA  72AF    ZOUTST  72A6    ZTYPDE  72A9    ZWARMS  7283
```



Letters

HAIR TODAY, GONE TOMORROW

Dear Suzanne,

Thank you for mailing me some copies of the September issue, containing an article by myself (A General Purpose Data Compression Program, *DDJ* #38, page 16). Upon reading that article, I was indeed delighted to discover (from my photograph) that your readers will perceive me as having somewhat more hair than my real self. However, the gentleman whose actual visage matches that in the photograph may be somewhat less than delighted!

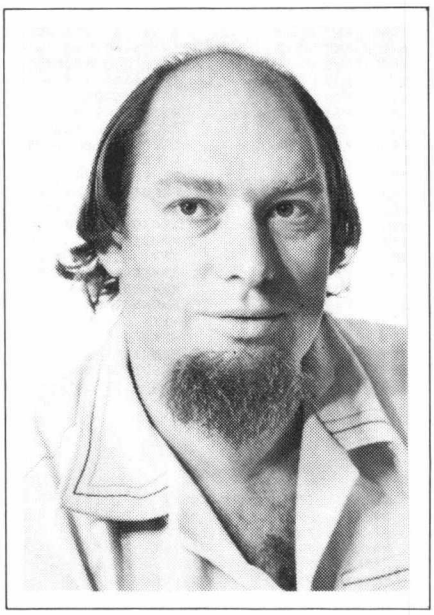
I am forwarding herewith another photograph in case any of your readers are especially interested (heaven help them!) in my appearance.

Yours faithfully,
Graham K. Jenkins.

Telecom Australia
Research Laboratories
762-772 Blackburn Rd., Clayton North,
Melbourne Vic. 3168

Thank God most of our mistakes around here are fit subjects for laughter. For those readers especially interested – and to restore Mr. Jenkins' feeling of self-identity – here is a picture of the real Graham Jenkins.

The guy with the thick blond hair is L.H. Reid, whose article (and, again, whose picture) appears on page 53. – SMR



SAVE THE STATE

Editor,

Here is my entry for the shortest routine to save the state of an 8080:

HLT

Seriously, saving the state of an 8080 is really more difficult than most programmers realize. A common mistake is to use a DAD instruction before saving the PSW. The DAD alters the Carry flag, and on Z80's and some 8080's, it also alters the Subtract flag. Usually, the PSW is PUSHed on the stack before DADing but this assumes a valid Stack pointer. When the machine state is being saved by an interrupt routine, monitor call or other general subroutine, the Stack Pointer validity can be assumed. However, break-point and debugger routines should avoid using the stack since it may or may not be valid, i.e., pointing to read only, write protected, disabled or undefined memory addresses.

Here's the challenge: Can an 8080 program be written to save the entire machine state without referencing any memory location not defined in the program; Furthermore, except for the register save area, the program should be ROMable, it should work with any 8080 and leave the state of the machine unaltered upon exiting.

Hint; central to the problem is testing the 'non-testable' flags in the PSW. I'll submit my solution next month. If anyone can do it in less than 58 bytes, please let me know.

Cheers,
Jon Bokelman

P.O. Box 1057
Menlo Park, CA94025

ERRATUM

Dear Ms. Rodriguez:

I have just discovered an error in line numbers in the note at the conclusion of *Main Software – Teletype Terminal Interface* (*DDJ* # 38). The note on the last page of the article gave the line numbers within the routine containing the baud-rate timing constants. Line numbers 560, 570, 800 and 810 *should have been* 360, 370, 600 and 610 respectively.

I regret this error.

Sincerely,
Donald E. Faulkner

1245 Springdale Drive
Jackson, Mississippi 39211

CONSERVING MEMORY WITH THE HUFFMAN CODE

Dear *Dr. Dobb's*:

If you really want to conserve memory . . . here are some ideas borrowed from Information Theory to allow English language text to be stored very efficiently. It is an alternative to the word-storage schemes proposed by Klaus Holtz in *DDJ* #33 and by Gordon Wilk in his *DDJ* #36 letter. It's called a Huffman code.

I'll describe the encoding method on a small, hypothetical alphabet rather than work with the entire ASCII set.

Suppose we wanted to store some text in the FERPOE language, which uses only the letters A, B, C, D, and E. We know that standard FERPOE contains the following distribution of letter frequences:

Letter	Probability
A	.37
B	.25
C	.20
D	.11
E	.07

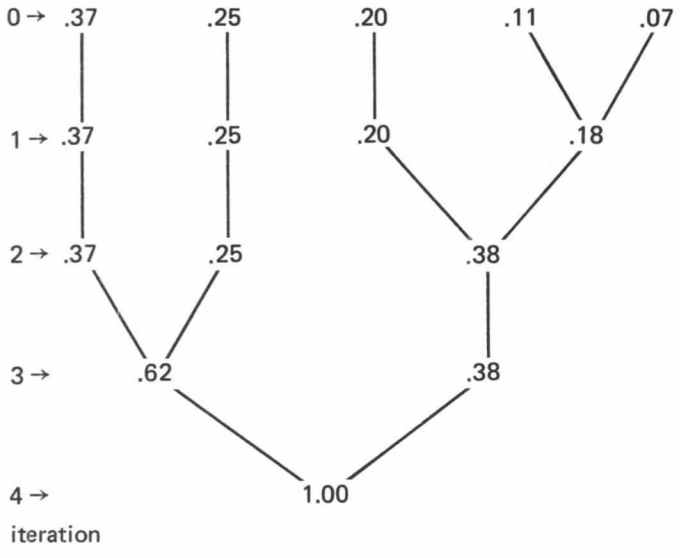
Instead of devoting eight or even three bits to each of these characters in our computer encoding, it is more efficient to use fewer bits on the more frequent characters (e.g., A). Morse code does this.

But how do you decide how many bits to use for each character? There's a simple algorithm which finds the best combination:

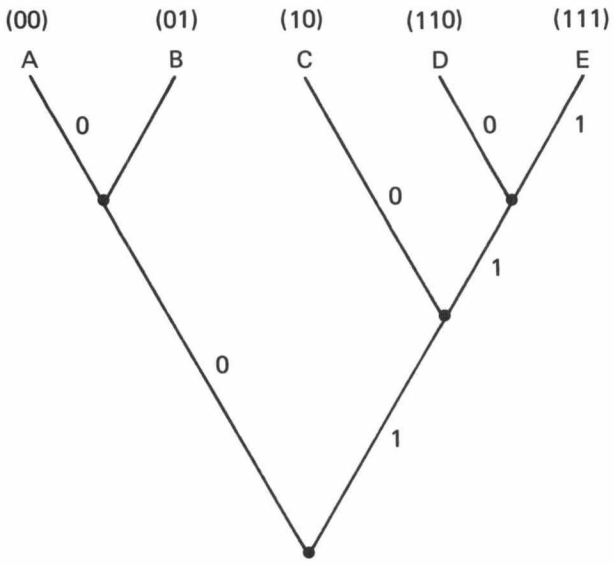
You write down the probabilities in a row, and grow a tree by performing the following iteration until there is only one number in your current row.

Iteration: copy the previous row, only remove the two smallest numbers from the row, and write their sum in their place.

For our FERPOE example, the tree looks like this:



Then we redraw the tree, labeling left branches with 0 and right branches with 1. Now, if you trace the branches of the tree from the bottom up, the sequence of bits is our encoding for the terminal character.

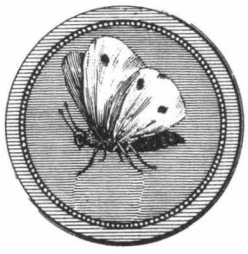


This is indeed better than a 3-bit code, since the average character will take .37(2) +.25(2)+ .20(2)+ .11(3)+ .07(3) = 2.18 bits.

To apply this method to English, you would need to find accurate statistics on the character frequencies in your specific kind of text. This scheme has the disadvantage that decoding (retrieval) involves bit operations, but its efficiency may come in handy someday when California suffers a memory shortage. Who knows? Perhaps the word processors of the future will do this encoding and decoding in hardware.

Sincerely,
Paul Heckbert

4 Ames St., H305
Cambridge, MA 02139



A CORRECTION

Dear *Doc*:

Alex Caemmerer's routine to save the 8080/280 machine status (*DDJ* #38, page 40) is flawed. the DAD instruction clears the carry flag before the PSW gets pushed.

Sincerely,
James Monagan

806 Clark Street
Iowa City, Iowa 52240

MORE 8080 RELATIVE ADDRESSING

Dear Suzanne,

The article by John Beetem on relative addressing for the 8080 was interesting. It occurred to me that this circuit would probably fail-safe if M1 and M2 were ANDed and then ORed with POC. Also, shouldn't the clear lines for IC1 and IC2 be connected to Y2 as well as the clock lines? This line goes up and is marked one, but is discontinued.

This circuit might also be used with a 6502 processor, but don't use it for a 680x processor, or an 1802. The 6800 won't like it because of fetching the high order byte first, and other considerations; and about the 1802: WHAT FOR?

It also might be used with a Z80 on the added instructions if additional circuitry is added to hold it in state one after an indexing prefix or the prefix for double add and subtract and other various instructions (DDH, EDH, FDH). This will allow loading BC, DE, HL, SP, IX and IY with a relative address, from a relative address, or storing them to a relative address. With the power offered by the Z80 coupled with true relative addressing this could be a powerful combination.

Please note that there is an extra instruction that is relative: POP! This includes RET (POP PC). What can you do with it? I don't know, but . . . .

Perhaps it isn't worth my time, but WHY bother with 8080 style mnemonics for the Z80? Although there are a lot of instructions in the expanded set, at least with Zilog you only need to know one data transfer mnemonic: LD. Ditto JMP, CALL, RET, ADD, ADC, SBC, etc.

Thanks to everybody at *DDJ*, and to everybody else that is committed to making computers easier and more natural to use rather than trying to change people into computers.

Sincerely,
John P. Morgan

P.O. Box 14002
Spokane, WA 99214

NOTHING WRONG WITH ANONYMOUS, WE THINK

"Anonymous" from Japan commented on page 45 of issue #37. This reader ought to see page 1059 of the *Communications of the ACM*, December 1972, Volume 15, #12. It seems to be a recurring phenomenon of the computer field that the many newcomers to the field repeat the thought patterns which were well-worn ten or twenty or even thirty years ago.

T.L. Gerber
Computer Place, Inc.

419 E. Railroad Avenue
Aberdeen, S.D. 57401

You sound a little intolerant. Maybe Anonymous never read the dialogue you refer to; maybe Anonymous is just getting into computers and his questions, while not new to you, are original and new to him. University students studying philosophy for the first time discover – without reading Plato – concepts Plato delineated a couple of thousand years ago, and think they have stumbled on to new truths about life. This fact neither invalidates the freshness of their thinking or the timelessness of Plato's. Give the guy a break. – *SMR*

Index

Index

This is an index to articles and programs, but some letters have also been indexed. References are of the form *issue number/page numbers.* Those in normal type are references to programs. References separated by a comma are connected in some way; those separated by a semicolon are completely distinct. Unless otherwise marked, a program is in assembly language. Except for author entries the processor type is marked or clear from the context. In general, a program can be found by author, processor type, and program type.

The following table gives the correspondence between month/year and issue numbers.

Vol 4 – 1979

January	31
February	32
March	33
April	34
May	35
June/July	36
August	37
September	38
October	39
November/December	40

1802
- Floating-point arithmetic 37/17-19L

650X
- Line-file editor 33/47-49L

6500
- EXDOS 37/20-26L
- EXOS operating system 36/29-31
- Patch to OSI-resident editor 34/40L
- Running OSI-BASIC 34/37-39L

6502
- Cassette loader for OSI tapes 32/43L
- Compared to 6800 39/38-39
- Opcode trap, hardware mod 31/32-33
- Random number generator 32/14-17 + L; 40/33-35 + L

6800
- Article on 38/39-39
- Compared to 6500 38/38-39
- Debugger 35/40-43 + L
- Disk directory program 33/24L
- Disk drivers, for LISP 34/35-36 + L; addenda 37/39L
- Diskutilities 40/53-55L
- File dumper under FLEX 40/36-37L
- Mods to SWTPC BASIC 35/25L
- Mods to the SWTPC-resident editor/assembler 35/28 errata 37/43
- PIA's for, hardware mod. 33/41-42
- PROM Programmer 32/34-35 + L
- Push/pull registers 33/45; 37/27L; 37/39-40L
- Speech synthesis 32/18-19L
- Telecommunications program 39/30-31L
- Word processor 34/4-6L

8080
- Conversion (PATB to ASCII) 33/15-23L
- Conversion (Pencil CPM) 33/26-29L
- Dazzler, utility 31/45L
- Disassembler 40/42-45L [BASIC]
- Diskette copy 37/28-29L
- File dumper 38/26-29L
- Floating-point arithmetic 33/4-14 + L; 34/16-25L
- Formatter 35/4-14L
- Hardware mod, relative addressing 37/30-34
- Memory clear 34/45-46L
- Mod to 8080 PILOT 40/4-17L
- Mod to Poly 88 to load several programs at once 34/40 + L
- Mod to POW 32/42-43L
- Mod to vocal memory dump 31/44-45L
- Operating system (Chaos) 31/16-13
- Programming trick 31/43L
- Push/pull registers 33/45; 35/49-50L; 38/40-41L; 38/42L; 40/56-58
- Simple linear search 36/49-50L
- Telecommunications program 40/46-52L
- Tiny-C on CDOS 35/37-39L
- UCSD PASCAL, file convert 34/26-27L; 37/12-16L

8085
Unspecified opcodes 34/28-29

Ackermann's Function 31/23; 33/46; 35/48-49; 36/45-46; 37/10-11; 37/43; 39/40
Albrecht, Bob 34/32-34
Alkire, Robert 31/40-41L
Allison, Dennis 34/32-34
Apple-II
- Hardware mod 32/40

Apple-II, BASIC
- mod to load multiple programs 38/37-38L [6502]
- page lister 39/19-21L [6502]

Arithmetic
floating-point 33/4-14 + L [8080] ; 34/16-25L [8080]
Assemblers
Aid to hand-assembly 38/24-25

Bakilinsky, Rick 34/32-34
Barker, L. 33/47-49L
BASIC
8080 disassembler 40/42-45L
F8 tiny 39/4-18L [f8]
Heathkit 32/41L [8080]
Palo Alto (convert files to ASCII) 33/15-33L [8080]
PET (renumberer) 31/24-27L [6502] ; mod 36/47
Preprocessor for 33/40+
Star maps plots 31/4-13L
Beck, Douglas K. 34/41-42
Beetem, John 37/30-34
Bender, A.L. 38/26-29L
Bibliography
on PASCAL 32/29-30
on tiny-C 38/39-40
Binary tree routines 36/37-41 + L [Tiny C]
Blue Sky 32/20-28+; 33/30-37
Borochoff, Ronald T. 40/42-45L
Bowden, Jay C. 39/26-29
Burris, David S. 33/25
Burt, Robert 32/4-13L
Bus standards 38/30-33
Business software 36/24-27; 31/36-37L [BASIC]
Byrd, David S. 33/25
Byrd, Wayne 33/25

Carpenter, Chuck 32/40
Chaos (operating system 31/6-13
Chapman, David 32/20-28+
Colburn, Don 36/29-31; 37/20-26L
Coleman, Roger 31/44-45L
Collis, Paul 37/28-29L
Computer music 36/33-36
COSMAC 36/28+
CPM (converting files to Pencil Format) 33/26-29L [8080] ;
errata 39/40
Tiny-C
Bibliography on 38/39-46
Binary tree routines 36/37-41 + L
Interface to CDOS 35/37-39L [8080]

DAL (Data acquisition language) 40/26-32
Daly, Ray 31/45L [8080]
Data compression 38/16-18 + L [BASIC]
Dazzler
Misc. 31/44-45L [8080]
Utilities 31/45L [8080]
Debugger, 35/40-43 + L
Dernhardt, Wolfgang 34/28-29
Derynk, Ronald 31/20-22L
Digitizing photos 39/26-29
Disassembler 35/16-24L [2-80] ; 40/42-45L
[BASIC for 8080] ; errata 37/46L; 38/34-36L
Duncan, Ray 35/37-39L

Editor
line-file (for 650X) 33/47-49L [650X]
Educational systems design 36/8-23
EXPOS (operating system) 36/29-31; 37/20-26L [650X]

F8
Tiny BASIC for 39/4-18L
Falconer, Charles B. 33/4-14 + 4; 34/16-25L; 38/30-33
Faulkner, Donald 39/34-37L
Felsenstein, Lee 40/38-41
File dumper 38/26-29L [8080] ; 40/36-37L [6800]
Fogg, David M. 31/28-31 + L
FORTH 31/45-47; *see also* STOIC
Formatter 35/4-14L [8080]
Fox, Jerry D. 39/4-18L

Gabrielson, Mike 31/23; 32/29-30; 32/40;
34/26-27L; 35/4-14L
Game
Mastermind (for SC/MP) 31/34-35 + L
Misc. 31/4-5; 35/26-27
Gagne, Jim 33/26-29L
Gaugler, Gary 32/18-19L; 33/24 + L; 34/4-16L; 35/40-43+L
Gass, Geoffery A.
GENIE 32/20-28+
Gordon, J.T. 32/14-17th; 40/33-35 + L

Hancock, Les 36/37-41 + L
Hardware mod
6502 opcode trap 31/32-33+
6800 speech synthesis 32/18-19L
Apple-II 32/40
Converting 8-bit memories to 16-bit 34/30-31
Interface H-14 to SOL 39/42-43
PIA's to a KIM 33/41-42
Relative addressing for 8080 37/30-34; addenda 40/58
Sound synthesis 36/32
Telephone dialer 38/15L [ZIBL]
To North Star Horizon for disks 37/44
To Z-80 board for 2716 35/50; 39/42
Holliday, Paul 33/15-23L
Holtz, Klaus 33/30-37
Hudson, Larry 34/45-46L
Humor 37/40
Hypertext 32/20-28+

Ingraham, Donald J. 40/18-19+
Insam, E. 32/34-35 + L
Interface
TTY 39/34-37L [2-80] ; errata 40/56
Interview
On the origins of DDJ 34/32-34

Jenkins, Graham K. 38/16-18 + L
Johnson, David L. 34/40 + L

Kapps, Charles 31/34-35 + L
Keay, Colin 38/24-25
KIM *see* 6800

Language design 37/44-45
Law
Computer crime bill 5240 37/4-9; 40/18-19+
Lee, Bock W. 31/20-22L; 35/16-24L; 40/20-25
Levinsky, Jeff 31/6-3
LISP
Diskdrivers for 6800 34/35-36 + L
Note on portable interpreter 32/42
Utilities 38/19-23L
Listings, paged (on Heathkit) 32/41L [8080]
Long, R.J. 32/31-33 + L; errata 34/44
Lyons, George 34/30-31

Main, Richard B. 31/45-47
Marshall, Gregg 40/26-32
McKeeman, W.M. 31/38-39L
Memory clear 34/45-46L [8080]
Memory mapper [Z-80] 31/40-41L
Memory tester [Z-80] 31/20-22L
Mersenne primes 36/4-6
Miscellaneous comments
To computer control of music tapes 33/46-47
To Klaus Holtz 36/47-48
To *The STD Bus* 34/43-44
Miscellaneous errata
Disassembler (in 27/4-18) 31/43-44
LISP (in 30/4-11) 36/47; (in 28/24-24) 31/44
Power function (in 29/14) 31/43 + L [BASIC]
Programming trick (in 29/13) 31/43L [8080]
To SYM-1 37/46
Miscellaneous mods
OSI-resident editor 34/46L
POW (In 29/18-29) 32/42-43L [8080]
SWTPC BASIC 35/25; 34/45; 36/48-49L [6800]
To North Star DOS 32/43L
To Poly 88 multiple file loading 34/40 + L [8080]
To TDL software 32/31-33 + L [Z-80]; errata 34/44
Vocal memory dump (in 28/42-46) 31/44-45L [8080]
Miscellaneous software
Report generator 31/38-39L [Zilog System]
Moore, Freeman L. 40/36-37L
Morganstern, David 39/30-31L
Morgenstern, Leonard 32/42-43L
Moser, Carl W. 31/32-33+
Mosgofian, Karl 36/28+
Moss, William W. 32/41L

Neivergelt, J. et. al. 36/8-23
Noll, Curt 36/4-6
North Star Horizon
Hardware mod for disks 37/44
DOS, mod to div list 39/43

Olsen, S. Cash 38/15L
Optical character recognition 38/39

Parsons, Ronald G. 37/12-166
PASCAL
Bibliography on 32/29-30
UCSA, file conversion 34/26-27L; 37/12-16L [8080]
Pencil, electric (converting file to CPM) 33/26-29L [8080]; errata 39/40
Pet 39/22-23
Petersen, Holger 38/34-36L
Phelan, Charles M. 36/32
Photos, digitizing 39/26-29
PILOT (8080)
Mods to 40/4-17L [PILOT]
Pittman, Tom 35/44-45
Product evaluation
PERCOM LFD-400 34/41-42
Programming Pastimes & Pleasures 31/4-5; 32/36-37; 35/26-27; 36/44+; 37/10-11; 39/32-33+
PROM Programming 32/34 - 35 + L [6800]

Rabin, Sylvan 33/40+
Random number generator 32/14-17 + L [650X]; 40/33-35 + L [650X]

Ratliff, Gary 39/22-23
Renumberer (for Pet BASIC) 31/24-27L [6502]; mod 36/47
Reynolds, D.J. 38/4-14
Rice, Lloyd 36/45-46
Roads, Curtis 33/38-39; 36/33-36
Robotics 38/4-14

SAM76
Update 31/16-19
Utilities 31/28-31 + L
SC/MP
Game, Mastermind 31/34-35 + L
Schaefer, K. Dieter 34/43-44
Scharschmidt, Anna K. 39/26-29
Scott, Greg B. 36/24-27
Scully, Tim 40/4-1
Seiler, Bill 31/24/27L
Simons, Kenneth A. 31/14-15
Software copyright 37/41-42
Software standards 35/44-45
Sorensen, Villy M. 34/28-29
Sorting 33/35
Sound synthesis 36/32
Speech synthesis
Hardware & software 32/18-19L [6800]
Standards
8080 compatibility 38/41
Star map plots 32/4-13L [BASIC]
STOIC 40/20-25; *see also* FORTH
Story, Glenn 40/46-52L
Swank, Joel 33/41-42
Sweet-16
Assembler for 39/24-25
SWTPC resident editor/assembler
Mod to 35/28-26L [6800]; errata 37/43

Taber, Jon. K. 37/4-9; 40/18-19+
Tan, B.T.G. 39/38-39
Tarvydas, Paul 38/19-23L
TDL Software
Modifications 32/31-33 + L [Z-80]
Telecommunications program 39/30-31L [650X]; 40/46-52L [8080]
Tiny C *see* C
Trombetta, Michael 31/36-37
TRS-80
Direct access to keyboard 36/50; 39/41
TUSIX 31/14-15
TUFIVE 31/14-15

VDM-2 40/38-41

Wasserman, Judith 32/4-13L
Wetherell, Charles 31/4-5; 32/36-37; 35/26-27; 36/44+; 37/10-11; 39/32-33+
Wheeler, Steve 39/24-25
Wordprocessor 34/4-16L [6800]
WP-68 34/4-16L [8080]

XS-0 36/8-23

Z-80
Disassembler 35/16-24K. errata 37/46L; 38/34-36L
Hardware mod to board 35/50; 39/42
Memory mapper 31/40-41L
Memory tester 31/20-22L
Mods to TDL software 32/31-33 + L; errata 34/44
TTY interface 39/34-37L; errata 40/56
Zamora, Ramon 32/4-13L
Zant, R.L. 38/37L; errata 40/56
ZIBL (Dynabyte Z-80 industrial BASIC language) 38/15L

5491-2
$20.75

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

ISBN 0-8104-5491-2