Dr. Dobb's Journal of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

A Reference Journal for Users of Home Computers

Volume Five

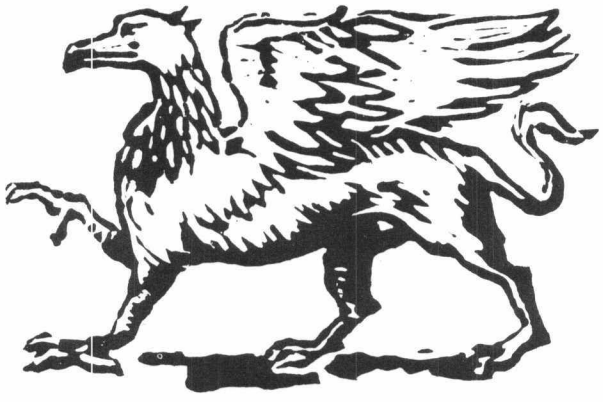
The People's Computer Company Series

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

A Reference Journal for Users of Home Computers

Volume Five



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 0-8104-5492-0

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

Dr. Dobb's Journal

of Computer Calisthenics & Orthodontia,

Volume Five

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, techniques 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 5, NUMBER 41, 1980

2 The History of CP/M Gary A. Kildall
4 An Overview of CP/M Compatible Software Steve Newberry
11 Go: A Utility Command Under CP/M Anthony Skjellum
13 Linking CP/M to George Morrow's Cassette Interface Wayne Miller
15 Implementing a Tiny Interpreter with a CP/M Flavored C Les Hancock
24 ALS-8 to CP/M File Converter Hollis Frampton
26 Converter Listing
29 CP/M and CDOS Program Compatibility Jim Warner
32 Changing SPI Formatted Programs to CP/M Compatible Text Mark M. Zeiger
34 CP/M-Pen, Pen-CP/M Rod Hallen
37 Programming Pastimes and Pleasures Charles Wetherell

VOLUME 5, NUMBER 42, 1980

46 TI 9900 Fully Reentrant Formatter Operating Under TIPMX Norman V. Walters
55 North Star Disk STOIC
56 Feature Enhancements for ADM-3A John Wakerly
59 A Table-Driven Assembler on CP/M Darrel J. Van Buer
67 A User Interface to APPLE-II Program Renumbering DeWitt S. Brown
73 Converting Binary to Decimal Dennis Allison
75 UCSD PASCAL Terminal Parameters Ronald G. Parsons
77 Programming Pastimes and Pleasures Charles Wetherell
79 A Microprocessor-Assisted Graphics System Griffith Hamlin, Jr. and Thomas Crockett

VOLUME 5, NUMBER 43, 1980

88 Mathematical Typography Donald E. Knuth
104 SYMBOL: Generating Symbol Table Alpha Listing on the H8 William W. Moss
108 TI9900: Fully Reentrant Formatter Operating Under TIPMX, Part 2
Norman V. Walters
121 Can You Afford to Retire? David E. Raeside
122 A Hex Keyboard with Applications for the 1802 ELF Martin Bronstein

VOLUME 5, NUMBER 44, 1980

130 An Introduction to Algorithm Design Jon Louis Bentley
141 All the Secrets You Ever Wanted About Poly 88 BASIC Marvin A. Konopik
142 Meta II: A Syntax-Oriented Compiler Writing Language D.V. Schorre
151 A Northstar BASIC Cross-Reference Tool Titus Purdin
159 Some Useful Subroutines for a Z-80 Assembler John W. Carter
164 A Fast Fourier Transform for the 8080 Robert D. Fusfeld
169 Algorithm: A Monthly Algorithm Column Dennis Allison

VOLUME 5, NUMBER 45, 1980

176 A Small C Compiler for the 8080's Ron Cain
191 The C Programming Language D. M. Ritchie, S. C. Johnson, M. E. Lesk, B. W. Kernighan
201 Structured Programming, C and tiny c Tom Gibson and Scott Guthery
205 Help with OSI's CP/M Lee Barker
207 Algorithm: A Monthly Algorithm Column Dennis Allison
209 MITS DOS – Disk to Disk Copy Routine Andrew Bender
216 An 8080 to Z80 Translator System Robert W. Dea

VOLUME 5, NUMBER 46, 1980

226 A Survey of Computer Networks Art Kleiner
230 The Personal Computer Network (PCNET) Project Dave Caulkins
234 MCALL-C: A Communications Protocol for Personal Computers Tim J. Pugh, Jr.
239 Electric Phone Book
240 On Modems William Smith
242 Adding Arrays to FORTH Ralph Deane
243 ROM Byte-Finder for the Apple II Stephen E. Bach
244 The Errata of Computer Programming Donald E. Knuth
257 Random Ruminations of Networks of the Future Jonathan Sachs
259 Displaying 2 Columns with North Star O. K. Hudson
260 A Versatile Memory Dump in BASIC Hal Knippenberg
261 Conversion: ALS-8 to CP/M Edward Elizondo
264 Algorithm: A Monthly Algorithm Column Dennis Allison

VOLUME 5, NUMBER 47, 1980

272 ZX65: Simulating a Micro Richard M. Kruse
279 Harlie: A "No Programming" Computer J. L. Lawrence and Jay Sprenkle
282 Clean Starts for COSMAC 1861 Video Output Gary H. Price
284 PCNET Notebook Dave Caulkins
285 Programming Pastimes and Pleasures Charles Wetherell
287 Z-80 Circuit for the H-8 Ted T. Stowe
289 Indirect Addressing for the 8080 and Z-80 Anthony Skjellum

VOLUME 5, NUMBER 48, 1980

312 A Runtime Library for the Small c Compiler Ron Cain
324 Thoughts on Small Systems and Monitors H. T. Gordon
330 Programming Pastimes and Pleasures Charles Wetherell
332 Polymorphic Talks to the H-14 Printer Marvin Konopik
333 A Note on 6502 Indirect Addressing W. D. Maurer
339 Northstar X-Reference Tool John Holtrichter, Jr.
341 Hardcopy Device Driver Programs for CP/M Steve Willoughby
345 The Love Program Gary Gaugler
348 A Problem with Logic Probes David N. Bertollo
349 An Error-Correcting Record Linking Method Sylvan Rubin
350 Algorithm: A Monthly Algorithm Column Dennis Allison

VOLUME 5, NUMBER 49, 1980

356 Even Further with Tiny BASIC Robert Hudson
364 Microcomputers as They Should Be Olav Naess
367 CP/M to UCSD PASCAL File Conversion Tor Sjøwall
371 Extended Source-Text Area and Label Capability for SC/MP NIBASM W. F. Long
373 Opcode Design: 9 or 16 Bits? H. T. Gordon
375 MFT: A Multiple File Transfer Utility Lawrence E. Hughes
382 A Cross Reference Program for Northstar BASIC Larry Hudson
392 Solution of Three First Order Differential Equations with an SR-52 Programmable Calculator S. Andrew Yakush

VOLUME 5, NUMBER 50, 1980

400 N-Logs: A New Number Language for Scientific Computers Keneth A. Simons
406 Las Vegas Super Slot: A CP/M Game Machine Program Using Flashwriter I Graphics David W. Friedman
418 Quick-Key: A North Star BASIC Statement Generator Leonard E. Garcia
423 TUTOR: A Foreign Language Vocabulary Trainer Gary D. Gaugler
425 Interfacing the WH14 Printer to Your Microcomputer's Serial Port Is Simpler Than You Think Lachman Sippy
426 Program Documentation Freeman L. Moore
431 MOVIT William D. Loughman
432 6800 and 1802 Cross-Assembler for CP/M William C. Colley, III
434 A Proposal on Strings for FORTH Ralph Deane
439 A 6800 Video Display Program Kenneth Drexler
447 Index

About People's Computer Company

People's Computer Company, the publisher of *Dr. Dobb's Journal*, was founded in 1972 as a non-profit source of education and information about computers. Bob Albrecht, one of the founders, explained the inspiration for the company in the first issue of the *PCC Newspaper*:

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

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

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

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

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

To make these ideals manifest, it has been necessary to think far ahead, to be creative and innovative. PCC published the first personal computing periodical in 1972, a time when the technical community at large found the thought of "personal" computers preposterous. Shortly thereafter, the company produced the world's first book of computer games. Now in its sixth printing, *What to Do After You Hit Return* has been a model for many other publishers.

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

People's Computer Company
Box E
Menlo Park, CA 94025

Dr. Dobb's Journal of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 41 | January 1980 | Volume 5, Issue 1

A REFERENCE JOURNAL FOR USERS OF SMALL COMPUTERS

All-CP/M Issue!

The Evolution of an Industry: One Person's Viewpoint 6
Gary A Kildall

Overview of CP/M Compatible Software 8
Steve Newberry

Implementing a Tiny Interpreter with a CP/M-Flavored C 20
Les Hancock

CP/M-Pen, Pen-CP/M 39
Rod Hallen

GO: A Utility Command Under CP/M 16
Anthony Skjellum

CP/M and CDOS Program Compatibility 34
Jim Warner

... and, as always, even more

THE EVOLUTION OF AN INDUSTRY: ONE PERSON'S VIEWPOINT

The History of CP/M

BY GARY A. KILDALL
Digital Research
P.O. Box 579
Pacific Grove, California 93950

1973. . . .

I was sitting quietly at my desk when Masatoshi Shima hurried into my office at Intel and asked me to follow him to his laboratory down the hall. In the middle of his work bench, among the typical snaggle of jumpers, oscilloscopes and multimeters, sat a binocular microscope with spider-leg probes, all of which were subjecting a minute piece of silicon to helpless investigation. I peered through the microscope at the enlarged regular patterns with particular interest. As a consultant, my job was to design and develop certain software tools for Intel. One was Interp/80, a program which simulated Intel's newly evolved 8080 microprocessor to be used by Intel customers on timesharing systems. As I searched for something recognizable, I hoped my simulation resembled the operation of Shima's first 8080 chip which had finally come to life.

My proposal to Intel had been simple: I would provide them with a language, called PL/M, to replace serious systems programming in assembly language. The compiler would first be written in FORTRAN for operation on timesharing computers and "cross compile" to the eight-bit processors. Next, we would write a PL/M compiler in PL/M and "bootstrap" from the timesharing computer to a resident compiler operating on Intel's new Intellec-8 development system. The first part was complete. PL/M cross compilers and Interp simulators were implemented for the now-best-forgotten 8008, as well as the 8080. Programs had been written and tested by Intel's software group, consisting of myself and two other people, and we were ready for the real machine. Things were going well: the resident compiler would be the next step.

Unfortunately, nearly all small computer systems in 1973 used paper tape as the backup storage device, with the ubiquitous model 33 Teletype serving as the nerve-shattering I/O device. It was readily apparent that resident development systems could not compete with timesharing services when considering throughput, resources, and services. Still the notion of a personal computer for software development interested everyone.

I became intrigued with a new device, called a floppy disk, which, though designed by IBM to replace punched cards, appeared to have much greater potential. The device was ideal: over 3,000 times the data rate of a Teletype, each $7 diskette could randomly access the equivalent of 2000 feet of paper tape. Best of all, the drive was priced at a low $500. Due to a slight problem of undercapitalization, I found this incredibly low price still a bit high. At that time, a smallish company called Shugart Associates was in operation a few miles up the road from Intel. Dave Scott, then marketing manager at Shugart Associates, donated one of their 10,000-hour test drives to the cause, complete with worn-out bearings and a bearing repair kit. It was only later, as I sat in my office at home, staring at the naked disk drive, that I realized I had no cabinet, no cables, no power supplies, no controller, and most distressing of all, no hardware design experience. To make matters worse, no controllers were commercially available, even if I could afford one.

After several abortive attempts at constructing an interface to my Intellec-8, it became readily apparent that my efforts would be better directed toward the software aspects.

Between projects I put together the first CP/M file system, designed to support a resident PL/M compiler. The timesharing version of PL/M, along with the Interp simulator, allowed me to develop and checkout the various file operations to the level of primitive disk I/O. A simulation is, after all, just a simulation, and the inability to make that 10,000 hour drive work for just one more hour was frustrating.

Shortly thereafter, in the fall of 1974, John Torode became interested in the project. I offered as much moral support as possible while John worked through the aberrations of the IBM standard to complete one of my aborted controllers. Our first controller was a beautiful rat's nest of wirewraps, boards and cables (well, at least it was beautiful to us!) which, by good fortune, often performed seeks, reads, and writes just as requested. For agonizing minutes, we loaded the CP/M machine code through the paper tape reader into the Intellec-8 memory. To our amazement, the disk system went through its initialization and printed the CP/M prompt at the Teletype.

Anyone who has brought-up CP/M on a homebuilt computer has felt this moment of elation. A myriad of connections are properly closed, bits are flying at lightning speeds over busses and through circuits and program logic, to produce a single prompt. In comparison to our paper tape devices, we had the power of a S/370 at our fingertips. A few nervous tests confirmed that all was working properly, so we retired for the evening to take on the simpler task of emptying a jug of not-so-good red wine while reconstructing battles, and speculating on the future of our new software tool.

In the months that followed, CP/M evolved rather slowly. Intel was experiencing enormous growth and all software development was halted while new mangement structures were instituted. Intel expressed no interest in CP/M, nor in continuing any resident compiler work. Nearly two years passed before Intel again took interest in resident software tools, with their introduction of the ISIS operating system and later, the resident PL/M-80 compiler.

Meanwhile, John Torode redesigned and refined our original controller and produced his first complete computer system, marketed under his company name, Digital Systems (which later became Digital Microsystems). The first commercial licensing of CP/M took place in 1975 with contracts between Digital Sytems and Omron of America for use in their intelligent terminal, and with Lawrence Livermore Laboratories where CP/M was used to monitor programs in the Octopus network. Little attention was paid to CP/M for about a year. In my spare time, I worked to improve overall facilities, and added an editor, assembler, and debugger which were predecessors of the current ED, ASM, and DDT programs. By this time, CP/M had been adapted for four different controllers.

In 1976, Glenn Ewing approached me with a problem: Imsai, Incorporated, for whom Glenn consulted, had shipped a large number of disk subsystems with a promise that an operating system would follow. I was somewhat reluctant to adapt CP/M to yet another controller, and thus the notion of a separated Basic I/O System (BIOS) evolved. In principle, the hardware dependent portions of CP/M were concentrated in the BIOS, thus allowing Glenn, or anyone else, to adapt CP/M to the Imsai equipment. Imsai was subsequently licensed to distribute CP/M version 1.3 which eventually evolved into an operating system called IMDOS.

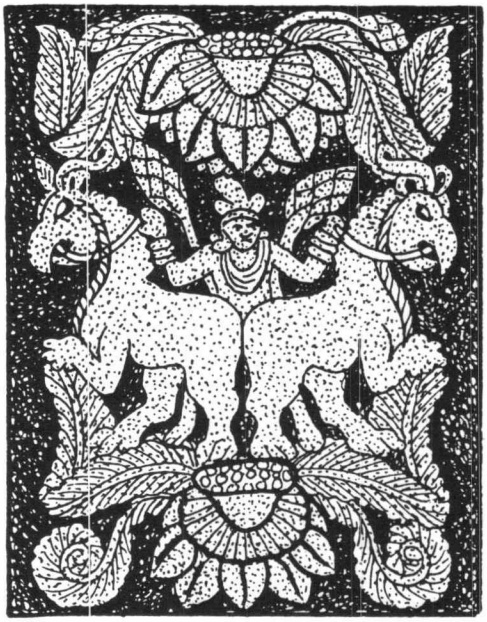
By coincidence, Jim Warren and I were both consulting at Signetics Corporation during this time. Jim was then the editor of *DDJ*, and pushed for sale of CP/M to the general public. There was, at the time, a pervading paranoia among software vendors who felt that any and all loose software would be immediately "ripped-off" by this immoral group of computer junkies. Jim's faith in the industry, however, led me to introduce the CP/M 1.3 system for sale on the open market at $70 per copy. In the months that followed, the nature of the computer hobbyist became apparent. In most cases he was, like myself, in the computer industry and merely wanted a personal computer for his own endeavors. CP/M gradually gained popularity through a "grassroots" effect and, to the amazement of the skeptics, the rip-off factor was practically nil. A new company called Digital Research was formed to support CP/M, develop new products, and provide administrative functions.

It's been nearly three years since CP/M's initial introduction, with several revisions and improvements. Although floppy disks maintain their popularity, CP/M 2.0 is now offerred to manage larger capacity hard disks which are becoming more readily available. Customer needs and demands have also led to the recent introduction of MP/M, a CP/M compatible multiterminal multiprogramming system for more sophisticated applications.

More important, however, is that CP/M provides a manufacturer independent basis for an evolving software market. We all know that software is expensive to develop and support, with numbers quoted in the hundreds of thousands of dollars over the product lifetime. In a classical computer marketplace, these costs are amortized over a few installations, resulting in seemingly outrageous prices. Active CP/M users, however, number in the tens of thousands and can be reached through any number of popular magazines. Thus, marketeers reduce their prices substantially to interest a much larger customer base. Software is sold profitably as an independent commodity by a large number of responsible companies, and the benefits to the consumer are clear. Competition forces low prices and quality control, with selection among a wide variety of software products. Currently, CP/M compatible products range from word processing programs through business systems to a variety of language processors for BASIC, FORTRAN, COBOL, PASCAL, and others. All are priced in the $100 to $700 range. The future is, without doubt, optimistic for producers and customers alike.

An Overview

of CP/M Compatible Software



BY STEVE NEWBERRY
NEWBERRY MICROSYSTEMS
24225 Summerhill Ave
Los Altos, CA 94022

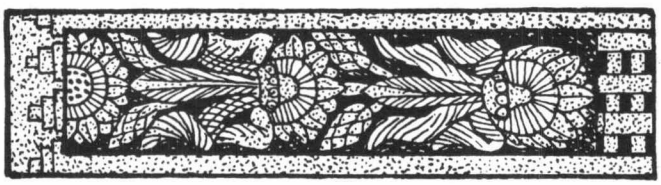
Steve Newberry is a graduate of Stanford University. A member of ACM since 1966, member of the IEEE Computer Society, he is an independent consultant, specializing in contract programming and documentation.

Introduction

One of the reasons I use and recommend CP/M is that there is a lot of very good software that runs under it. In what follows I will give a necessarily sketchy overview of some packages of which I have knowledge. I make no attempt to be either complete or unbiased. There is only one computer game that I care about, The Greatest Game Of All: PROGRAMMING! Therefore, most of the packages that follow will be found to reflect that outlook.

NON-COMMERCIAL SOFTWARE

By non-commercial I mean either public-domain or copyrighted but distributed very inexpensively.



CP/M User's Group Software

Most user's group libraries represent a spectrum of programming interests and capabilities. The CPMUG library is no exception. Most of the programs fall into one of four categories:

1. Utility programs other than language facility programs;
2. Language facility programs, e.g., compilers, assemblers and interpreters;
3. Specialized device drivers which, e.g., interface a modem board or a TVT controller or a printer to the system in question; and,
4. Recreational programs, games.

The following programs are available from CP/M User's Group, 164 West 83rd Street, N.Y.C. 10024. The Price is $8.00 per 8" floppy disk for domestic mailings and $12.00 foreign. CPMUG membership costs $4.00, and includes a catalog of the library. As of this date (Nov 14, '79) there are 33 volumes in the library. The diskettes will be referenced in the sequel by the notation 'x.y' which means "Volume x, program y". Thus 'XREF.ASM (8.27)' refers to program #27, of Volume #8, called XREF.ASM. The organization of the volumes recognizes the difference between games and non-games, so if you decide to purchase a volume (diskette) for one or two of its programs, there is a good chance that the rest of the volume will be of some interest as well.

Volumes 1, 6, 8, 14, 15, 16, 18, 19, 24, 25 and 29 are primarily of the first and third categories: general utilities and specialized device drivers. Working backwards through this list (to get the most recently updated versions), we find:

UNLOAD.ASM (29.23) This is the CP/M version of the ISIS OBJHEX utility which takes a COM file (binary load module) and converts it to hex. This is a VERY useful tool, since CP/M lacks a linking loader. The CP/M utilities DDT and SAVE do, however, enable you to load (typically in low memory) several COM (or HEX) files contiguously, to link these manually (i.e., by patching in the link addresses), and then to SAVE the linked memory image to a COM file origined at 100h. Using UNLOAD, you can convert the COM file to hex and then, again using DDT, relocate the HEX file to any desired location. Warning: Only the source of UNLOAD is given, and the Digital Research Macro assembler is required for assembly (but you ought to have that anyhow). This program alone is worth the $8.00, but there's lots of other good stuff on the volume.

COPY.ASM (1,3 and 25.17) A track-to-track diskette copying routine (the second version is for Tarbell controller) which is very fast and convenient to use.

FORMAT.ASM (25.18) Initializes a diskette, requires Tarbell controller.

DUMP.COM (24.2) Updated version of (14.6). Allows you to look at the contents of a diskette by file, by track, by sector or by group (the sequence of sectors used by CP/M in allocating disk space to a file), and allows you to directly alter any byte on the disk. Used to recover inadvertently ERAsed files or to recover (sometimes!) files which have been otherwise lost. An invaluable tool! Three cheers for Sam Sinser, the man who wrote it. Includes source code.

XDIR.COM (24.11) Another Sam Sinser goodie, this prints out the directory of the indicated diskette, alphabetically sorted, formatted so that it all falls on a single screen display, and tells you how much space is left. Try it, you'll love it.

BYTEMOV.ASM (8.7) Driver program for either Cromemco Bytesaver or Piiceon Program Saver from boards. Very elegantly programmed by Jeff Shook.

XREF.ASM (8.27) Use this to convert the assembly listing from ASM or MAC (the Digital Research assemblers) to a fully CROSS-INDEXED listing. An invaluable aid in documenting (and debugging) assembly code programs.

COMPARE.ASM (6.4) Compares two files, tells you WHERE they differ (if they differ) and how long they are (if they don't). This is the sort of program, that you don't use often but that you'll sell your soul to have when you DO need it! (Like when PIP tells you that a filecopy has aborted on a VERIFY error but won't tell you WHERE—it KNOWS, but it won't tell—at such times COMPARE is a godsend.)

IDIR.COM (1.18)

ICOPY.ASM (1.17) Given an ISIS diskette, you can (a) find out what is on it, and (b) copy what is on it to CPM diskette. NICE!

The above programs represent only a small fraction of the utilities in the CPMUG library, but they are some of the ones that I have on my system disk and use all the time.

Volumes 2, 4, 5, 7, 10, 11, 12, 16, 23, 30 and 31 are primarily devoted to the second category, i.e., language facility programs.

LLLBASIC.ASM (2.1 and 10.1)

LLLBASIC.COM (2.2 and 10.2)

LLLFP.ASM (2.4 and 10.3) Lawrence Livermore Lab BASIC is in the public domain. The later versions are to be preferred.

TINYBAS.COM (2.8 and 11.7) Lichen Wang's (public domain) Palo Alto Tiny BASIC, patched for CP/M SAVE and LOAD, with full documentation (2.9 and 11.8). The later version is to be preferred.

BASIC.COM (30.5)

RUN.COM (30.7)

FPCONV.SRC (29.24)

FPDATA.SRC (29.25)

FPINT.SRC (29.26)

FPPKG.SRC (29.27)

TRAN.SRC (29.28) Gordon Eubanks' BASIC-E is the best BASIC in the public domain. It is a semi-compiled BASIC and is a precursor of the very successful CBASIC by the same author (Volume 30 also contains the PL/M source if you care to see how he did it). BASIC-E includes the transcendental functions, string manipulation and CP/M file I/O. Floating point arithmetic numbers are represented to slightly better than seven digits of significance. Variable length strings and n-dimensional arrays are both dynamically allocated. Line numbers are required only when targeted by GOTO or GOSUB. Identifiers may be any length and the first 31 characters are significant. Execution is rather slow: the canonical "FOR I=0 TO 10000: A=5: NEXT I: END" takes about two minutes and ten seconds to execute. A manual used to be available from Digital Research.

BASIC/5.ASM (11.1)

BASIC/5.COM (11.2)

BASIC/5.DOC (11.3) This is Processor Technology's 5k BASIC adapted for disk. An alternative to LLL BASIC, approximately the same size, but different flavor. Not in the public domain.

PILOT Volumes 7 and 12 are devoted to John Starkweather's masterpiece; the former is for Intel MDS, the latter for CP/M. If you are interested in writing Computer Aided Instruction programs, then this is the language of choice—it was designed for precisely that purpose. It can be EASILY learned by children in kindergarten or by people who "hate

computers." PILOT is in the public domain. Persons wishing to obtain further information on PILOT would probably get pointed in the right direction by Gregory Yob, P.O. Box 354, Palo Alto, CA 94301, Telephone (415) 326-4039.

Z80ASM The Z80ASM package (16.15 through 16.21) includes a running Zilog-mnemonics 8080/Z80 cross-assembler, source code for same, and some documentation.

MACASM.COM (16.8) This is a short macro assembler (no source) with a small bug which could be rapidly located by the DDT II debugger (see below in the commercial section).

ASMX.COM (16.1) This is another 8080/Z80 cross-assembler (no source), recognizes extended-Intel mnemonics, and is extensively documented in file VOLUME16.DOC and (17.11).

STOIC Volume 23 is devoted to STOIC, a dialect of FORTH which is well-suited to interactive real-time control systems. The system includes a file handling system, an editor, an interrupt handler, a rich assortment of floating and fixed-point arithmetic routines and "much, much more." WARNING: Learning to program in STOIC (or FORTH) is a non-trivial undertaking. Although its proponents regard it as a high-level language, it is in fact a structured macro-assembly language for a virtual stack machine. The experienced systems hacker will take it in stride, particularly if he has a background in assembly programming a zero-address machine, but the neophyte may find it hard going at first.

PROCSELF.ASM (18.7) The source code for the classic "Processor Technology Software #1". A small, fast, dumb assembler for non-disk systems, primarily of historical interest as the first public-domain software for 8080 available to the general public. Back in 1975, it and Microsoft BASIC were almost the only systems software available for the 8080. Rest in peace, old friend.

RATFOR.COM (26.4) This is a preprocessor which translates a nicely structured language (called RATFOR) into ANSI Fortran. It is extensively documented in the classic text "Software Tools", by Kernighan and Plauger. Some slight amount of editing is required to reconcile the emitted FORTRAN declarations to the conventions recognized by the Microsoft FORTRAN compiler. Although I no longer remember the specifics, a couple of macros in ED does the job. I've done it myself—it's no big deal.

ALGOLM.COM (28.10)

RUNALG.COM (28.11) Volume 28 is one of the most outstanding software values that I know of. It contains, in addition to the ALGOL package, a set of database, inventory and mail-list programs. But it is the ALGOL package which is so utterly mind-blowing. Like BASIC-E, it is the product of the Naval Postgraduate School in Monterey, and is an intelligent and thoroughly reasoned adaptation of a classic general-purpose programming language (in this case, ALGOL-60) to the CP/M environment.

First, the bad news: No linkage to assembly-language subroutines and no built-in transcendental functions. But now the good news: ALGOLM, ALGOL-60 and PASCAL are fundamentally just variant dialects of the same language, and therefore conversion of programs from one dialect to the other is, in the general case, largely a matter of reconciling the declaration-formats. Since there is a prodigious amount of PUBLISHED software in ALGOL-60 and in PASCAL, it follows that ALGOLM is not merely elegant (which certainly it is), but REALLY useful as well! Here are some of the characteristics of ALGOLM:

Three types of variables, signed integer, signed decimal and string. Decimals may be declared with up to 18 digits of precision and strings may contain up to 255 characters. Decimal precision and string length may be represented as integer variables which are assigned actual values at run time.

Arrays may be of type integer, decimal or string, may have up to 255 dimensions, and dimension bounds may be represented as integer variables which are assigned actual values at run time.

Control structures are: BEGIN, END, FOR, IF THEN, IF THEN ELSE, WHILE, CASE and GOTO.

BEGIN/END blocks may be nested to nine levels. Variables declared within a given BEGIN/END block are only accessible from within that block, and run-time storage space for such variables is de-allocated upon exit from that block.

Functions and procedures may take parameters (call-by-value only) and may be called recursively.

ALGOLM is semi-compiled and therefore both compact and reasonably fast in execution. (The canonical benchmark ran in one minute and five seconds.)

The ALGOLM package is well documented and there are a number of sample programs.

TBASIC The complete package consists of Volumes 31 and 32. TBASIC is said to be "upward compatible" to Microsoft 8k (Release 4.0), and is very well documented. The editor is a vast improvement (to my taste) over that in Microsoft BASIC, and is rather similar in style to ED. TBASIC has several features not found in other BASICs. ASSIGN and DROP commands permit assignment/de-assignment of up to 10 physical devices to any of 6 logical I/O devices. Long variable names and non-numeric labels are permitted. Arguments may be passed to subroutines, and local variables may be declared. Linkage to machine-language subroutines is convenient (CALL command). The bad news is that I was unable to get the CSAVE (to disk) to work. But I imagine that the version which is sold by Tarbell (for $36.00) has that fixed. TBASIC IS FAST: THE CANONICAL BENCHMARK (SEE BASIC-E ABOVE) RAN IN TEN SECONDS FLAT! All in all, a very well thought out alternative to BASIC-E.

The balance of the CPMUG library is largely devoted to games (except for volume 9, which is a General Ledger package for Microsoft 12k Disk Basic Version 4.0, and for volume 33, which is a very serious set of programs for automating aircraft search-and-rescue operations).

WARNING!!! WARNING!!! WARNING!!!

The majority of the CPMUG programs have never heard of lower-case ASCII. Fortunately, the assembly-source code is generally given, so this problem is easily fixed: Invoke ED with

the source file. Look for an instance of 'MVI C,1' (or 'MVI C,1' if tabs are used). You are looking for the set-up for a system call "read console" (Function #1, see p.7, CP/M Interface Guide). The command 'bmfMVI C,1ZOtt' (or 'bmfMVI C,1') will quickly tell you how many such lines there are, (if any). If this draws a 'BREAK "*" AT ^Z', then there will be an EQU statement setting something like 'CONSINP' to 1, so look for that. Then follow the 'CALL BDOS' or 'CALL 5' (just after the 'MVI C,1') with code to mask out bit 5 of all characters between 61 h and 7ah. Now assemble and enjoy.

Two other sources of CP/M compatible public-domain software are:

The Tarbell Public Domain Software volume, available from Tarbell Electronics, 950 Dovlen Place, Carson, CA 90746 for $10.00 (domestic). This is a collection of all of the Tarbell-dependent software in the CPMUG library brought together on a single disk.

The fig-FORTH package. The diskette is $35.00, available from FORTHRIGHT ENTERPRISES, P.O. Box 50911, Palo Alto, CA 94303. The documentation consists of two volumes, the Installation Manual and Glossary, and the Source Code listing + aids. These are available from the FORTH INTEREST GROUP, P.O. Box 1105, San Carlos, CA 94070 at $10.00 per volume.

fig-FORTH is a standardized FORTH which ports smoothly between 8080, 6502, PDP/11 and very possibly several other machines as well. Its proponents regard it as the greatest thing since the invention of toilet paper, and they may well be right. (But see the comment on STOIC above).

COMMERCIAL SOFTWARE

The following are packages which either I own and use on my own system, or which I do not own but have used on some other system, or which I know of only indirectly through the testimony of colleagues and/or study of the documentation. Here again the selection is far from complete and reflects my own personal biases and interests. For a broader overview of the commercially available packages which run under CP/M, consult the advertisements of LIFEBOAT ASSOCIATES, 2248 Broadway, NY, NY 10024.

Editors and Output Formatters

WORD-STAR WORD-MASTER is ED with a cleaner protocol for inputting and outputting subfiles, a more powerful macro facility (conditional termination and nesting) and, in addition, a VIDEO MODE in which the alterations to text are dynamically monitored on the display screen as the edit commands are entered. WORD-STAR is the screen editor module of WORD-MASTER, plus an output formatter which is also dynamically monitored on the display screen. This is REALLY powerful, since you can see how the page sets up as you enter the formatting commands. It also has some other goodies, e.g., a menu-driven installation program which tailors the editor/formatter to a number of video terminals and hard copy printers, including the DIABLO and QUME daisywheel printers.

If you already have extensive experience with ED (or any of the other sons-of-TECO), and/or if you already have a formatter that you like (or NEED a formatter that automatically generates a sorted index), then WORD-STAR is less attractive than WORD-MASTER. But if your text-entry is to be done by inexperienced personnel, then WORD-STAR would more than justify its purchase price by increasing the output of a newly trained operator. Both packages are SUPERBLY documented—an outstanding job. WORD-MASTER sells for $125.00, and WORD-STAR for $495.00.

TEX This is the text formatter offered by Digital Research. It offers the standard controls, pagination, margins, line length, paragraphing, etc., and is entirely adequate for letters or short brochures. There is no user-defined macro facility. Documentation is very good, with many examples. TEX sells for $85.00.

TEXTWRITER III This has all the functions of TEX, plus several extremely powerful capabilities for the generation of form-letters and sorted indexes. The form-letter feature will either take the names and addresses from a diskfile, or stop and prompt the operator to enter them manually. The indexing feature allows you to flag key-words (as they occur in the text) for inclusion in an index. Upon command, TEXTWRITER III will compile a sorted index with appropriate indentations. For tech writers this is an invaluable aid. The documentation is quite good. The power of the package, if you leave out the form-letter and sorted-index feature, is about the same as other formatters. There is no user-defined macro facility. TEXTWRITER III sells for $125.00 and is available from LIFEBOAT ASSOCIATES.

TSC TEXT PROCESSING SYSTEM If you don't need the sorted-index feature you should look into the TSC TEXT PROCESSING SYSTEM, available from Technical Systems Consultants, Inc., P.O. Box 2574, West Lafayette, Indiana 47906. This sells for $50.00, is patterned after the UNIX utility NROFF, and is sold with a manual which INCLUDES THE ASSEMBLY SOURCE CODE! This is an EXTREMELY powerful package, with 27 memory-registers, user-defined macros, conditional macro-execution and access to diskfiles. There is a pre-defined set of macros which includes form-letter generation. I've used it since it first came out and have never had a complaint.

ASSEMBLERS

MAC In addition to the ASM assembler which comes as an integral part of the CP/M package, Digital Research also markets a macro assembler, MAC, which does not yet support relocation, but otherwise is very similar to the Intel macro assembler, ASM-80. MAC allows long (up to 16 characters) identifiers and puts out a special symbol table which is used by the SID debugging package (see below). The MAC package includes a set of macros called SEQIO.LIB (which VERY much simplifies interfacing assembly-level programs with the operating system), and another set of macros called Z80.LIB which enable MAC to assemble Z80 op codes. The Z80 facility utilizes extended-Intel mnemonics, rather in the style of the TDL/Xitan assembler. The documentation consists of a well-written tutorial and includes lots of worked examples. It got me started sooner than would have been the case had I ONLY the Intel "8080/8085 ASSEMBLY LANGUAGE PROGRAM-

MING MANUAL (Order Number 9800301C)" to work with. However, I did find it necessary to purchase the Intel manual in order to fill in certain gaps in the tutorial treatment. (The Intel manual was only $5.00, so no big deal.)

MAC will assemble any source code accepted by the Intel ASM-80 assembler documented in the above manual, EXCEPT the relocation pseudos, ASEG, CSEG and DSEG. The MAC package sells for $100.00 and I recommend it highly.

MACRO-80 The Microsoft MACRO-80 package is available either separately at $150.00, or as a part of either of the Microsoft BASIC or FORTRAN compiler. It is said to conform to the Intel ASM-80 specification in all respects, and the package includes a Linking Loader and Library Manager. The documentation is adequate. (But see remarks on FORTRAN compiler below).

SMAL/80 SMAL/80 is for those who need to write assembly level code on an occasional basis, but are not yet familiar with the 8080 instruction set mnemonics. It is a Structured Macro Assembly Language which uses a semi-high level notation convention which shows a family resemblance to the C language. (The original version of SMAL was developed by C. Popper at Bell Labs.) Control structures are several flavors of brackets (BEGIN/END, DO/END, [/], {/}), IF < condition > THEN < statement >, IF < condition THEN < statement > ELSE < statement > and loop controls LOOP < statement > REPEAT and LOOP < statement> REPEAT WHILE <condition>, THE MACRO PROCESSOR PERMITS NESTING TO ANY DEPTH AND RECURSION. Although it can be cogently argued that SMAL should be used even by experienced 8080 assembly programmers, (because of the enhanced readability of the code), I doubt that many will be convinced: The sensation is one of "too many extra keystrokes for too little payoff." The documentation is in the Bell Labs style, clear, succinct and complete. SMAL/80 sells for $75.00.

SP80

SAL/80 These two packages represent a middle way between SMAL/80 and straight macro-assembly programming. They are libraries of macros which provide control structure macros for specific macro assemblers. SP80 has versions for the TDL/Xitan macro assembler, the Digital Research MAC, "the Intel macro standard and MACRO-11 (PDP-11). I have not seen this package or its documentation, only the promotional literature, but I am myself the author of SAL/80, and it would appear that the two packages are quite similar. The SP80 package sells for $95.00 from SP80, P.O. Box 2745, Reston, VA 22091.

SAL/80, in addition to the library of control structures, contains a library of utility macros, such as, e.g., multiplication, division, string move and search, output string to console, etc., and will go into distribution in February for $50.00, available from Newberry Microsystems, 24225 Summerhill Ave., Los Altos, CA 94022.

BASIC

MICROSOFT BASIC (MBASIC) The "Industry Standard" is, of course, MBASIC by Microsoft. It is fast (as semi-compiled languages go) executing the canonical benchmark in 28 seconds. Unlike most semi-compiled systems, MBASIC does the translation to intermediate code during program entry, so the translation process is invisible. The intermediate code is extremely compact and program development time is greatly reduced by the continuously interactive nature of the process. You enter some code, test it, modify it until satisfied, and then SAVE it to disk. The current release has overcome the shortcomings of most other BASICs: you can now use long variable names, loops may be written in structured fashion with the WHILE/WEND statement form, and conditionals may be nested. MBASIC has evolved into quite a civilized programming language. Moreover, MBASIC can be compiled to native code on the Microsoft BASIC compiler. Thus, the convenience of interactive program development can be combined with the speed of native code compilation—a truly impressive (and USEFUL!) achievement. MBASIC goes by the name of 'Disk Extended BASIC' and sells for $300.00. The MBASIC compiler sells for $350.00.

CBASIC II CBASIC grew out of BASIC-E by the addition of bcd arithmetic to fourteen places of significance, the WHILE/WEND control statement, integer variables, chaining with common variables, PRINT USING and improved file-handling capabilities. A number of very useful data-processing packages (e.g., accounting, database, merge-sort etc.) have been written in CBASIC and field-tested for several years. CBASIC sells for $109.00.

COMPILERS

FORTRAN FORTRAN-80 by Microsoft is being distributed by a number of firms under a number of names. There are a zillion copies out there and running. It's been in distribution for several years now and is (presumably) well wrung out. It conforms to the ANSI '66 standard (except for COMPLEX) and includes the MACRO-80 assembler, Linking Loader, Library Manager and a library of standard subroutines. The documentation appears to be complete but reads more nearly like a design spec than a user document.

PASCAL/M Written by DIGICON and distributed by Digital Marketing, 2670 Cherry Lane, Walnut Creek, CA 94596, (415) 938-2880, Pascal/M is a recent arrival on the CP/M scene, and one most welcome. It is source-code compatible with UCSD Pascal, and hence is an almost "full spec" Pascal. The following restrictions apply: No parameters of type Procedure or Function. A File is NOT allowed as a component of a structure. The procedure DISPOSE performs no function. (However, MARK and RELEASE are provided to manage the heap.) Standard procedures PACK and UNPACK are not provided. A GOTO is not permitted to jump out of the procedure/function in which it occurs. (However the EXIT procedure of the UCSD version is provided). Digicon has designed a new virtual machine (new P-codes) and two interpreters are available, one for 8080 and one for Z80. (I have the Z80 version and it is FAST. The canonical benchmark executes in 10 seconds.)

The package consists of the compiler, the interpreter, a runtime library and a set of installation programs. I brought my copy up in just the time that it took to insert the diskette and execute the INSTALLP program. (There is another program for tailoring the interface to particular consoles, but I

did not have to use it.) The documentation is concise without being excessively terse. It sells for $350.00 and I think it's a real winner.

PASCAL/Z This is a native code compiler for a subset of Pascal. Missing are floating point, variant records, the ability to pass procedures/functions as parameters and (SOB!) POINTERS. This assessment is based on the documentation which prevailed almost a year ago, before they (Ithaca InterSystems) had a CP/M version. The documentation of the compiler was rather scanty, suggesting that the product was, at that time, VERY new indeed. It sells for $275.00 and merits further investigation.

WHITESMITH'S C Compiler This is a BIG compiler. It compiles the full spec C language as set forth in Kernighan and Richie's book "The C Programming Language" (Prentiss-Hall, 1978). The package includes the compiler, "narrative assembler", linking loader, library and a set of installation programs. The documentation is in the style of the UNIX Programmer's Manual, encyclopedic but terse. The installation is simple: plug in the diskette and follow instructions. The compiler consists of four modules (including the "narrative assembler"), which total out to 156 Kb. There are lots of error diagnostics and substantial code optimization is performed. The first module is the preprocessor which does macro expansions. This is overlaid by the P1 pass, which does lexical and syntactical analysis and some optimization. P1 is then overlaid by P2 which completes the optimization and emits the intermediate code in "narrative assembler" notation. (This is a highly condensed assembly level notation which, like that of SMAL, is reminiscent of C.) The assembler, called 'A-natural', overlays P2 and generates a relocatable load module. The compilation process is initiated by calling a SUBMIT file which handles the overlaying and creation and deletion of scratch files. The whole operation is somewhat faster than a PLM compilation, but not a whole bunch. The link-load process is also handled by a SUBMIT file, and generates a COM file.

Bill Plauser (at Whitesmith's) tells me that there are about 150 copies in the field to date and that the feedback indicates that the system is very stable. The Whitesmith's C Compiler sells for $630.00 from LIFEBOAT ASSOCIATES. If you want an optimizing compiler for a structured language, this is the one.

BDS C Compiler This is the work of Leor Zolman (at M.I.T.) and realizes a subset of C. There is no data initialization, no long types, no floating point, no STATIC or REGISTER declarations, and not a whole bunch of optimization. However there is still plenty left for systems programming, and the package includes a linking loader and library manager. Tom Gibson (of Tiny-c Associates) tells me that he and his associates have been using the compiler with uniformly satisfactory results. The compiler generates relocatable load modules for 8080, the linking loader link/loads (multiple) object file(s) into a COM file. The documentation (again quoting Tom) is written in the style of, "Here's how to use my compiler". The BDS C Compiler sells for $110 including the Kernighan & Richie book cited above.

BASIC See MBASIC, above.

A GREAT WAY TO LEARN STRUCTURED PROGRAMMING

TINY-C This is a tiny masterpiece. The documentation is so good that it can (and should) be used as an elementary textbook. The tiny-c language facility consists of a machine-dependent interpreter (currently available, 8080 and PDP-11), and a Program Preparation System (PPS) written in tiny-c (and hence machine independent). PPS comprises an editor and run-time monitor. This results in an interactive program development system which is as convenient to use as any of the interpreted BASICs. You enter the program, test it, revise it, retest, etc., and write out to disk without having to change environment. When runtime errors are detected, the monitor halts execution and gives remarkably good diagnostic messages. UNlike the BASICs (with the exception of TBASIC) the editor is both pleasant to use and a preparation for bigger and better editors. Similarly, the language tiny-c.

Although tiny-c is, indeed, a VERY tiny C, it is nonetheless quite adequate for getting taste of the "hows and whys" of full-scale languages. Indeed, the expressive power of the language is comparable to the systems language XPL (the forerunner of PLM), and literally hundreds of compilers have been written in XPL! Tiny-c has IF/ELSE, WHILE, RETURN and BREAK for control-structures, integer and character for data types, one-dimensional arrays for data-structures, and a pointer convention which permits call-by-reference as well as call-by-value. Not too shabby!

The source code of PPS, together with several worked examples provide excellent models of style and program structure. PPS allows you to load any number of program modules, which are then "linked" by the runtime monitor. Provision is made for linkage to machine-code subroutines. The assembly source of the interpreter is given together with a 34 page analysis of its anatomy and functional behavior. Finally, there is extensive systems-level documentation to facilitate the installation of the system on non-CP/M systems. The CP/M installation is essentially a matter of inserting the disk and calling the interpreter and PPS. Instructions are provided for tailoring PPS to systems with more than 24Kb. An appendix contains Newsletter updates. Tiny-c sells for $75.00 and is distributed by tiny-c associates, P.O. Box 269, Holmdel, N.J. 07733.

DEBUGGING AIDS

DDS-II This is the best debugging tool I have ever used. It is comparable to the ICE software on the Intel MDS system. DDS takes over the video display screen (you can toggle back to the program-generated display at will) and maintains a running display of:

the value of the stack pointer, the top five words on the stack, the five words pointed at BY the words on the stack (displayed in both hex and ASCII)

the next five instructions, disassembled (and the first byte also displayed in hex), beginning with the current instruction

the AF, BC, DE, HL registers together with the value of the PC at the instruction PRIOR to the current instruction, including the ASCII interpretation of the A register and

mnemonic representation of the condition flags, and the hex contents of the memory words pointed at by the BC, DE and HL registers, and a DDT-like display (in either 'D' or 'L' mode) of any chunk of memory you wish to look at.

There is also a display, the "call-stack", which shows the last five return-addresses stacked as the result of a call, and you can choose to be notified whenever a subroutine returns to an address other than that which was stacked when it was called (think about that!).

You can choose to be notified of a LOT of behaviors: if the program references data outside of a predefined range, or when the PC goes out of a predefined range, or when the program references a given location, or when the SP goes beyond a predefined range, or when the program IS ABOUT TO store at a predetermined location, or when a given register or memory byte/word attains a given value. You can do any and all of the fill, insert, display, move etc., commands usually found in most debug monitors, including all the flavors of breakpoints imaginable. You can single-step or two-step or n-step, I mean you have that program by the short-hairs!

The documentation is excellent, even to the point of including a nine-page explanation of what assembly programming is all about. (I think that qualifies as overkill.) The package is available in off-the-rack versions for either VDM-1 or ADM3A video displays, and can be patched to run on any display which has x-y cursor addressability. I have only two criticisms of the package: the time and space given to a tutorial on assembly-language programming would have been more usefully employed in outlining the patching procedures for non-VDM/ADM displays, and I wish to God that the disassembler that drives this program understood about Z80 code. Distributed by PRS, 257 Central Park West, New York City, N.Y. 10024, and selling for $65.00, DDS-II is one of the great bargains of all time. No joke.

SID Digital Research's update of DDT, the Symbolic Instruction Debugger adds three instructions to the DDT vocabulary, two utilities (TRACE and HIST) and allows you to specify addresses by the labels assigned them in the source code as an alternative to the hex representation. While certainly it is very nice to be relieved of the need to refer back and forth to the listing to figure out the logical signficance of hex addresses, I haven't really got much good out of that feature, since when I'm doing that kind of debugging I'm talking to DDS II instead. But the extensions to the command set, particularly the hex arithmetic and the pass counter command constitute a signficant enhancement of DDT.

The TRACE utility will compile a backtrace record of the program addresses executed PRIOR TO hitting a specified breakpoint. This is one of those "you don't need it often, but WHEN you need it, nothing else will do" utilities. Similarly, the HIST utility profiles the activity regions of a program in execution, enabling you to identify the "hot spots". Empirical studies have shown that most programs tend to "spend 90-95 percent of their time executing 5-10 percent of the code." Thus an improvement of even rather modest proportions in a critical area will have more impact on over-all execution speed than a dramatic speedup of the relatively idle areas. The HIST utility justifies the price of the package. SID sells for $85.00 and an updated version which knows about Z80 code is in the works.

GO:
A UTILITY COMMAND UNDER CP/M

BY ANTHONY SKJELLUM
1695 Shenandoah Road
San Marino, CA 91108

No facility is provided under CP/M to reexecute a machine language program already residing in the transient program area, nor is there a direct facility for loading binary files (.COM files) into the TPA without executing them except with the use of DDT, the debugging tool. However, both these functions can be performed through the CCP (central command processor) by way of two programming "tricks."

A special command must be created under CP/M in order to reexecute a program. This is done by way of a "null" .COM file. Using the CCP "SAVE" command, such a file may be created as follows:

SAVE 0 @.COM

When "@ <CR>" is typed at the console, CP/M will search the directory for "@.COM" but will not load any new bytes into the TPA since the program has zero length. CP/M will then give control to the program at the beginning of the TPA (100H) and, in effect, the old program will have been reexecuted. It is desirable to place the "@" command at or near the beginning of the directory to save search time by CP/M. Also, the user should be aware of programs which operate erroneously when reexecuted, such as the SAVEUSER command under CP/M on Micropolis and CP/M on Vector MZ.

In order to get the CCP to load a .COM file without executing it, you must cause it to issue an error message while processing your command line. This is done by imbedding a control character in the data field after the program name as follows:

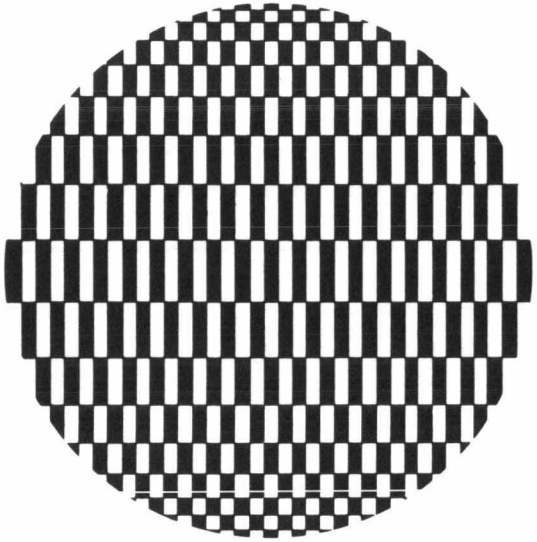
FILE ^A <CR>

Note: Any control character not used by the CCP's line input routine are considered illegal and will remain imbedded in the command line, causing the desired error to occur. In this case the program FILE.COM will be loaded into the TPA. However, when CP/M scans the data field to set up the two default FCB's, it discovers an illegal character and issues an error message (a "?"). The program can be executed when desired by using the "@" command discussed above. This is generally more useful for single drive systems, but sometimes comes in handy in two drive systems also.

The program listing below is a non-trivial CP/M command called "60." This performs a function which is useful when no monitor or front panel is available. The program will execute the address specified by the four hex digits in the data field. Spaces are considered to be zeroes.

I hope the preceding is useful to some of the CP/M users who read *DDJ*. I enjoy the useful content of this magazine and hope it continues in the future. Finally, if there are any usrs of CP/M Micropolis or Vector MZ who have programs they would like to trade, I would appreciate hearing from them.

```
;
; 60: A utility command under CP/m by Anthony Skjellum
;
; September 14, 1979
;
;
; This command performs the following function:
;
; 60 (hex addr) executes the address specified
;
;
;
0100                    org     100h            ; tpa start
;
0080 =          tbuff   equ     80h
005C =          fcb     equ     5ch
;
0100 110000             lxi     d,0             ; initial hex value to zero
0103 215D00             lxi     h,fcb+1         ; point to fcb created by
                                                ; the CCP (i.e. has hex
                                                ; address )
;
0106 0604               mvi     b,4             ; digit count
0108 7E         go:     mov     a,m             ; get digit
0109 FE20               cpi     20h             ; check for space
```



```
010B C21001             jnz     go1             ; if not then use data fetched
010E 3E30               mvi     a,30h           ; equate a space with an entry of zero
0110 CD1B01     go1:    call    hexin           ; accumulate value
0113 D8                 rc                      ; return to CCP if error
0114 23                 inx     h               ; look at next location
0115 05                 dcr     b               ; decrement digit count
0116 C20801             jnz     go              ; if non-zero then continue
0119 EB                 xchg                    ; get address value in HL
011A E9                 pchl                    ; after accumulation, execute address
                        ;
                        ;
                        ; support routines:
                        ;
                        ;
                hexin:
011B EB                 xchg
011C CD2801             call    check
011F D8                 rc
0120 29                 dad     h
0121 29                 dad     h
0122 29                 dad     h
0123 29                 dad     h
0124 B5                 ora     l
0125 6F                 mov     l,a
0126 EB                 xchg
0127 C9                 ret
                        ;
                        ;
                        ;
0128 D630       check:  sui     30H             ; check for valid hex digit
                                                ; return with carry set if invalid
012A D8                 rc
012B FE0A               cpi     0AH
012D 3F                 cmc
012E D0                 rnc
012F D607               sui     7H
0131 FE0A               cpi     0AH
0133 D8                 rc
0134 FE10               cpi     10H
0136 3F                 cmc
0137 C9                 ret
                        ;
                        ;
                        ;
                        ;
0138                    end

^Z

 HEX DUMP OF "GO" PROGRAM

:10010000110000215B0006047EFE20C210013E3079
:10011000CD1B01DB2305C20801EBE9EBCD2801DB9E
:1001200029292929B56FEBC9D630D8FE0A3FD0D688
:0801300007FE0AD8FE103FC9CA
:0000000000

A>
```

Linking CP/M To George Morrow's Cassette Interface

BY WAYNE MILLER
905 Fairmount Blvd.
Jefferson City, MO 65101

I have found that maintaining compatibility with others is ideal for exchanging software; however, I have had problems since I acquired a double density minidisk system. Therefore, I decided to link a cassette interface to CP/M, allowing me to exchange software with others even if they do not have disk. I decided on the Speakeasy by George Morrow because it used the KC300 baud standard. This is very slow; but the chances of losing data from bit drop-out is very unlikely. I accepted the drawbacks of the slow speed since I would only use the cassette to read the program to disk.

I am presenting this program primarily as an example to others to use when writing software to transfer data from a disk to a cassette tape.

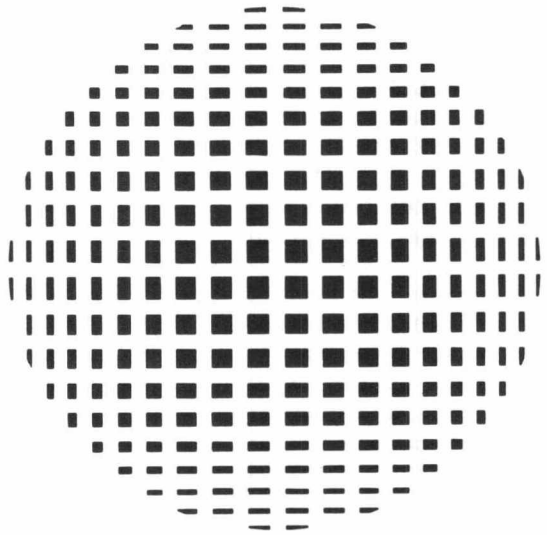
The functions that effect a data transfer have been coded in a structured format, which should make it wasy to follow. The reader, however, will need to consult the CP/M and Speakeasy manuals to get the full description of the disk and tape functions.

To change the direction of the transfer, the order of reading/writing the tape and disk will need to be reversed. I have marked the areas in the program listing for the reader's benefit. Note that I transfer a sector at a time. This could be easily changed if block transfers are desired. Also, there are two possible disk errors. A #1 error is caused when the program tries to open a nonexisting file. A #2 error occurs when a disk write exceeds available disk space or the disk is in a write protect mode. If such an error occurs, the message generated by my program will have the following format:

"# – DISK I/O ERROR ON BDOS".

```
        ;CP/M TO GEORGE MORROW'S CASSETTE INTERFACE
        ;SOFTWARE LINKAGE
        ;                WAYNE MILLER
        ;                905 FAIRMOUNT BLVD.
        ;                JEFFERSON CITY, MO. 65101
        ;
        ;       MAY 1979
        ORG     100H
BDOS    EQU     0005H; ENTRY POINT TO CP/M
CASSET  EQU     810AH   ; COPE TAPE SYSTEM
CREATE  EQU     22; DOS CREATE FILE
CLOSE   EQU     16; CLOSE THE FILE
WRITE   EQU     21; DOS WRITE RECORD
TAPEF   EQU     202Q    ; TAPE READ COMMAND
OPENF   EQU     15; OPEN FILE CODE
READF   EQU     20; READ FILE CODE
PRINTF  EQU     9; PRINT ON CONSOLE
TYPEF   EQU     2; TYPE ON CONSOLE
CONIN   EQU     1; READ FROM CONSOLE
FCB     EQU     05CH; FILE CONTROL BLOCK ADDRESS
BUFF    EQU     80H; DISK DMA BUFFER
 ; BDOS REGISTERS RETURN VALUES IN A AND B
 ;                FUNCTION REG. IN C
 ;                FORWARD VALUES IN E AND D
 ;
        LXI     H,0
        DAD     SP
        SHLD    OLDSP
        LXI     SP,STKTOP
        MVI     A,'0'
        STA     IBP
        JMP     MAIN
IBP     DS      1; SECTOR COUNTER
OLDSP   DS      2
STACK   DS      64
STKTOP  EQU     $
 ; PRINT CHARACTER ROUTINE
PCHAR   PUSH H! PUSH D! PUSH B ; SAVE REGISTERS
        MVI     C,TYPEF
        MOV     E,A
*
        CALL    BDOS
        POP B! POP D! POP H ;RESTORE
        RET ; TO THE CALLER
        ;ROUTINE TO PRINT BUFFER ON CONSOLE
PRIOUT  PUSH    H! PUSH D! PUSH B; SAVE
        MVI     C,PRINTF
        CALL    BDOS
        POP     B! POP D! POP H; RESTORED
        CALL    CRLF
        RET ;   TO THE CALLER
        ;ROUTINE TO PERFORM DISK I/O
FILEOP  PUSH H! PUSH D! PUSH B; SAVE
        CALL    BDOS
        POP B! POP D! POP H; RESTORE
        RET
        ;ROUTINE TO PERFORM TAPE I/O
TAPEOP  PUSH H! PUSH D! PUSH B; SAVE
        CALL    CASSET
        POP B! POP D! POP H; RESORED
        RET
        ;ROUTINE TO SEND CARRIAGE RETURN AND LINE FEED
CRLF    MVI     A,0DH
        CALL    PCHAR
        MVI     A,0AH
        CALL    PCHAR
        RET
        ;ROUTINE PERFORMS LOW LEVEL ERROR ANALYSIS
ERR      CALL   CRLF
        MVI     A,'#'
        CALL    PCHAR
        MOV     A,B
        ADI     '0'
        CALL    PCHAR
```

```
        CALL    CRLF
        LXI     D,MSG3
        CALL    PRIOUT
        JMP     FINIS
MAIN    LXI     D,MSG6
        CALL    PRIOUT
        IN      40H
*
        ANI     200Q
        JZ      MAIN
        CALL    SETUP
        LXI     D,MSG1
        CALL    PRIOUT
        MVI     A,'$'
        STA     DELIM
        LXI     D,FCB
        CALL    PRIOUT
        CALL    CRLF
GLOOP   IN      0FFH ; GET SEGMENTS LIMITS
        MOV     B,A
        LDA     IBP     ; DONE
        CMP     B
        JZ      FINIS   ;YES
        LDA     IBP     ;NO
        INR     A
        STA     IBP
        CALL    CRLF
        LDA     IBP
        CALL    PCHAR
        MVI     A,TAPEF <- CHANGE TO READ OR
        MVI     C,0        WRITE FROM DISK AND TAPE
        LXI     D,128
        LXI     H,080H
        CALL    TAPEOP
        CALL    DISK
        LXI     D,MSG4
        CALL    PRIOUT
        JMP     GLOOP
FINIS   LXI     D,MSG2
        CALL    PRIOUT
        LXI     D,FCB
        MVI     C,CLOSE
        CALL    FILEOP
        LHLD    OLDSP
        SPHL
        JMP     0H
MSG1    DB      '   CASSETTE TO DISK ROUTINE $'
MSG2    DB      ' END OF JOB
*
MSG3    DB      ' DISK I/O ERROR ON BDOS $'
MSG4    DB      ' TRANSFER FROM CASSETTE COMPLETE $'
MSG6    DB      ' PRESS ANY KEY TO RUN $'
SETUP   LXI     D,FCB
        MVI     C,CREATE <-- OR OPEN
        CALL    FILEOP
        CPI     255
        JNZ     OPNOK
        MVI     B,1
        CALL    ERR
OPNOK   XRA     A
        STA     FCBCR
        RET
DISK    PUSH H! PUSH D! PUSH B
        LXI     D,FCB
        MVI     C,WRITE <-- OR READ
        CALL    BDOS
        POP B! POP D! POP H
        CPI     0
        RZ
        CPI     1
        JZ      FINIS
        MVI     B,2
        CALL    ERR
FCBDN   EQU     FCB+0
FCBFN   EQU     FCB+1
FCBFT   EQU     FCB+9
FCBRL   EQU     FCB+12
FCB13   EQU     FCB+13
FCB14   EQU     FCB+14
FCBRC   EQU     FCB+15
FCBCR   EQU     FCB+32
FBBLN   EQU     FCB+33
DELIM   EQU     FCB+34
        END     ; STOP RUN
*
```



Implementing a tiny interpreter with a CP/M-flavored C

BY LES HANCOCK
Software Products Group
Yourdon Inc.
1133 Sixth Avenue
New York, N.Y. 10036

CP/M users finally have a compiler for the C language. It's a subset of the C defined at Bell Labs, [1] but a full and proper subset, perfectly adequate for most systems' work. It was written by Leon Zolman, alias BD Software. Although he introduces the compiler as an "educational tool" in his user's manual, I know at least one software designer who's making serious professional use of it already. This article shows how it can be used as an implementation language on the 8080/Z80.

Last summer was quite a compiler season: a copy of the BDS ccompiler came to me for pre-release testing a few days before I was due to leave for the Denver SIGPLAN symposium on computer construction. The obvious thing to do was to pick up some hot tips in Denver, then test Zolman's C by using it to implement a simple compiler. I believe it was just after coffee break on the first morning of the symposium, while Kari-Jouko Räihä was explaining *Dynamic Allocation of Space for Attribute Instances in Multi-Pass Evaluators of Attribute Grammars*, that I began to rethink this plan. What I finally did write wasn't a compiler, it was an interpreter almost as tiny as my humbled ego. Maybe the results will be interesting to those of you who (like me) are less than expert at language implementation. At any rate the interpreter puts BDS C on its mettle.

My first job was to specify the language. It's best to do this by building an LALR(1) grammar that can be elaborated into a parser. But I cut my teeth on comic books, not on syntactic structures, so I had to make do with a list of rules:

- A program consists of one or more lines, each ending in a newline or an end-of-file mark. Any line may begin with a label.
- The character set of the language includes the usual numerals and letters (upper and lower case), plus the special characters.

```
+ - / * : ; ? ! " ` ' < > { }
```

These symbols are used to represent tokens which fall into six categories: identifiers, labels, constants, strings, operators, and keywords.

- *Identifiers* (variable names) are single lower-case letters. They may be assigned numeric values.
- *Labels* (line labels) are single upper-case letters. They may not be assigned values by the programmer.
- *Constants* are signed integers from -32768 through 32767.
- Anything inside a pair of double quotes is a *string.*
- The *operators* take action on their *operands.* An operand may be a label, an identifier or a constant. The operators are:

+ - * /	the usual arithmetic operators
=	assignment
== < > <= >= <>	relational operators; if the result is true the value returned is 1, else 0
;	prints its argument
:	prints argument on a new line
-> A	branches to line labeled A
""	encloses a string
`	opens a comment
'	closes a comment

- The *keywords* provoke events but do not take operands. They are:

{ EEE } SSS	if-then: if expression EEE is evaluated to nonzero, executes the statement SSS, which may contain another if-then
?	waits for numeric input from user's console
!	halts execution

- Expressions are evaluated right to left. (But note that the if-then construction proceeds from left to right, though the expression in each if-clause is evaluated right to left.)
- Blanks and comments are ignored unless they occur in strings.
- The operators -> : ; and the keyword ! terminate evaluation of a statement.
- Only integer variables are permitted.
- Comments nest.

Now we know as much about the target language as the interpreter does, so we should be able to interpret a program ourselves. Let's try this one:

```
A : "enter a number (0 to stop)"
  { 0 == x = ? } !
  : "its square is "
  ; x * x
  -> A
```

We can easily execute it by eye. The first line, labeled *A*, prints a string that tells the user what to enter. The second line contains an if-then construction. Evaluating the *if* clause right to left, we begin with the keyword ?, which gets a number from the user's console. The = operator assigns that value to the identifier *x*. The == operator compares the value of *x* with the constant 0. If this comparison works – in other words, if the user entered a zero – the result of the expression is 1, which causes the *then* statement to be executed. That statement consists of the keyword !, which halts the program. But if *x* is anything except 0 the comparison fails and the program continues on to the next line, printing *"its square is"* followed by the value of *x* multiplied by itself. In the last line the -> operator causes control to pass back to the line which begins with *A*.

The interpreter works by doing exactly what we've just done: scan a line, evaluate it, scan another line, evaluate that, and so on. Scanning means reading the symbols and recognizing the tokens they stand for. Some tokens can be recognized immediately: + always denotes the ADD operator, *x* an identifier, *X* a label. Others take more analysis. For example, when scanning turns up the symbol - the token it represents may be SUB (for subtraction) or NEG (negation). The - may even be half of the GOTO symbol ->. Context tells us what's what: if the - symbol immediately follows an operator the token must be NEG; if it's followed by a > the token is GOTO; if it follows an identifier or constant and precedes another identifier or constant, the token has to be SUB. Once all the tokens in a given line have been recognized, the line can be evaluated by following the rules of action set up for each operator and keyword.

So the interpreter needs two main moving parts, a *scanner* and an *evaluator*. A *prescanner* would be nice too – it could strip out blanks and comments and make the source code more compact.

That's exactly how the interpreter was designed. Its driving function begins by calling out the prescanner, which filters the ASCII input file and pours the results into the *program buffer*. Then the scanner and evaluator are called to process the filtered code. The scanner gets the next token from the program buffer and disposes of it in one of six possible ways:

1. *operators* go into an *operator stack*
2. *constants* go into a *value stack*
3. *labels* are copied from the symbol table into the value stack
4. *identifiers* (variable names) are copied from the *symbol table* into the value stack; if an identifier isn't yet in the symbol table, the scanner puts it there
5. *strings* go into a *string buffer;* an entry in the value stack points to each string
6. *keywords* are dealt with immediately.

The evaluator never sees the ASCII symbols in the program buffer. Its job is to evaluate a single line on the basis of the tokens it finds in the operator stack, value stack, symbol table and string buffer. It pops operators and values from their respective stacks and handles them according to the rules that define the tiny language.

Before looking at the C functions that run the interpreter we'd better understand its four data structures in more detail. The *program buffer* is a character array that contains the program's ASCII source code as filtered through the prescanner. Here's how the square-your-number program looks in its buffer:

```
:"enter a number (0 to stop)"\n{0==x=?}!\n;"
                  its square is "\n;x*x\n->A
```

Note that the carriage-return/linefeeds have been replaced by the C newline (represented here in conventional C notation as */n*) and that the initial line label *A* has disappeared into the symbol table, where it's equipped with a pointer aimed at the first character in the program buffer. Incidentally, the last character in the buffer is always a CP/M end-of-file (0x1a), not shown in the example.

The *symbol table* contains one entry for every label or identifier encountered in the process of scanning (or prescanning) the source code. The symbol table *symtab* is an array of structures each of which contains a one-character symbol and its associated value, either an integer (for identifiers) or a pointer to the head of some line in the program buffer (for line labels):

```
SYMBOLS {
     char symbol;   /* l.c. for IDENT, u.c. for LABEL */
     union {
          int integer;   /* if ident is IDENT */
          char *line;    /* if ident is LABEL */
     } quid;
} symtab[STAKSIZE], *symptr;
```

The symbol table pointer *symptr* will always point to the next available element of *symtab*.

Unlike the other data structures, the symbol table never shrinks; it can grow as the scanner finds new symbols, and its integer component can change as variables get new values, but nothing's ever elided. The operator stack, value stack and string buffer are rebuilt from scratch every time a line (or an if-clause or a then-clause) is scanned.

The *value stack* is an array of structures topped off by a stack pointer – tokens may be pushed onto it or popped off it by incrementing or decrementing *valsptr*. In the first draft of the interpreter this data structure was called *operandstack*, which is more exact than *valuestack* but too easy to confuse with *operatorstack*, its opposite number. At any rate, the structures look like this:

```
VALUES {
     int type;   /* may be IDENT, LABEL, CONSTANT, STRING */
     union {
          SYMBOLS *ptrsym;   /* if type is IDENT or LABEL */
          char *substring;   /* if type is STRING */
     } quod;
     int value;   /* if type is IDENT or CONSTANT */
} valuestack[STAKSIZE], *valsptr;
```

The *type* of an entry determines which if any of *quod*'s members are useful. When an operand is an identifier or a line label *quod*'s value is *ptrsym*, a pointer to the appropriate symbol table entry. When it's a string *quod*'s value is *substring*, a pointer to the string's location in the array *strings*. For constants *quod* has no use at all.

Where operands have a numeric value an entry is made in *value*, which otherwise contains garbage.[2]

The *operator stack* is just an integer array with a stack pointer:

```
int operatorstack[STAKSIZE], *opsptr;
```

Every operator that can go into the stack has a numeric code:

ASCII symbol	mnemonic	code	comments
+	PLUS	1	
–	MINUS	2	
*	TIMES	3	
/	DIV	4	
=	GETS	5	assignment
:	PRINT	6	print with newline
;	PRT	7	print sans newline
–>	GOTO	8	
==	EQ	9	logical equivalence
>	GT	10	greater than
>=	GE	11	greater than or equal to
<	LT	12	less than
<=	LE	13	less than or equal to
<>	NE	14	logical inequality

The numbers can be anything so long as there's no duplication.

The *string buffer* is the simplest of all the data structures:

```
char strings[LINE], *strptr;
```

As strings are encountered in the scan they go into *strings*. The pointer *strptr* always indicates the next available element in this array.

So much for the data structures used to implement the interpreter. You may have noticed that each one is associated with a pointer. Like most other up-to-date implementation languages, C leans heavily on pointers and recursion. We'll come to the recursion now as we look at the interpreter's main functional modules, *main*, *prescan*, *scan* and *evaluate*.

main is the driving function. It follows this outline:

```
initialize stop/go flag, bugger pointer, symbol table pointer
prompt for input file name
prescan the file
if no errors so far,
    reinitialize buffer pointer
    while the stop/go flag is GREEN,
        clear the value stack
        clear the operator stack
        clear the string buffer
        move the scanner through the program buffer till it
          hits the end of the current line or the end of the
          buffer, or encounters a STOP or an error
        if the scanner has reached the end of the buffer, set the stop/
          go flag to RED
        call the evaluator
```

prescan opens the input file and calls the supporing function *readchar* to get new characters in sequence until end-of-file is reached. *readchar* is delegated the job of ignoring comments and linefeeds. Linefeeds are easy enough, but we've stipulated that comments nest. A global counter called *incomment* is used to keep track of how deeply we're nested in comments at any given moment. There are two kinds of comment tokens, LCOMMENT (to open the comment) and RCOMMENT (to close it). *readchar* follows this strategy:

```
get the next non-LINEFEED character from the file if it's an LCOM-
MENT,
    increment incomment
else if it's an RCOMMENT,
    decrement incomment
if incomment is nonzero or if the character's an RCOMMENT,
    call readchar recursively
else return the character
```

Recursion provides a neat solution to the nesting problem.

prescan itself has the job of eliminating blanks and of putting labels into the symbol table. Since blanks in strings must be preserved, the local variable *instring* is used as a flag. Strings don't nest, so *instring* need take only two possible values, YES and NO. *prescan's* internals aren't very interesting. When it finds an upper case alphabetic character at the head of a line it assumes it has a label and installs the character in the symbol table along with the current value of the program buffer pointer. Since the labels are no longer needed in the source code, they're not read into the program buffer. All other characters, including labels which don't occur at the head of a line, do go in.

scan is the biggest and trickiest function of the lot—too big, really. Its argument is an ASCII character from the program buffer. A long series of case statements are used to recognize the argument as a keyword or as an operator (or as the first character of an operator's symbol); if that fails, it should be an operand. Most of the operators get minimal processing: things like PLUS or TIMES or DIV simply go onto the operator stack. But the keywords are handled entirely by the scanner, and one of them, IF, exercises *scan* to its fullest.

Consider the if-then construction as a whole. Where we have a line like

```
{ 0 == x = ? } !
```

we want to parse *and evaluate* the conditional clause before going on to parse and evaluate the statement that will be executed if the condition is true. This calls for recursion. Look at the '{' case of *scan* in the C source code. When *scan* meets this symbol, it clears the value stack and the operator stack and calls itself just as it gets called in *main*. Successive characters are scanned until the matching '}' is reached. The effect of the '}' case is simply to return a zero, which is the signal *scan* uses to tell its calling function that it's reached a stopping place. At this point control passes back to the first invocation of *scan*, which checks to be sure that a '}' is in fact what stopped the scanning process. If so, it calls *evaluate* to evaluate the part of the line just scanned—that is, the part inside the if-clause. The function *evaluate* returns the numeric result of its evaulation. Should that result be nonzero, *scan* clears the stacks again and returns 1, which tells the calling function to go ahead and finish scanning the current line. If the result is zero, *scan* pushes *bufptr* ahead to the next newline, skipping the statement following the THEN. In this little recursive drama *scan* briefly plays the part of *main*.[3]

Recognizing operands is a fairly straightforward business. Except in strings, an upper case letter must be a label, a lower

case letter must be an identifier, and a digit can only be part of an integer constant. Since labels at the head of a line have already been dealt with by the prescanner, the scanner should find labels only after GOTOs. Whenever a GOTO turns up we peek ahead one character to be sure there's a label coming: as soon as we reach the label we look back one character to be sure we had a GOTO.

When the scanner encounters a label it calls *idlookup* to see whether the label's in the symbol table. If not, that's an error. Otherwise an entry is made in the value stack, setting the *type* to LABEL and setting *quod.ptrsym* to point to the label's symbol table entry.

The story's somewhat different for identifiers. It's perfectly OK to find that an identifier is not yet in the symbol table. When that happens the scanner calls *idinstall* to put it there. The entry made in the value stack will have a *type* of IDENT, a *quod.ptrsym* pointing to the symbol table entry, and a *value* equal to whatever was contained in the symbol table's *integer* field.

When the scanner meets a constant it accumulates the consant's integer value with the help of the library routine *atoi*.[4] This goes into the value stack's *value* member, with the *type* set to CONSTANT. Negative numbers are accumulated by a recursive call from the - case.

Eventually we scan up to a newline, at which point *main* passes control on the *evaluate*. Though it's fairly long, *evaluate* is a simple function. If it sees a RED flag it simply returns, leaving *main* to stop the program. Otherwise it pops the operator stack once and the value stack twice. If either stack is empty, evaluation is finished. If not, *evaluate* goes to the case statement that matches the operator and carries out the action it finds there. Most operators are binary—that is, they take two operands. All binary operators do essentially the same thing: they combine the two operands in some way (add them, divide them, compare them), then put the result back onto the value stack as a CONSTANT. One binary operator, GETS, stands apart. Its left operand must be of type IDENT. Otherwise it's like any other binary operator, which makes expressions like

```
0 == x = ?
```

perfectly legal.

The unary operators are PRINT, PRT and GOTO. One of the rules we used to define the language being interpreted was that these operators terminate execution of a statement. In fact they jump right out of the evaluator, using C's own *goto*. There are nicer ways to do this—for example, by setting a local flag to be checked in the *while*—but I did want to wring out the compiler, and I've always had a salacious yen to use C's *goto*.

The interpreter's GOTO operator works by setting the program buffer pointer to the address recorded in the label's symbol table entry. When *evaluate* ends, *main* tells *scan* to start scanning the line that begins at that address. It's very much like the 8080 jump instruction, which loads the jump address into the program counter.

Examples are included here to show how BDS C compiles and runs the tiny interpreter. *cc1* and *cc2* are the two halves of the compiler. They generate relocatable object files which *clink* links with libraries to produce absolute 8080 code.

According to Leor Zolman, his C trades object code efficiency at runtime for structure and clarity during program development. This is certainly true in a sense—the interpreter's .COM file size is 12K bytes—but memory's cheap these days, and as for time, I hate to think how long it would've taken me to code this interpreter in 8080 mnemonics. Where speed and space are really critical, C can be used to prove the logic of a system which will eventually be written in assembler.

The BDS C compiler, with libraries, linking loader and good documentation, is available from tiny C associates (201/671-2296) and Lifeboat Associates (212/580-0082) for about a hundred dollars. I recommend it.

Author's Self-Portrait

Footnotes

(1) The best definition is Appendix A of Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, (Prentice-Hall, 1978).

(2) I know: *value* could go into *quod*. There's a reason I chose not to do that, but I can't remember what it was. Also: why *quod* and not another *quid*? It's one of the BDS compiler's few serious restrictions. To quote the user's manual, "Names of members and tags in the structure definitions *cannot* be the same as any regular local variable names, nor can more than one structure or union per function use a given identfier as a member or tag name."

(3) See line 18 of the test program *16.tst* for an example of deeper recursion in if-then's.

(4) Zolman provides the usual C runtime routines, plus quite a few useful library functions.

listing

```
B>type x.c

/* * * * * * * * * * * * * * * * * * * * * * *
** Tiny interpreter, version of 10/23/1979.
**      by Les Hancock
**      New York City
**
** Operators as follows:
**
**       + - / * as usual
**       > >= < <= <> also as usual
**       : prints its argument, then a newline
**       ; prints its argument sans newline
**       ->A branches to line label A
**
** Keywords:
**
**       ?  prompts for numeric input
**       ! halts execution
**       { expression } statement   executes statement
**             if expression is evaluated to nonzero
```

```
**      " opens and closes a string
**      ` opens a comment
**      ' closes a comment (nesting OK)
**
** All identifiers must be single lower-case chars.
** Line labels must be single upper-case chars.
**
*/

/*
** Value types
*/
#define IDENT 0
#define LABEL 1
#define CONSTANT 2
#define STRING 3

/*
** Operators
*/
#define PLUS 1
#define MINUS 2
#define TIMES 3
#define DIV 4
#define GETS 5
#define PRINT 6   /* print with newline */
#define PRT 7   /* print sans newline */
#define GOTO 8
#define EQ 9
#define GT 10
#define GE 11
#define LT 12
#define LE 13
#define NE 14

/*
** Miscellaneous parameters
*/
#define SYMBOLS struct a
#define VALUES struct b
#define LINE 81   /* chars in biggest input line */
#define STAKSIZE 24   /* max tokens in symbol table, value stack,
#define BUFSIZE 1000   /* max chars in the program buffer */
#define LCOMMENT '`'
#define RCOMMENT '\''
#define LINEFEED 10
#define NEWLINE 13   /* i.e., carriage return */
#define END 0x1a   /* EOF for CP/M */
#define YES 1
#define GREEN YES   /* for use with <flag> */
#define NO 0
#define RED NO   /* for use with <flag> */

/*
** Error codes
*/
#define LVALERR 1   /* lvalue error */
#define FOPNERR 2   /* file open error */
#define OVRBUFERR 3   /* program buffer overflow */
#define IFERR 6   /* IF - THEN error */
#define NOLABLERR 7   /* missing label */
#define IDLABERR 8   /* attempt to use a label as lvalue */

/*
** Symbol table, symbol table pointer
*/
SYMBOLS {
   char symbol;   /* may be IDENT or LABEL */
   union {
      int integer;   /* if symbol is IDENT */
      char *line;   /* if symbol is LABEL */
   } guid;
} symtab[STAKSIZE], *symptr;

/*
** Value stack, value stack pointer
*/
VALUES {
   int type;   /* may be IDENT, LABEL, CONSTANT, STRING */
   union {
      SYMBOLS *ptrsym;   /* if type is IDENT or LABEL */
      char *substring;   /* if type is STRING */
   } quod;
   int value;   /* ignore if type is LABEL or STRING */
} valuestack[STAKSIZE], *valsptr;

/*
** progbuf = program buffer
** bufptr = program buffer pointer
** iobuf is used by <getc()>
** strings = string buffer
** strptr = string buffer pointer
*/
char progbuf[BUFSIZE], *bufptr, iobuf[134], strings[LINE], *strptr;

/*
** Operator stack, operator stack pointer
*/
int operatorstack[STAKSIZE], *opsptr;

/*
** flags:
**       incomment = set if prescanner is inside a comment
**       flag halts program when it turns RED
*/
int incomment;
char flag;
/* * * * * * * * * * * * * * * *
**
**
** Print error message and set flag
** to stop further processing.
*/
int error(errno)
int errno;
{
   switch(errno) {
   case LVALERR:
      printf("\n= wants lvalue");
      break;
   case FOPNERR:
      printf("\nfile won't open");
      break;
   case OVRBUFERR:
      printf("\noverflows input buffer");
      break;
   case IFERR:
      printf("\nif-clause error");
      break;
   case NOLABLERR:
      printf("\nneed label");
      break;
   case IDLABERR:
      printf("\nlabel must follow ->");
   }
```

```
    flag = RED;    /* halt program */
}

/*
** At this point the scanner has filled the operand
** and operator stacks.  <evaluate> pops operators and
** operands and carries out the prescribed actions.
*/
int evaluate()
{
    VALUES *val;

    if (flag == RED)
        return (0);
    while (--opsptr >= operatorstack && valsptr > valuestack) {
        val = --valsptr;
        --valsptr;
        switch (*opsptr) {
        case PLUS:    /* binary operators */
        case MINUS:
        case TIMES:
        case DIV:
        case EQ:
        case GT:
        case GE:
        case LT:
        case LE:
        case NE:
            switch (*opsptr) {
            case PLUS:
                valsptr->value = valsptr->value + val->value;
                break;
            case MINUS:
                valsptr->value = valsptr->value - val->value;
                break;
            case TIMES:
                valsptr->value = valsptr->value * val->value;
                break;
            case DIV:
                valsptr->value = valsptr->value / val->value;
                break;
            case EQ:
                if (valsptr->value == val->value)
                    valsptr->value = 1;
                else
                    valsptr->value = 0;
                break;
            case GT:
                if (valsptr->value > val->value)
                    valsptr->value = 1;
                else
                    valsptr->value = 0;
                break;
            case GE:
                if (valsptr->value >= val->value)
                    valsptr->value = 1;
                else
                    valsptr->value = 0;
                break;
            case LT:
                if (valsptr->value < val->value)
                    valsptr->value = 1;
                else
                    valsptr->value = 0;
                break;
            case LE:
                if (valsptr->value <= val->value)
                    valsptr->value = 1;
                else
                    valsptr->value = 0;
                break;
            case NE:
                if (valsptr->value != val->value)
                    valsptr->value = 1;
                else
                    valsptr->value = 0;
                }
                valsptr++->type = CONSTANT;    /* ++ puts valsptr back on stack */
                break;
            case GETS:    /* special binary operator */
                if (valsptr->type == IDENT) {    /* must be lvalue */
                    valsptr->value = valsptr->quod.ptrsym->quid.inteser
                        = val->value;
                    ++valsptr;
                }
                else
                    error(LVALERR);
                break;
            default:    /* unary operators */
                switch  (*opsptr) {
                case PRINT:    /* print with newline */
                    if (val->type == CONSTANT || val->type == IDENT)
                        printf("\n%d", val->value);
                    else    /* must be STRING */
                        printf("\n%s", val->quod.substring);
                    break;
                case PRT:    /* print sans newline */
                    if (val->type == CONSTANT || val->type == IDENT)
                        printf("%d", val->value);
                    else
                        printf("%s", val->quod.substring);
                    break;
                case GOTO:
                    if (val->type == LABEL)    /* must go to label */
                        bufptr = val->quod.ptrsym->quid.line;
                    else
                        error(NOLABLERR);
                }
                goto getout;
            }
        }
getout:    return ((--valsptr)->value);    /* return value */;
}

/*
** Install an identifier in the symbol table.
*/
SYMBOLS *idinstall(c)
char c;
{
    symptr->symbol = c;
    return (symptr++);    /* return pointer to the new entry */
}

/*
** Look up identifier in symbol table (simple linear
** search).  If there, return pointer to the entry;
** if not, return null pointer.
*/
SYMBOLS *idlookup(c)
char c;
{
    SYMBOLS *p;

    p = symtab;
    while (p < symptr) {  /* <symptr> is current top of table */
```

```
        if (p->symbol == c)
            return (p);
        p++;
    }
    return (0);
}

/*
** Driver.
*/
main()
{
    char c, name[20];

    flag = GREEN;  /* a RED <flag> halts the program */
    bufptr = progbuf;   /* set buffer ptr to head of buffer */
    symptr = symtab;   /* set symbol table pointer */
    printf("\nInput file: ");
    if (prescan(gets(name))) {  /* source file read successfully */
        bufptr = progbuf;  /* reset buffer pointer */
        while (flag == GREEN) {
            valsptr = valuestack;  /* set value stack pointer */
            opsptr = operatorstack;  /* set operator stack ptr */
            strptr = strings;   /* set string pointer */
            /* Scan a line for tokens.  NEWLINE ends line,
               END ends program. */
            while ((c = *bufptr++) != NEWLINE && c != END
                && flag == GREEN && scan(c))
                    ;
            if (c == END)
                flag = RED;
            evaluate();
        }
    }
/*
** Read the contents of named file into the
** input buffer.  Strip off labels, comments and all
** blanks not inside "".
*/
int prescan(name)
char *name;
{
    char c;
    int filedescr, instring;

    if ((filedescr = fopen(name, iobuf)) < 0) { /* open for buffer input */
        error(FOPNERR);
        return (0);   /* report failure to open file */
    }
    incomment = 0;   /* see <readchar> */
    instring = NO;
    do {
        c = readchar();   /* get next non-LINEFEED, non-comment from file */
        if (c == ' ' && !instring)   /* ignore blanks not in strings */
            continue;
        if (c == '"')
            instring ^= YES;   /* if YES make NO, if NO make YES */
        if (bufptr < &progbuf[BUFSIZE]) {
            if (isupper(c) && (bufptr == progbuf || *(bufptr - 1) == NEWLINE))
                symptr->symbol = c;  /* put label in symbol table */
                symptr->quid.line = bufptr; /* put pointer in symbol table */
                ++symptr;
            }
            else   /* it wasn't a label */
                *bufptr++ = c;
        }
        else {
            close(filedescr);
            error(OVRBUFERR);
            return (0);
        }
    } while (c != END);
    close (filedescr);   /* close the input file */
    return (1);   /* report success */
}

/*
** Read a new (buffered) character from the
** input file, ignoring linefeeds and comments.
**
** Note that <incomment> indicates how deeply we're
** nested in comments.
*/
readchar()
{
    char c;

    while ((c = getc(iobuf)) == LINEFEED)
        ;
    if (c == LCOMMENT)
        ++incomment;
    else if (c == RCOMMENT)
        --incomment;
    if (incomment || c == RCOMMENT)
        return (readchar());
    else
        return (c);
}
/*
** Scan a line for tokens.  Carry out keyword actions
**    (IF, THEN, STOP, INPUT).
*/
int scan(c)
char c;
{
    switch (c) {
    case '{':   /* keyword IF */
        valsptr = valuestack;
        opsptr = operatorstack;
        while ((c = *bufptr++) != NEWLINE && c != END
            && flag == GREEN && scan(c))
            ;
        if (c == NEWLINE || c == END || flag != GREEN) {
            error(IFERR);
            return (0);
        }
        if (evaluate()) {   /* finish the line */
            valsptr = valuestack;
            opsptr = operatorstack;
            return (1);
        }
        while (*bufptr++ != NEWLINE)  /* skip to next line */
            ;
        return (0);
    case '}':   /* keyword THEN */
        return (0);
    case '!':   /* keyword STOP */
        flag = RED;   /* this will stop <evaluate()> and <main()> */
        return (0);   /* no more scanning this line */
    case '?':   /* keyword INPUT */
        printf("\n?");   /* print prompt character */
        scanf("%d", &valsptr++->value);   /* get input */
        break;
    case '"':   /* keyword " */
        valsptr->type = STRING;
        valsptr++->quod.substring = strptr;
```

```
        while (*bufptr != '"')
            *strptr++ = *bufptr++;
        *strptr++ = '\0';   /* terminate substring with null byte */
        ++bufptr;   /* skip the " */
        break;
    case '+':
        *opsptr++ = PLUS;
        break;
    case '-':   /* could be MINUS (-) or NEG or GOTO (->) */
        if (*bufptr == '>') {    /* it's GOTO */
            if (isupper(*++bufptr))  /* next token must be LABEL */
                *opsptr++ = GOTO;
            else
                error(NOLABLERR);
        }
        else if (!islower(*(bufptr - 2)) /* char preceding - not identifier */
            && !isdigit(*(bufptr - 2)) /* and not a digit */
            && isdigit(*bufptr)) {   /* it's NEG */
            scan(*bufptr);   /* get the integer */
        }
        else    /* it's MINUS */
            *opsptr++ = MINUS;
        break;
    case '*':
        *opsptr++ = TIMES;
        break;
    case '/':
        *opsptr++ = DIV;
        break;
    case '<':
        if (*bufptr == '=') {
            *opsptr++ = LE;
            ++bufptr;
        }
        else if (*bufptr == '>') {
            *opsptr++ = NE;
            ++bufptr;
        }
        else
            *opsptr++ = LT;
        break;
    case '=':
        if (*bufptr == '=') {
            *opsptr++ = EQ;
            ++bufptr;
        }
        else
            *opsptr++ = GETS;
        break;
    case '!':   /* print with newline */
        *opsptr++ = PRINT;
        break;
    case ';':   /* print sans newline */
        *opsptr++ = PRT;
        break;
    case '>':   /* could be GE (a >= b) or GT (a > b) */
        if (*bufptr == '=') {
            *opsptr++ = GE;
            ++bufptr;
        }
        else
            *opsptr++ = GT;
        break;
    default: /* we have an operand (LABEL, IDENT, CONSTANT) or an error */
        if (isupper(c)) {   /* label */
            if (*(bufptr - 2) == '>') {   /* must follow GOTO */
                if ((valsptr->quod.ptrsym = idlookup(c)))    /* in symtab */
                    valsptr++->type = LABEL;
                else
                    error(NOLABLERR);
            }
            else
                error(IDLABERR);
        }
        else if (islower(c)) {   /* identifier */
            /* if not in symbol table, install it */
            if ((valsptr->quod.ptrsym = idlookup(c)) == 0)
                valsptr->quod.ptrsym = idinstall(c);
            valsptr->value = valsptr->quod.ptrsym->quid.integer;
            valsptr++->type = IDENT;
        }
        else if (isdigit(c)) {   /* constant */
            /* accumulate value, put in operand stack.
               Note that if this is a recursive call from
               case '-', backing up one char (in atoi(bufptr - 1))
               includes the - and makes the number negative.
               Otherwise <scan> will have been called with
               argument *bufptr++, so backing up one is necessary
               to get the first digit of a positive number. */
            valsptr->value = atoi(bufptr - 1);
            valsptr++->type = CONSTANT;
            while (isdigit(*bufptr))
                ++bufptr;   /* advance to next nondigit */
        }
        else {   /* error */
            printf("\ncan't scan %c", c);
            error(0);  /* no further message needed */
        }
    }
    return (1);
}
```

B.

```
B.cc1 x.c
BD SOFTWARE C COMPILER V1.2  (PART I)
COPYRIGHT (C) 1979 BY LEOR ZOLMAN

B.cc2 x
BD SOFTWARE C COMPILER V1.2 (PART II)

B.clink x
BD SOFTWARE C LINKER   V1.2

B.x

Input file: 16.tst

We start with 23 matches.
Taking from 1 to 3, try to leave
your opponent with the last match.

! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !

your move
?1

! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !

I take 3

! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !

your move
?4
```

```
take 1, 2 or 3 only

your move
?3
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !
I take 1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !
your move
?3
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !
I take 1
! ! ! ! ! ! ! ! ! ! ! ! ! !
your move
?1
! ! ! ! ! ! ! ! ! ! ! !
I take 3
! ! ! ! ! ! ! !
your move
?3
! ! ! !
I take 3
!
your move
?0
take 1, 2 or 3 only

your move
?1
I WIN
```

```
B.type 16.tst
: "We start with 23 matches."
: "Taking from 1 to 3, try to leave"
: "your opponent with the last match."
: " "
x = 23
z = 1 ` when this flag is 0, computer takes turn '
  ` print x matches '
A : "! "
  y = x
B y = y - 1
  { y == 0 } -> C
  : "! "
  -> B
C { z == 0 } -> E ` whose turn? '
  : " "
  : "your move"
  u = ?
  { u <= 3 } { u > 0 } -> D   ` not error '
  : "take 1, 2 or 3 only"   ` error '
  -> C    ` try again '
D x = x - u
  { x < 1 } -> G    ` computer wins '
  z = 0    ` computer sets next turn '
  -> A
E { x > 4 } i = 4 - u    ` computer makes move '
  { x <= 4 } i = x - 1
  { x == 1 } i = 1
  : " "
  : "I take "
  ; i
  : " "
  x = x - i
  { x < 1 } -> F    ` user wins '
  z = 1    ` user sets next turn '
  -> A
F : "YOU WIN "
  : " "
  -
G : "I WIN "
  : " "

B.
```

end

ALS-8 to CP/M File Converter

BY HOLLIS FRAMPTON
SUNY - Center for Media Study
Buffalo, New York 14214

This routine may be administered to alleviate pain. We can't be the only people who are making the transition from ALS-8 and its various cousins (Software Package #1, *et al.*) to CP/M. There's enough work in interfacing a library to the utilities of a new operating system, without battling the discrepancies between file formats as well. Reasoning that some drudgery is too grim even for system programmers, and faced with the transfer of some hundreds of files to the new system, I wrote CPCON and would like to share it.

With several embellishments, it runs in 1085 bytes plus 24 bytes worst case for the stack, and on a Sol-20 running at not quite 2 MHz converts *itself* in about 2½ seconds; about half this time is taken up by two special passes through the ALS-8 source file to handle the special case of the quoted ASCII blank (as in MVI A,' '). A few redundancies allow some of the bells and whistles to be detached with ease: a couple of these are documented in the preamble to the listing. Some *seeming* software overkill (the recurrent MVI A, – 1) deals with bizarre special cases in source files. They are there to "bullet-proof" the code, so remove them at your peril. If it seems over-commented, it is: I love to play the 'you-figure-it-out' game with other people's code, but not with my own . . . 3 days later.

A word about constraints. Our ALS-8 is the PROM version, resident from E000-FFFF hex. We're running CP/M on North Star minifloppies, with a bootload PROM at E900 and a Thinkertoys DISCUS with PROM at E000. Hence the painfully slow strategy of dumping to tape. Those with more flexible memory maps, if they can arrange to have ALS-8, CUTER and the BDOS part of CP/M simultaneously resident, will of course want to automate the whole procedure. Since both SOLOS and CUTER return tape header data in known RAM areas, it's a trivial matter to have CP/M perform automatic disk saves from tape files read in under the SOLOS/CUTER utilities. Equates in CPCON adhere strictly to the widely published labels in documentation for both SOLOS/CUTER and ALS-8.

The VDM output driver is a CRT driver that uses the VDM-1, a memory-mapped display, but partially simulates a terminal, conventionally blanking the screen and cursor on receiving ctrl-S and ctrl-A respectively. Presumably the CP/M source file conventions will be known to anyone desiring to use CPCON; for those who wish to patch it for other source files than those native to ALS-8, the specs are as follows. A *line* consists of: (1) a 'character count' byte. The count tallies all characters in the line, plus the terminating CR, *plus itself*; (2) Zero or more printable ASCII characters; and (3) a CR (*not* a CRLF). Meaningful data begins in column 6 of an assembler source file, with a blank separating it from a 4-digit decimal line number. Labels may or may not be followed by a colon; in either case the label field terminates with a blank. Labels are considered unique to 5 characters, but may contain 6; CPCON disregards label length. Opcode and operand fields terminate with blanks and may contain no imbedded blanks. The ALS-8 output formatter requires a dot as a dummy operand to pretty-print after RET, PCHL, XCHG and so on. Comment fields are separated by at least one blank from the preceding field, pretty-print with 2 blanks, and *tolerate* a leading semicolon or asterisk delimiter. Comment *lines* may be indicated only by a leading asterisk. Does that sum it up? It's hard to remember: the ALS-8 assembler supports some fairly baroque ambiguities, dating perhaps from the days of the 4004 or the 2002 or1001. Handling of string constants is summarized on the preamble.

Dear Suzanne,

Here's a fresh listing of CPCON, which converts resident source of text files in ALS-8 (SPKG1) format to CP/M format. The routine will accept anything the ALS-8 assembler finds edible, and produce an output file equivalent to a CP/M *.PRN file from ASM.COM or MAC.COM but stripped of its first 16 columns. All labels emerge followed by colons; semicolons precede comment lines and fields; tabs are appropriately inserted. The ASC pseudo-op is converted to a DB, and all ambiguities in its argument string are resolved to single-quote delimiters. ALS-8's directives COM, LIST and NLST are left undisturbed; the feature that packed them out was removed when a file written under the ALS-8 video editor suffered deletion of a macro call to something named LIST. Hand checking is urged for these cases.

Users who aren't in the habit of quoting ASCII blanks (20H) may shorten the routine by two entire passes, by deleting lines 94-106, line 141 (CALL SPFIX) and SPFIX itself (lines 404-412). Yet another pass is taken to convert comment fields in statement lines to lowercase: disable the option by deleting the CALL LCASE at line 142, and the apposite subroutine at lines 414-460. With all options enabled, CPCON reformats itself (about 24K) in less than 2 seconds. It then saves the reformatted file *twice* to tape, and offers to display the reformatted file at the console (tabs are not expanded). The very nervous, wishing more than one verification, should enter via a CALL to CCOPY; the confident may delete the option and shorten the code by omitting lines 150-167 and all of MSSG4.

Two idiosyncrasies should be noted. First: comment fields in statement lines, once detected (and proved not 'empty') are passed intact to the reformatted output. An earlier version deleted leading blanks, and succeeded in trashing most of what was intelligible in a banner headline routine that depended on a large table. Those who wish to pack out leading blanks in comment fields can insert an appropriate test in TSTCF or TSCF1. Second: the command switch in CPCON offers to reformat pure text files, a much simpler procedure which simply removes the "character count byte" (see below), the line number, and the column thereafter. It has been our custom to use an ASCII dot (.) as a "line holder" in such text files, since an empty line produces bizarre (try it!) output from the ALS-8 console TEXT command. Accordingly, entry at PTFOR tests for such "line-holders" and converts them to blanks; more than one dot is assumed to be part of an ellipsis and is passed unchanged. Anyone who has used the video text editor in ALS-8 as a convenient alternative to ED.COM may experience difficulty with this protocol where they have written input files for text processors, since most such software uses leading dots in column 1 to flag a command. Those wishing to reformat only assembly language files may make entry at ASFOR . . . but be sure you test for a non-existent source file or CPCON will attempt to reformat all of RAM.

A few notes about system calls and file-structure assumptions. Those working with the PROM version of ALS-8, which resides at EOOO-FFFF and SOLOS/CUTER, those superdreadnoughts of the late Processor Technology Corp., will be able to use this program as-is. ALS-8 expects to find RAM at D000-DFFF and stores the 5-character name of it's current file starting at D000 (FNAME). Thereafter, binary pointers to start-of-file and end-of-file are stored at D005 and D007. As a convenience, ALS-8 returns the current filename and its address parameters to the console output driver on entry at FILE (E5BO) without checking file structure. EORMS (E060) is the warm entry point to ALS-8 (signs on with "READY".) The VDM console output driver (FE77) accepts a printable character in register B and trashes the accumulator; a ctrl-Z blanks the screen, assuming 1K of display RAM starting at CC00. The SOLOS/CUTER input routine behaves like the specs for CONIN in CP/M, returning with the zero flag set on no input, else delivering the character in A. Tape routines in SOLOS/CUTER assume a header block, starting address and byte count: patch them for your own drivers ad lib. The persent version dumps formatted files to tape at 1200 baud, in CUTS format, under the filename known to ALS-8 and typed "A" or "T".

For those who have not undergone the pain of working with the ALS-8/SPKG1 format, this is it. Files are line- oriented and line-numbered. A formal line consists of 1-65 lines (there's a terminal-width switch in command mode, but EDIT truncates all lines on exit). Each line consists of: (1) a "character count" byte. The count of characters in the line *includes itself*; (2) a 4-character ASCII decimal line number; (3) a dummy character, normally a blank, ignored by the assembler; (4) a string of printables; (5) an ASCII carriage return (CR: = ODH), the end-of-line marker. Thus an empty line contains only a "character count" byte, which evaluates to a binary 1 (that is itself). Therefore hex 01 is the end-of-file character. The far simpler CP/M file structure assumes no line numbers, inserts a CR and a LF after each line, and posts an arbitrary ctrl-Z as an end-of-file mark on all but *.COM files. Incomplete pages under CP/M are filled with ctrl-Z, a convention to which CPCON conforms.

CPCON was built for systems (5¼" North Star and 8" Thinkertoys) whose boot load PROMs overlap the ALS-8 address space. Hence the download to tape. Then change circuit boards, and read back the tape files under SOLOS/CUTER. The tape loader returns a hexadecimal filesize parameter: convert it to decimal and call it "bb". Warmboot CP/M and type "SAVE bb FILENAME.TYP" and CR. It's a lot easier if CP/M and your version of ALS-8/SPKG1 and SOLOS/CUTER can be made co-resident, but ours can't, so we didn't bother to write it that way. No trouble about asking CP/M to pick up the parameters, build a new FCB (file control block), and autosave the source: see the manual "CP/M Interface Guide," a model of lucidity which allows the programmer to treat an operating system like a device with a few control ports.

Regards,
Hollis Frampton Digital Arts Lab

```
0000 ************************************************
0001 *                                              *
0002 *  CPCON.ASM: AN ALS-8 TO CP/M FILE CONVERTER  *
0003 *                                              *
0004 ************************************************
0005 *
0006 * CONSTANTS
0007 *
0008 CR:     EQU    0DH
0009 CTRLZ:  EQU    1AH     ;CP/M EOF
0010 ESC:    EQU    1BH     ;PANIC BUTTON
0011 ETX:    EQU    3
0012 FILL:   EQU    0       ;DUMMY BYTE FOR ' ' CASE
0013 LF:     EQU    0AH
0014 QUOTE:  EQU    27H     ;ASCII <'>
0015 TAB:    EQU    09H     ;ASCII <HT>
0016 *
0017 * SOLOS/CUTER EQUATES
0018 *
0019 BLOCK:  EQU    0C823H
0020 FNUMF:  EQU    0C854H
0021 HTYPE:  EQU    0C822H  ;TYPE BYTE IN SOLOS TAPE HEADER
0022 LOADR:  EQU    0C825H
0023 SINP:   EQU    0C01FH  ;SOLOS SYSTEM INPUT DRIVER
0024 THEAD:  EQU    0C81CH
0025 UNIT1:  EQU    80H
0026 WTAPE:  EQU    0C77FH
0027 *
0028 * ALS-8 EQUATES
0029 *
0030 BOFP:   EQU    0D005H  ;BEGINNING-OF-FILE POINTER
0031 EOF:    EQU    01H     ;ALS-8 END-OF-FILE MARK
0032 EOFP:   EQU    BOFP+2  ;END-OF-FILE POINTER
0033 EORMS:  EQU    0E060H  ;ALS-8 SIGN-ON ENTRY PT
0034 FILE:   EQU    0E5B0H  ;RETURNS CURRENT FILE PARMS
0035 FNAME:  EQU    0D000H  ;CURRENT FILE NAME
0036 VDM:    EQU    0FE77H  ;ALS-8 CRT OUTPUT DRIVER
0037 *
0038         ORG    100H    ;PUT IT WHERE YOU WANT IT
0039 *
0040 CPCON:  LXI    H,0     ;CLEAR TO RECEIVE...
0041         DAD    SP      ;...CALLING SP
0042         SHLD   SYSSP   ;SAVE IT
0043         LXI    SP,STACK  ;FORCE PROGRAM STACK
0044         LHLD   BOFP    ;GET START OF FILE PTR
0045         MOV    A,M     ;GET FIRST CHAR
0046         CPI    EOF     ;FILE EMPTY?
0047         JZ     ERROR   ;ABORT & REPORT
0048         STA    CCBYT   ;ELSE SAVE CHARACTER COUNT BYTE
0049         SHLD   LINAD   ;START ADDR OF CURRENT LINE
0050         SHLD   DEST    ;DEST FOR NEW SOURCE FILE
0051         CALL   GTSRC   ;FIND START OF NEXT SOURCE LINE
0052 BEGIN:  MVI    B,CTRLZ  ;BLANK SCREEN
0053         CALL   VDM
0054         LXI    H,MSSG2  ;POINT TO MESSAGE
0055         CALL   VDPRT   ;& SIGN ON
0056 INPUT:  CALL   SINP    ;LOOK FOR KEY PRESSED
0057         JZ     INPUT   ;NONE,LOOK AGAIN
0058         CPI    ESC     ;RETURN TO ALS-8?
0059         JZ     RETRN   ;BACK TO SYSTEM
0060         CPI    'T'     ;TEXT?
0061         JZ     PTFOR   ;GO FORMAT PURE TEXT FILE
0062         CPI    'A'     ;ASSEMBLY LANGUAGE FILE?
0063         JZ     ASFOR   ;A DIFFERENT KETTLE OF FISH
0064         JMP    BEGIN   ;ELSE OPERATOR ERROR
0065 *
0066 PTFOR:  LHLD   SOURC  ;HL = SOURCE FILE PTR
0067         XCHG   .       ;DE = SOURCE FILE PTR
0068         LHLD   DEST    ;HL = DEST FILE PTR
0069         LDAX   D       ;GET SOURCE CHAR
0070         CPI    '.'     ;POSSIBLE TEXT "LINE HOLDER"?
0071         JNZ    PTFR1   ;NO,IT'S A NORMAL LINE
0072         INX    D       ;POINT TO NEXT SOURCE CHAR
0073         LDAX   D       ;AND FETCH IT
0074         CPI    '.'     ;PROBABLE ELLIPSIS?
0075         JZ     PTFR0   ;FIRST DOT WAS NOT A LINE HOLDER
0076         MVI    M,' '   ;SUBSTITUTE BLANK FOR DOT
0077         INX    H       ;BUMP DEST PTR
0078         JMP    PTFR1
0079 PTFR0:  DCX    D       ;REWIND TO START OF NORMAL LINE
0080 PTFR1:  CALL   MVRM1   ;MOVE TEXT LINE
0081         CALL   GTNXT   ;CHECK FOR NEXT LINE
0082         JZ     PTDON   ;EOF ENCOUNTERED
0083         CALL   GTSRC   ;UPDATE SOURCE PTR
0084         JMP    PTFOR   ;PROCESS NEXT LINE
0085 PTDON:  MVI    A,'T'   ;"TEXT" FILE TYPE
0086         STA    HTYPE   ;INSERT IN TAPE HEADER
0087         CALL   NDFIL   ;ENDFILE & FILL BLOCK
0088         JMP    FINIS   ;DISPLAY & SAVE
0089 *
0090 ASFOR:  LHLD   BOFP    ;HL = START OF SOURCE FILE
0091 ASFR0:  MOV    A,M     ;GET A CHAR
0092 ASFR1:  CPI    EOF     ;DONE WITH FIRST PASS?
0093         JZ     ASFP2   ;YES, BEGIN FORMATTING
0094         CPI    QUOTE   ;TEST FOR START OF ' ' CASE
0095         JNZ    ASMR1   ;NOT A QUOTED BYTE
0096         INX    H       ;POINT TO QUOTED BYTE
0097         MOV    A,M     ;GET IT
0098         CPI    ' '     ;IS IT A BLANK?
0099         JNZ    ASMOR   ;SOMETHING ELSE, MOVE PAST
0100         INX    H       ;POINT TO POSSIBLE ENDQUOTE
0101         MOV    A,M     ;FETCH IT...
0102         CPI    QUOTE   ;...& TEST
0103         JNZ    ASFR1   ;NOT A VALID ' ' CASE
0104         DCX    H       ;YES, POINT TO QUOTED BLANK
0105         MVI    M,FILL   ;INSTALL TEMPORARY DUMMY
0106 ASMOR:  INX    H       ;POINT TO CLOSING QUOTE
0107 ASMR1:  INX    H       ;POINT TO NEXT CHAR...
0108         JMP    ASFR0   ;...& GO TEST IT
0109 ASFR2:  LHLD   SOURC   ;PT TO SOURCE BLOCK
0110         MOV    A,M     ;GET FIRST SOURCE CHAR
0111         CPI    '*'     ;IS IT ALS-8 REMARK?
0112         JNZ    TSTLB   ;NO,MUST BE A LABEL
0113         CALL   MVREM   ;MOVE THE REMARK INTACT
0114         CALL   GTNXT   ;UPDATE FORMAL SOURCE LINE ADDR
0115         JZ     NOMOR   ;EOF FLAG RETURNED
0116         CALL   GTSRC   ;UPDATE PHYSICAL SOURCE LINE ADDR
0117         JMP    ASFR2   ;LOOK AT NEXT LINE
0118 TSTLB:  LHLD   SOURC   ;GET PTR TO PHYSICAL SOURCE LINE
0119         MOV    A,M     ;GET FIRST CHAR
0120         CPI    ' '     ;ALPHA HERE FOR LABEL
0121         JZ     DOOPC   ;NO LABEL, FIND OPCODE
0122         CALL   MVLAB   ;MOVE THE LABEL
0123 DOOPC:  CALL   MKTAB   ;FORCE CP/M FORMATTER
0124         CALL   MOVOC   ;NOW MOVE OPCODE FIELD
0125         JZ     NDLIN   ;PREMATURE EOL
0126         CALL   MKTAB   ;INSERT TAB AFTER OPCODE FIELD
0127         CALL   MOVOP   ;MOVE OPERAND FIELD
0128         JZ     NDLIN   ;PREMATURE EOL
0129         CALL   MKTAB   ;INSERT TAB AFTER OPERAND FIELD
0130         CALL   FNDCF   ;FIND COMMENT FIELD
0131         JZ     NDLIN   ;COMMENT NOT PRESENT, EOL
0132         CALL   MOVCF   ;MOVE COMMENT FIELD
0133 *
0134 NDLIN:  CALL   GTNXT   ;LINE FINISHED,TEST FOR NEXT
0135         JZ     NOMOR   ;EOF ENCOUNTERED
0136         CALL   GTSRC   ;COMPUTE PHYSICAL SOURCE PTR
0137         JMP    ASFR2   ;PROCESS NEXT LINE
0138 NOMOR:  MVI    A,'A'   ;ASSEMBLY LANGUAGE FILETYPE
0139         STA    HTYPE   ;INSTALL IN SOLOS HEADER
0140         CALL   NDFIL   ;MARK EOF & FILL BLOCK
0141         CALL   SPFIX   ;CONVERT DUMMIES TO BLANKS
0142         CALL   LCASE   ;CONVERT COMMENTS TO LOWERCASE
0143 FINIS:  MVI    B,CTRLZ  ;BLANK SCREEN
0144         CALL   VDM
0145         LXI    H,MSSG3
0146         CALL   VDPRT
0147         CALL   FILE    ;DISPLAY FNAME & NEW PARMS
0148         CALL   TSAVE   ;ONCE!
0149         CALL   TSAVE   ;TWICE!
0150         LXI    H,MSSGA
0151         CALL   VDPRT
0152 YESNO:  CALL   SINP    ;TEST FOR INPUT
0153         JZ     YESNO   ;NONE, LOOK AGAIN
0154         CPI    'Y'     ;SIGNAL TO VERIFY FILE?
0155         JNZ    RETRN   ;FUNCTION NOT DESIRED
0156 CCOPY:  LHLD   BOFP    ;GET PTR TO START OF FILE
0157 CPCOP:  MOV    A,M     ;GET A CHAR
0158         CPI    CTRLZ   ;END OF CP/M FILE
0159         JZ     RETRN   ;YES, RET TO SYSTEM
0160         CPI    TAB     ;ELSE LOOK FOR TAB
0161         JNZ    CPOUT   ;NO, PUBLISH THE CHAR
0162         MVI    B,' '   ;SUBSTITUTE SPACE FOR TAB
```

```
0163          JMP     CPOT1   ;DISPLAY IT
0164 CPOUT:   MOV     B,A     ;SET THE CHAR UP
0165 CPOT1:   CALL    VDM     ;SEND TO CRT
0166          INX     H       ;BUMP FILE PTR
0167          JMP     CPCOP   ;LOOP & DO NEXT CHAR
0168 *
0169 RETRN:   LHLD    SYSSP   ;GET BACK CALLING SP
0170          SPHL    .       ;RESTORE
0171          JMP     EORMS   ;BACK TO CALLER
0172 *
0173 ERROR:   LXI     H,MSSG1
0174          CALL    VDPRT
0175          CALL    FILE    ;SHOW EMPTY FILE PARMS
0176          JMP     RETRN
0177 *
0178 GTSRC:   LHLD    LINAD   ;GET PTR TO FORMAL LINE
0179          LXI     D,6     ;FORCE OFFSET
0180          DAD     D       ;HL = PTR TO PHYSICAL SOURCE LINE
0181          SHLD    SOURC   ;SAVE IT
0182          RET
0183 *
0184 MVREM:   MVI     M,';'   ;CP/M REMARK
0185          XCHG    .       ;DE = PTR TO SOURCE
0186          LHLD    DEST    ;HL = PACKED SOURCE DESTINATION
0187 *
0188 MVRM1:   LDAX    D       ;GET SOURCE CHAR
0189          CPI     CR      ;EOL MARK?
0190          JZ      MRDON   ;REMARK LINE MOVED
0191          MOV     M,A     ;MOVE CHAR TO DEST
0192          INX     D       ;BUMP SOURCE...
0193          INX     H       ;...& DEST PTRS
0194          JMP     MVRM1   ;CONTINUE MOVE
0195 MRDON:   MVI     M,CR    ;CP/M END OF...
0196          INX     H       ;BUMP DEST PTR
0197          MVI     M,LF    ;...LINE MARKERS
0198          INX     H       ;BUMP FOR NEXT AVAIL DEST
0199          SHLD    DEST    ;UPDATE POINTER VARIABLE
0200          RET
0201 *
0202 GTNXT:   LHLD    LINAD   ;GET PTR TO LAST FORMAL LINE
0203          MVI     D,0     ;CLEAR HIGH OFFSET ADDR
0204          LDA     CCBYT   ;GET LOW ORDER OFFSET
0205          MOV     E,A     ;INSTALL PAGE ADDR
0206          DAD     D       ;MAKE NEW FORMAL LINE ADDR
0207          MOV     A,M     ;GET FIRST CHAR OF NEW LINE
0208          CPI     EOF     ;IS IT ALS-8 END OF FILE
0209          RZ      .       ;YES, RET WITH Z FLAG SET
0210          SHLD    LINAD   ;UPDATE FORMAL LINE PTR
0211          STA     CCBYT   ;SAVE CHAR COUNT BYTE
0212          RET
0213 *
0214 MVLAB:   XCHG    .       ;DE = SOURCE PTR
0215          LHLD    DEST    ;HL = NEXT AVAIL DEST PTR
0216 MVLB1:   LDAX    D       ;GET SOURCE CHAR
0217          CPI     ' '     ;ALS-8 END OF LABEL?
0218          JZ      MLDON
0219          CPI     ':'     ;INTEL END OF LABEL?
0220          JZ      MLDON
0221          MOV     M,A     ;MOVE THE CHAR
0222          INX     D       ;BUMP SOURCE...
0223          INX     H       ;...DEST PTRS
0224          JMP     MVLB1   ;CONTINUE MOVE
0225 MLDON:   MVI     M,':'   ;FORCE INTEL DELIMITER
0226          INX     H       ;BUMP DEST PTR
0227          SHLD    DEST    ;UPDATE DESTINATION PTR
0228          INX     D       ;DE = PTR TO SOURCE AFTER LABEL
0229          XCHG    .       ;HL = SOURCE PTR
0230          SHLD    SOURC
0231          RET
0232 *
0233 MKTAB:   LHLD    DEST    ;HL = DEST FILE PTR
0234          MVI     M,TAB   ;FORCE CP/M FORMATTER
0235          INX     H       ;BUMP FOR NEXT DEST PTR
0236          SHLD    DEST    ;UPDATE VARIABLE
0237          RET
0238 *
0239 MOVOC:   LHLD    SOURC   ;HL = DEST FILE PTR
0240          XCHG    .       ;DE = SOURCE PTR
0241          LHLD    DEST    ;HL = DEST PTR
0242 MVOC1:   LDAX    D       ;GET POSS OPCODE CHAR
0243          CPI     ' '     ;SPACE DELIMITER?
0244          JNZ     TSTAS   ;NO, LOOK FOR ASC PSEUDO
0245          INX     D       ;BUMP SOURCE PTR
0246          JMP     MVOC1   ;KEEP LOOKING FOR OPCODE
0247 TSTAS:   CPI     'A'     ;POSSIBLE ASC PSEUDO?
0248          JNZ     MVOCF   ;NO,GO MOVE OPCODE FIELD
0249          INX     D       ;PT TO NEXT CHAR
0250          LDAX    D       ;GET IT
0251          CPI     'S'     ;TEST 2ND OPCODE CHAR
0252          DCX     D       ;BACK UP IN EITHER CASE
0253          JNZ     MVOCF   ;FALSE ALARM
0254          MVI     A,-1    ;MAKE DEFINITE NON-ZERO
0255          STA     ASCSW   ;OPERAND WILL BE ASCII STRING
0256          XCHG    .       ;HL > "A", DE > DEST
0257          MVI     M,'D'   ;WE ARE MAKING...
0258          INX     H
0259          MVI     M,'B'   ;...A PADDED "DB"...
0260          INX     H
0261          MVI     M,' '   ;...PSEUDO-OP
0262          DCX     H       ;POINT TO 'B'
0263          DCX     H       ;POINT TO 'D'
0264          XCHG    .       ;DE = SOURCE PTR, HL = DEST PTR
0265 MVOCF:   LDAX    D       ;GET SOURCE CHAR
0266          CPI     CR      ;END OF OPCODE AND LINE
0267          JNZ     MVCF1   ;NO EOL, LOOK FOR FIELD DELIM
0268          SHLD    DEST    ;UPDATE DEST VARIABLE
0269          JMP     LNDON   ;TERMINATE OPCODE FIELD
0270 MVCF1:   CPI     ' '     ;END OF OPCODE?
0271          JZ      MVDON   ;TERMINATE OPCODE FIELD
0272          MOV     M,A     ;PUT CHAR TO DEST FILE
0273          INX     D       ;BUMP SOURCE...
0274          INX     H       ;...& DEST PTRS
0275          JMP     MVOCF   ;CONTINUE OPCODE FIELD MOVE
0276 *
0277 MVDON:   SHLD    DEST    ;UPDATE DEST VARIABLE
0278 MVDN1:   XCHG    .       ;HL = NEXT SOURCE ADDR-1
0279          SHLD    SOURC   ;SAVE SOURCE PTR
0280 *
0281 OKRET:   MVI     A,-1    ;MAKE DEFINITE NON-ZERO
0282          ORA     A       ;CLEAR ZERO FLAG FOR NO EOL
0283          RET
0284 *
0285 LNDON:   LHLD    DEST    ;GET DEST PTR
0286 LNDN1:   DCX     H       ;POINT TO LAST PACKED CHAR
0287          MOV     A,M     ;GET IT
0288          CPI     TAB     ;CP/M FORMATTER?
0289          JNZ     TRMIN   ;NO,APPEND CRLF & UPDATE
0290          JMP     LNDN1   ;YES,BACK UP
0291 TRMIN:   INX     H       ;POINT TO TAB
0292 TRMN0:   MVI     M,CR    ;APPEND CP/M...
0293          INX     H       ;...LINE...
0294          MVI     M,LF    ;...TERMINATOR
0295          INX     H       ;UPDATE DEST PTR
0296          SHLD    DEST    ;UPDATE DEST VARIABLE
0297          XRA     A       ;SET Z FLAG FOR "NO OPERAND" RET
0298          RET
0299 *
0300 MVOP:    LHLD    SOURC   ;HL = NEXT SOURCE PTR
0301          XCHG    .       ;DE = SOURCE PTR
0302          LHLD    DEST    ;HL = FILE DEST PTR
0303 MVOP1:   LDAX    D       ;GET SOURCE CHAR
0304          CPI     CR      ;PREMATURE EOL?
0305          JZ      LNDON   ;NO OPERAND FIELD FOUND
0306          CPI     ' '     ;FIELD DELIMITER?
0307          JNZ     MVOP2   ;WE HAVE AN OPERAND
0308          INX     D       ;ELSE CONTINUE...
0309          JMP     MVOP1   ;...SEARCH FOR OPERAND
0310 MVOP2:   CPI     '.'     ;ALS-8 DUMMY OPERAND?
0311          JNZ     TRUOP   ;NO, GENUINE OPERAND
0312          INX     D       ;POINT AFTER DUMMY OPERAND
0313          JMP     MVDON   ;UPDATE & RET
0314 TRUOP:   PUSH    PSW     ;SAVE THE CHAR
0315          LDA     ASCSW   ;GET ASCII STRING FLAG
0316          CPI     -1      ;FFH IF ASCII STRING
0317          JZ      ASTMV   ;YES, DELIMIT & MOVE STRING
0318          POP     PSW     ;RETRIEVE THE CHAR
0319 TRUP1:   MOV     M,A     ;ELSE MOVE CHAR TO DEST
0320          INX     D       ;BUMP SOURCE PTR
0321          LDAX    D       ;GET SOURCE CHAR
0322          CPI     CR      ;LINE DELIMITER?
0323          JNZ     TRUP2   ;NO EOL, POSSIBLE FIELD DELIMITER
0324          INX     H       ;BUMP DEST PTR
0325          SHLD    DEST    ;UPDATE DEST VARIABLE
0326          JMP     LNDON
0327 TRUP2:   CPI     ' '     ;FIELD DELIMITER?
0328          JZ      OPDON   ;YES, MOVE IS FINISHED
0329          INX     H       ;NO,BUMP DEST PTR
0330          JMP     TRUP1   ;MOVE NEXT CHAR
0331 ASTMV:   POP     PSW     ;RETRIEVE SAVED CHAR
0332          MOV     B,A     ;SAVE STRING DELIMITER
0333          CALL    CPDEL   ;DELIMITER HOUSEKEEPING
0334 ASMV1:   LDAX    D       ;GET STRING CHAR
0335          CPI     CR      ;ANOMALOUS RIGHT DELIMITER?
0336          JNZ     ASMV2   ;NO, CHECK FOR NORMAL DELIMITER
0337          CALL    CPDEL   ;UPDATE TARGET FILE
0338          XRA     A       ;MAKE A ZERO
0339          STA     ASCSW   ;NO ASSUMPTIONS ABOUT NEXT LINE
0340          JMP     TRMN0   ;PREMATURE EOL = STRING DELIMITER
0341 ASMV2:   CMP     B       ;RIGHT STRING DELIMITER?
0342          JZ      ASDON   ;STRING IS MOVED
0343          MOV     M,A     ;ELSE MOVE CHAR DOWN
0344          INX     D       ;BUMP SOURCE PTR
0345          INX     H       ;BUMP DEST PTR
0346          JMP     ASMV1   ;CONTINUE STRING MOVE
0347 ASDON:   XRA     A       ;MAKE ZERO
0348          STA     ASCSW   ;CLEAR OPERAND SWITCH
0349          CALL    CPDEL   ;DO RIGHT DELIMITER
0350          LDAX    D       ;GET CHAR AFTER OPERAND
0351          CPI     CR      ;PREMATURE EOL?
0352          JNZ     ASDN1   ;NOT YET
0353          SHLD    DEST    ;UPDATE DEST VARIABLE
0354          JMP     LNDON   ;NOTHING AFTER STRING CONSTANT
0355 ASDN1:   SHLD    DEST    ;UPDATE DEST VARIABLE
0356          JMP     MVDN1   ;UPDATE & RET
```

```
0357 OPDON: INX    H       ;BUMP DEST PTR
0358        JMP    MVDON   ;UPDATE & RET
0359 *
0360 CPDEL: MVI    M,QUOTE  ;FORCE CP/M STRING DELIMITER
0361        INX    H       ;BUMP DEST PTR
0362        INX    D       ;BUMP SOURCE PTR
0363        RET
0364 *
0365 FNDCF: LHLD   SOURC  ;GET SOURCE PTR
0366 FDCF1: MOV    A,M     ;GET SOURCE CHAR
0367        CPI    CR      ;END OF LINE AFTER OPERAND?
0368        JZ     LNDON   ;COMMENT FIELD ABSENT
0369        CPI    ' '     ;FIELD DELIMITER?
0370        JNZ    TSTCF   ;COMMENT FIELD FOUND,TEST FOR EMPTY
0371        INX    H       ;BUMP SOURCE PTR
0372        JMP    FDCF1   ;CONTINUE SEARCH FOR COMMENT
0373 TSTCF: CPI    ';'     ;COMMENT DELIMITER?
0374        JZ     TSCF1   ;WE MAY HAVE A COMMENT
0375        CPI    '*'     ;THE OTHER WAY TO DELIMIT A COMMENT?
0376        JNZ    OKCOM   ;WE DEFINITELY HAVE A COMMENT
0377 TSCF1: INX    H       ;POINT TO CHAR AFTER SEMICOLON
0378        MOV    A,M     ;LOAD IT
0379        CPI    CR      ;COMMENT FIELD EMPTY?
0380        JZ     LNDON   ;YES
0381 *
0382 OKCOM: SHLD   SOURC  ;UPDATE SOURCE PTR VARIABLE
0383        LHLD   DEST    ;HL = DEST PTR
0384        MVI    M,';'   ;FORCE CP/M COMMENT DELIMITER
0385        INX    H       ;BUMP DESTINATION PTR
0386        SHLD   DEST    ;UPDATE DEST PTR VARIABLE
0387        JMP    OKRET   ;SET FLAG FOR "COMMENT PRESENT"
0388 *
0389 MOVCF: LHLD   SOURC  ;GET SOURCE FILE PTR
0390        XCHG   .       ;DE = SOURCE FILE PTR
0391        LHLD   DEST    ;HL = DEST FILE PTR
0392        JMP    MVRM1   ;MOVE COMMENT FIELD DOWN
0393 *
0394 NDFIL: LHLD   DEST    ;HL = DEST PTR
0395 NDFL1: MVI    M,CTRLZ  ;FORCE CP/M END OF FILE MARK
0396        MOV    A,L     ;GET LS ADDR BYTE
0397        CPI    0FFH    ;IS BLOCK FULL?
0398        JZ     NDFL2   ;BLOCK FILL COMPLETE
0399        INX    H       ;ELSE BUMP POINTER
0400        JMP    NDFL1   ;LOOP & FILL WITH CP/M ENDFILE MARKS
0401 NDFL2: SHLD   EOFP    ;UPDATE FOR DISPLAY & TAPE SAVE
0402        RET    .
0403 *
0404 SPFIX: LHLD   BOFP    ;HL = START OF TARGET FILE
0405 SPFX0: MOV    A,M     ;FETCH A CHAR
0406        CPI    CTRLZ   ;CP/M ENDFILE ENCOUNTERED?
0407        RZ     .       ;PASS COMPLETE
0408        CPI    FILL    ;IS IT A DUMMY CHAR?
0409        JNZ    SPMOR   ;NO, KEEP MOVING
0410        MVI    M,' '   ;SUBSTITUTE BLANK FOR DUMMY
0411 SPMOR: INX    H       ;BUMP FILE PTR
0412        JMP    SPFX0   ;LOOP & SEARCH
0413 *
0414 LCASE: LHLD   BOFP    ;HL = START OF FILE
0415 LCAS1: MOV    A,M     ;GET A CHAR
0416        CPI    CTRLZ   ;ARE WE DONE?
0417        RZ     .       ;YES
0418        CPI    ';'     ;IS IT A REMARK LINE?
0419        JZ     NEWLN   ;YES, IGNORE & LOOK FOR NEXT
0420        CALL   GETCF   ;ELSE LOOK FOR A COMMENT FIELD
0421        JZ     LCAS1   ;NONE FOUND
0422 LOWER: MOV    A,M     ;GET A COMMENT CHAR
0423        CPI    LF      ;ARE WE DONE YET?
0424        JNZ    LOWR1   ;NO,CONTINUE CONVERSION
0425        INX    H       ;HL = POSS NEW LINE
0426        JMP    LCAS1
0427 LOWR1: CALL   UCTST   ;TEST FOR UPPER CASE
0428        JZ     NXTUC   ;NOT UC,TEST NEXT CHAR
0429        ADI    20H     ;ELSE BIAS UP
0430        MOV    M,A     ;REINSTALL IN FILE
0431 NXTUC: INX    H       ;BUMP FILE PTR
0432        JMP    LOWER   ;LOOP TO NEXT CHAR
0433 *
0434 NEWLN: INX    H       ;PT TO NEXT CHAR
0435        MOV    A,M     ;GET IT
0436        CPI    LF      ;HAVE WE FOUND EOL?
0437        JNZ    NEWLN   ;NO, LOOP & SEARCH
0438        INX    H       ;POINT TO START OF NEW LINE
0439        JMP    LCAS1   ;GO PROCESS IT
0440 *
0441 GETCF: INX    H       ;POINT TO NEXT CHAR
0442        MOV    A,M     ;GET IT
0443        CPI    LF      ;EOL REACHED?
0444        JZ     CFABS   ;COMMENT FIELD ABSENT, SET FLAG
0445        CPI    ';'     ;START OF COMMENT FIELD
0446        JNZ    GETCF   ;NOT YET, KEEP LOOKING
0447        INX    H       ;POINT TO FIRST CHAR IN COMMENT FIELD
0448        ORA    A       ;CLEAR ZERO FLAG FOR COMMENT PRESENT
0449        RET    .
0450 CFABS: INX    H       ;POINT TO POSSIBLE NEW LINE
0451        XRA    A       ;SET ZERO FLAG FOR FIELD NOT FOUND
0452        RET    .
0453 *
0454 UCTST: CPI    'A'     ;LOWER LIMIT
0455        JC     NOTUC   ;OUT OF BOUNDS
0456        CPI    'Z'+1   ;UPPER LIMIT
0457        JC     YESUC   ;WITHIN RANGE
0458 NOTUC: XRA    A       ;SET Z FOR INVALID RET
0459 YESUC: ORA    A       ;Z RESET ON VALID RET
0460        RET    .
0461 *
0462 TSAVE: MVI    A,UNIT1  ;LOAD UNIT/SPEED BYTE
0463        STA    FNUMF   ;UPDATE SOLOS VARIABLE
0464        LXI    H,FNAME  ;POINT TO CURRENT FILE NAME
0465        LXI    D,THEAD  ;& SOLOS HEADR BYTE
0466        MVI    B,5     ;ONLY 5 ASCII CHARS ALLOWED
0467 MORE:  MOV    A,M     ;SHOVEL THEM INTO SOLOS HEADR
0468        STAX   D
0469        DCR    B
0470        JZ     SAVFL   ;EXECUTE THE SAVE
0471        INX    H       ;ELSE HACK AWAY
0472        INX    D
0473        JMP    MORE
0474 *
0475 SAVFL: CALL   SPRAM   ;DO SOME HOUSEKEEPING
0476        LDA    FNUMF   ;GET UNIT & SPEED BYTE
0477        LXI    H,THEAD  ;POINT TO SOLOS HEADR
0478        CALL   WTAPE   ;DUMP 256 BYTE BLOCK(S) TO TAPE
0479        RET    .       ;END OF TRANSMISSION
0480 *
0481 SPRAM: LHLD   BOFP    ;ALS-8 > SOLOS HEADER
0482        SHLD   LOADR
0483        PUSH   H       ;SAVE FOR THE MOMENT
0484        LHLD   EOFP    ;END > HL
0485        XTHL   .       ;SWAP TOP OF STACK
0486        MOV    A,L     ;COMPUTE BLOCK SIZE
0487        CMA
0488        MOV    L,A
0489        MOV    A,H
0490        CMA
0491        MOV    H,A
0492        INX    H
0493        POP    D       ;GET START
0494        DAD    D       ;ADD OFFSET
0495        INX    H       ;INCLUDE EOF
0496        SHLD   BLOCK   ;SHOVEL IT INTO HEADER
0497        RET
0498 *
0499 * CRT MESSAGE HANDLER
0500 *
0501 VDPRT: MOV    A,M     ;GET CHAR FROM BUFFER
0502        CPI    ETX     ;IZZIT THE END?
0503        RZ     .       ;SURE ENOUGH
0504        MOV    B,A     ;MAIS NON,SET IT UP!
0505        CALL   VDM     ;& VIDEO IT
0506        INX    H       ;& ADJUDICATE THE INDEX
0507        JMP    VDPRT   ;& OBTAIN ANOTHER
0508 *
0509 * CONSOLE MESSAGE AREA
0510 *
0511 MSSG1: DB     CR
0512        ASC    #FILE EMPTY#
0513        DB     CR
0514        DB     ETX
0515 *
0516 MSSG2: DB     CR
0517        ASC    #TYPE: "A" FOR ASSEMBLY LANGUAGE FILE#
0518        DB     CR
0519        ASC    #       "T" FOR TEXT FILE#
0520        DB     CR
0521        ASC    #       ESC TO RETURN TO SYSTEM#
0522        DB     CR
0523        DB     ETX
0524 *
0525 MSSG3: ASC    #SAVING CP/M FILE:#
0526        DB     ETX
0527 *
0528 MSSG4: DB     CR
0529        DB     CR
0530        ASC    #VERIFY? (Y/N)#
0531        DB     CR
0532        DB     CR
0533        DB     ETX
0534 *
0535 * PROGRAM GLOBAL AREA
0536 *
0537 ASCSW: DB     0       ;ASCII STRING SWITCH
0538 CCBYT: DS     1       ;CHARACTER COUNT BYTE
0539 DEST:  DS     2       ;DEST FILE PTR VARIABLE
0540 LINAD: DS     2       ;CURRENT FORMAL LINE PTR
0541 SOURC: DS     2       ;SOURCE FILE PTR VARIABLE
0542 SYSSP: DS     2       ;STORE CALLING SP HERE
0543        DS     24      ;THIS IS THE STACK
0544 STACK: EQU    $
0545 *
0546        END
0547 *
```

CP/M and CDOS Program Compatibility

BY JIM WARNER
Chemistry Board of Studies
Natural Sciences II
University of California
Santa Cruz, CA 95064

Disk operating systems—CP/M by Digital Research (D.R.) for 8080's and CDOS by Cromemco for Z80's—have occasionally been called "virtually identical" (*DDJ* # 35). Cromemco's Users Bulletin (Dec. 1978) makes more specific claims:

> "... CDOS was designed to be upwards CP/M-compatible... This means that most programs written for CP/M (versions upto [sic] and including 1.33) will run without modification under CDOS. This also means that programs written for CDOS will *not* generally run under CP/M."

Cromemco's CDOS and CP/M share identical disk directory structures. In both operating systems, machine language programs are loaded and commence execution at 100H. The programs, termed 'transients,' make calls upon the operating system for I/O operations through a single entry point (05H) which is independent of the amount of memory in the host computer. A JMP at the entry address passes control to the operating system, resident in high memory.

Both CP/M and CDOS decode I/O requests from transient programs according to a function number passed in the C-register. CP/M provides codes 1 to 27 for access to disk files, the console, printer and other I/O devices. CDOS, patterned after CP/M, has adopted all of CP/M's function codes. The principal difference between the operating systems is that Cromemco has included 14 additional codes not possessed by CP/M. CP/M is not protected against out of range function calls and the certain result of an illegal call is a crash.

Another difference is that function codes which open (15) and close (16) disk files and search the disk directory (17, 18) return with registers pointing to the file name differently arranged if the operation was successful. More important, however, is that the returned value for an unsuccessful operation is the same for CDOS and CP/M. The difference is important only for programs which read the directory as data (e.g., to prepare an alphabetized directory listing).

CP/M 1.4 contains three undocumented function codes included in neither CDOS nor earlier CP/M versions. Two of these are used to set (28) and monitor (29) read-only status of disk drives. The nature of the last (30) is unknown to me. The codes are used by some of the utility programs distributed by D.R. as part of the CP/M package.

Finally, CP/M's "jump vector" table can be used in programs that need direct access to I/O drivers. Such programs include track-at-a-time disk copiers, direct disk editors, and utilities such as PASTOCPM (*DDJ*, page 12, No. 37) which read files from non-CP/M structured diskettes. This table is the interface between CP/M's system independent portion (BDOS) supplied by D.R. and the I/O routines (BIOS) supplied by the system configurer. The description of the table and the responsibilities of the I/O routines can be found in D.R.'s *CP/M System Alteration Guide*.

The address of the jump vector table depends on the amount of memory allocated to CP/M and unfortunately D.R. has not specified how transient programs should locate it. As a result many programs from the CP/M Users Group which worked under CP/M 1.3 require modification to function with CP/M 1.4. It is a safe bet that these programs are not CDOS compatible.

From the CP/M perspective it seems an unmanageable task to sort through 14 function numbers and treat them all as exceptions to the normal operating system. Some of these codes provide intricate (and dangerous) direct disk access. Others provide features unrelated to I/O such as integer multiply and divide routines. Happily, many excellent programs for Z80's distributed by Cromemco are made serviceable with CP/M on treating only 2 of the 14 additional CDOS function codes.

The first is READ CONSOLE WITHOUT ECHO (128). The result, a single character with the parity bit reset, is returned in the accumulator. A complicating feature of CP/M makes it necessary to bypass CP/M's INTEROGATE CONSOLE STATUS (11) function. This is required because this function actually reads a character if one is available and saves it where READ CONSOLE WITHOUT ECHO would be unable to find it.

The second important function code is FORMAT STRING TO FILE CONTROL BLOCK (134). HL points to the source string, while DE indicates where the file control block is to be set up. This procedure must pad the name and extension in the file name with the proper number of spaces while excising the period between them. The optional drive designator at the head of the string must be decoded and lower case letters must be converted to upper case.

One of the CP/M function codes must be given special treatment. It is necessary to restore the accumulator after a WRITE CONSOLE (2) function call. If this is not done EDIT will not print control characters.

An implementation of these modifications to the CP/M run time environment is presented in FILTER. Programs we have successfully run under our filtered CP/M include Cromemco's relocating macro assembler (ASMB 02.15), linker (LINK), editor (EDIT 00.09), 8080 to Z-80 assembly language mnemonic converter (TRANSLAT V.2.1), machine language debugger (DEBUG 00.08) and 16K BASIC 5.0 (in 18.5K!).

The location and installation of the filter are the final concern. The jump instruction at 05H will have to be changed to route system function calls through the filter. The address field of this jump instruction, however, serves a secondary purpose. It is read by many transient programs to find the highest memory location that they can use without running into the operating system. This feature must be preserved.

Our solution is to put the filter in the memory page immediately below CP/M's BDOS with the entry jump instruction suitably modified. Its installation is similar to CP/M's DDT in that FILTER, loaded initially in low memory, boosts itself into operating position. FILTER is not a permanent modification to CP/M. Rather, as explained in the listing, it is made a part of the memory image of the Cromemco programs which require it. The location of the installed filter will depend on the amount of memory available to CP/M. To accommodate variable memory sizes, FILTER contains an installation section which configures the actual filter to run at the top of the transient program area.

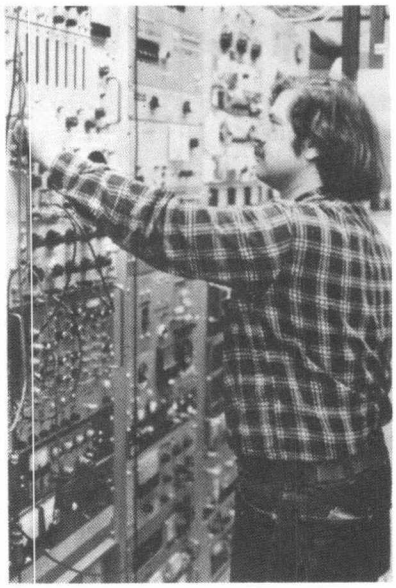
To implement FILTER you will need to provide a system dependent value for the equate IGOR. This determines where FILTER will jump if it encounters an untreated CDOS function code. If you have a ROM monitor program that preserves the registers, put its entry address here. Lacking same, setting IGOR to zero will force CP/M to reboot. This will destroy the evidence (C-register) of the offending function code, however.

If your standard console input routine includes code which backs up the cursor and erases characters when you type rubout you will probably experience strange behavior in BASIC and EDIT. The easiest cure is to include a simple keyboard input routine within FILTER in place of 'GBYTE: JP Z,DUMMY'. If you make this modification be sure to remove the four instructions in part I which modify the address field of the JP instruction. These instructions are marked with *'s in the comment field of the program listing. If you make changes that lengthen part III by more than about 18 bytes you will also need to change the instructions in part II that set the stack pointer below BDOS. Finally, if your basic input output system (BIOS) changes the Z-80 index registers, you will run into difficulties that we have not fully documented.

Filter does not resolve all differences between CDOS and CP/M. One deficiency is that the DIR command in BASIC will give meaningless (but harmless) results. The ability of BASIC to work with disk files is not, however, in any way impaired. It will usually be necessary to hit the 'esc' key twice instead of once to stop a RUNning BASIC program. BASIC and DEBUG contain commands to utilize the programmable timers and BAUD rate selection hardware found on Cromemco I/O cards. These commands will not work without Cromemco hardware.

The present Cromemco software license agreement restricts the use of their software to a single computer of their manufacture. FILTER therefore will be of use primarily to owners of Cromemco computers who cannot run CDOS because their disk controller was manufactured by another company.

The author is a physical chemist at the University of California at Santa Cruz. He is shepherd to a small flock of microcomputers used in several laboratories for data acquisition and apparatus control in transient spectroscopy, molecular dynamics, chemical kinetics and nonlinear dynamics.



```
; VERSION OF MAY 15, 1979   ***   BY J. WARNER
;;;
```

```
Ø1ØØ                    ORG     1ØØH
Ø1ØØ 2AØ6ØØ             LD      HL,(6)  ; GET BDOS ENTRY ADDRESS
Ø1Ø3 2261Ø1             LD      (CPM1+1),HL     ; INSTALL
Ø1Ø6 226EØ1             LD      (CPM2+1),HL     ;    (2 PLACES)
Ø1Ø9 25                 DEC     H               ; MAKE ROOM FOR FILTER
Ø1ØA 2EØØ               LD      L,Ø
Ø1ØC 22Ø6ØØ             LD      (6),HL          ; INSTALL FILTER ADDR
Ø1ØF EB                 EX      DE,HL
Ø11Ø 2AØ1ØØ             LD      HL,(1)          ; GET WARMSTRT ADDR
Ø113 23                 INC     HL              ;   IN BIOS
Ø114 23                 INC     HL              ;     JMP TABLE
Ø115 23                 INC     HL              ; CSTAT ADDRESS
Ø116 226AØ1             LD      (CSTAT+1),HL    ; INSTALL.
Ø119 23                 INC     HL              ; INCREMENT 3X      *
Ø11A 23                 INC     HL              ;   TO FIND         *
Ø11B 23                 INC     HL              ;     CONIN.        *
Ø11C 2271Ø1             LD      (GBYTE+1),HL    ; INSTALL.          *
Ø11F 2145ØØ             LD      HL,NAME-FILTER  ; CALCULATE RELATIVE
Ø122 19                 ADD     HL,DE           ;   SUBROUTINE ADDR.
Ø123 228EØ1             LD      (NAM1+1),HL     ;    AND INSTALL
Ø126 2299Ø1             LD      (NAM2+1),HL     ;       2 PLACES.
Ø129 215AØ1             LD      HL,FILTER
Ø12C Ø171ØØ             LD      BC,FILTER$END-FILTER
Ø12F EDBØ               LDIR                    ; INSTALL FILTER
Ø131            ;
Ø131            ;
Ø131            ; PART II - THE FILTER, JUST INSTALLED, CONTAINS A
Ø131            ;       ONE USE ROUTINE TO MOVE THE *.COM FILE  ON
Ø131            ;       TOP OF THIS STUFF AND INTO POSITION. FIRST
Ø131            ;       PUT THE STACK BETWEEN CP/M AND THE FILTER.
Ø131            ;
Ø131 ED5BØ6ØØ           LD      DE,(6)          ; GET FILTER ADDR BACK
Ø135 DD2161ØØ           LD      IX,MOVE$DOWN-FILTER
Ø139 DD19               ADD     IX,DE   ; MOVE$DOWN RELOCATED ADDR
Ø13B 21E1ØØ             LD      HL,MOVE$DOWN-FILTER+8ØH
Ø13E 19                 ADD     HL,DE           ; FOR THE
Ø13F F9                 LD      SP,HL           ;   STACK
Ø14Ø 114AØ1             LD      DE,MESG
Ø143 ØEØ9               LD      C,9     ; SEND
Ø145 CDØ5ØØ             CALL    5       ; FILTER LOGO
Ø148 DDE9               JP      (IX)    ; JMP TO RELOCATED MOVE$DOWN
Ø14A ØDØA4344   MESG:   DEFB     13,1Ø,'CDOS FILTER',13,1Ø,'$'
Ø15A            ;
Ø15A            ; PART III - ****  THE FUNCTION CODE FILTER  ****
Ø15A            ;       THE FOLLOWING CPM/CDOS FUNCTION CODES ARE
Ø15A            ;       AFFECTED:
Ø15A            ;        2  WRITE CHARACTER TO CONSOLE: THE A-REG
Ø15A            ;           MUST BE SAVED AND RESTORED.
Ø15A            ;       11  CHECK CONSOLE STATUS: THIS REQUEST IS
Ø15A            ;           PASSED DIRECTLY TO BIOS (I.E.  ROUTED
Ø15A            ;           AROUND CP/M).
Ø15A            ;      128  READ CONSOLE (NO ECHO): THIS REQUEST
Ø15A            ;           PASSED DIRECTLY TO BIOS (I.E. ROUTED
Ø15A            ;           AROUND CP/M).
Ø15A            ;      134  FORMAT STRING TO FILE CONTROL BLOCK.
Ø15A            ;           DONE HERE, IT DOES NOT ACCEPT '*' WILD
Ø15A            ;           CARDS.
Ø15A            ;
Ø15A F5         FILTER: PUSH    AF
Ø15B 79                 LD      A,C
Ø15C FEØ2               CP      A,Ø2            ;WRITE FCN CODE ??
Ø15E 2ØØ5               JR      NZ,NOT$WRT-$
Ø16Ø CDØØØØ     CPM1:   CALL    DUMMY   ; ADDR. SUPPLIED BY PART I
Ø163 F1                 POP     AF
Ø164 C9                 RET
Ø165 F1         NOT$WRT POP     AF
Ø166 79                 LD      A,C
Ø167 FEØB               CP      ØBH     ; CONSTAT FCN CODE
Ø169 CAØØØØ     CSTAT:  JP      Z,DUMMY
Ø16C 87                 ADD     A
Ø16D D2ØØØØ     CPM2:   JP      NC,DUMMY ; NOT A SPECIAL FCN CODE
Ø17Ø CAØØØØ     GBYTE:  JP      Z,DUMMY  ; CY=1 & Z=1 --> FCN.CODE=3ØH
Ø173 3E86               LD      A,86H    ; FORMAT STRING TO FCB FCN CODE
Ø175 B9                 CP      C
Ø176 C2ØØFØ             JP      NZ,IGOR  ; NOT DECODED, **ERROR**
Ø179            ;
Ø179            ; -- FORMAT STRING TO FILE CNTRL BLOCK - 86H
Ø179            ;
Ø179 D5                 PUSH    DE
Ø17A 46                 LD      B,(HL)   ; MIGHT BE DISK DESIGNATOR
Ø17B 23                 INC     HL
Ø17C 7E                 LD      A,(HL)   ; IF THIS IS ":" THEN B-REG
Ø17D FE3A               CP      ':'      ;    CONTAINS DRIVE #
Ø17F 2ØØ6               JR      NZ,NOCOLON-$
Ø181 23                 INC     HL       ; POINTER PAST COLON
Ø182 3EØ7               LD      A,7      ; MASK
Ø184 AØ                 AND     A,B
Ø185 18Ø2               JR      FIRSTBYTE-$
Ø187 2B         NOCOLON DEC     HL
Ø188 AF                 XOR     A        ; SET FOR DEFAULT DISK DRIVE
Ø189            FIRSTBYTE:
Ø189 12                 LD      (DE),A   ; FIRST BYTE OF
Ø18A 13                 INC     DE       ;   FILE CONTROL BLOCK
Ø18B Ø6Ø8               LD      B,8      ; MAX # CHAR'S IN A NAME
Ø18D CDØØØØ     NAM1:   CALL    DUMMY
Ø19Ø 3E2E               LD      A,'.'
Ø192 BE                 CP      A,(HL)
Ø193 2ØØ1               JR      NZ,EXT-$
Ø195 23                 INC     HL
Ø196 Ø6Ø3       EXT:    LD      B,3      ;MAX # CHAR'S IN AN EXTENSION
Ø198 CDØØØØ     NAM2:   CALL    DUMMY
Ø19B AF                 XOR     A        ; ZERO
Ø19C 12                 LD      (DE),A   ; FILE EXTENT
Ø19D D1                 POP     DE
Ø19E C9                 RET
Ø19F            ;
Ø19F 7E         NAME:   LD      A,(HL)   ; GET NEXT CHAR FROM STRING
ØlAØ FE21               CP      21H      ; TERMINATION CHARACTER ??
Ø1A2 381Ø               JR      C,FILL-$
Ø1A4 23                 INC     HL
Ø1A5 FE2E               CP      '.'      ; NONE OF THESE IN FCB'S
Ø1A7 28ØB               JR      Z,FILL-$
Ø1A9 FE6Ø               CP      6ØH      ; LOWER CASE 'A'-1
Ø1AB 38Ø2               JR      C,NAME1-$; JMP IF NOT LOWER CASE
Ø1AD D62Ø               SUB     2ØH      ; CONVERT LOWER TO UPPER CASE
Ø1AF 12         NAME1:  LD      (DE),A   ; PUT CHAR IN FCB
Ø1BØ 13                 INC     DE
Ø1B1 1ØEC               DJNZ    NAME-$
Ø1B3 C9                 RET
Ø1B4            ;
Ø1B4 3E2Ø       FILL:   LD      A,2ØH    ;ASCII SPACE
Ø1B6 12                 LD      (DE),A
Ø1B7 13                 INC     DE
Ø1B8 1ØFA               DJNZ    FILL-$
Ø1BA C9                 RET
Ø1BB            MOVE$DOWN
Ø1BB 21ØØØ2             LD      HL,2ØØH      ; BOTTOM OF PROGRAM
Ø1BE ED4BØ6ØØ           LD      BC,(6)       ; TOP OF T.P.A.
Ø1C2 Ø5                 DEC     B            ; AMOUNT TO MOVE DOWN
Ø1C3 11ØØØ1             LD      DE,1ØØH      ; ORG'N FOR *.COM FILES
Ø1C6 EDBØ               LDIR
Ø1C8 C3ØØØ1             JP      1ØØH
Ø1CB            FILTER$END      EQU     $
Ø1CB                    END     $
```

Changing SPI Formatted Programs to CP/M Compatible Text

BY MARK M. ZEIGER
198-01B 67th Ave.
Flushing, N.Y. 11365

Many computer hobbyists who started a few years ago with relatively simple equipment probably used assemblers such as Processor Tech's SP1 or ALS-8. Now they have 'graduated' to more sophisticated operating systems, CP/M being the most prevalent. Think of all those great source programs still on your cassettes that you would love to rewrite to use with CP/M. However, who has the time to do all that retyping? Computer to the rescue! The following program will change source programs written by the SP1 executive (as well as programs written by ALS-8, XEK, and the PDS assemblers to name a few) to text acceptable to CP/M assemblers and editors.

The program is quite simple to use. First the SP1 file must be saved on the CP/M disk. Load the source into RAM at 100H. Then boot in CP/M and type

SAVE xx NAME.EXT.

Next run the change program by typing the command line

CHANGE NAME1. EXT NAME2. EXT

where NAME1 is the name of the SP1 source file and NAME2 will be the name of the CP/M compatible file. Each file may be prefixed by a drive designation. For example, the command

A >CHANGE B:LIST.SP1 LIST.ASM

will change the file on drive B called LIST.SP1 to a CP/M compatible file called LIST.ASM and will write the file on the logged-in drive (A). Note that both the source and destination file must be specified when running the program or a run-time error will occur.

Much of the program is self-documented. Studying the CP/M interfacing guide and the program will help in understanding most of the program's logic. The major subroutine is MOVEREC. This routine reads in one record (128 bytes), takes out the line numbers and character count found in SP1 text, and substitutes a carriage return-line feed for the single carriage return in the SP1 source. It will also format the text so that labels, code, and comments are neatly placed in columns. The end of the SP1 file is denoted by an 01 which is searched for by MOVEREC. The subroutine RESTORENEG is called only if a line number is located in two different records. Any run-time error will abort the program with an appropriate message. The CP/M file will not be written if there is an error.

Listing

```
; SOURCE FOR CHANGING XEK FILES TO CP/M FILES.

        ORG 100H

BDOS    EQU 0005
SOURCE  EQU 5CH

PRTBUF  EQU 9
OPEN    EQU 15
CLOSE   EQU 16
MAKE    EQU 22
READFIL EQU 20
WRTFILE EQU 21
SETDMA  EQU 26
DELETE  EQU 19
RESET   EQU 13
BL      EQU ' '

0100 210000     LXI H,0
0103 39         DAD SP
0104 220402     SHLD STKSAVE
0107 310302     LXI SP,NEWSTK

010A CDB302     CALL GETNAMES
010D CDE102     CALL OPENFILE
0110 CD0C03     CALL READREC
0113 CD2F03     CALL MOVEREC
0116 CD0C03     CALL WRITEREC
0119 CD1704     CALL CLOSEFILE

011C 2A0402     LHLD STKSAVE
011F F9         SPHL                    ; RETURN TO CP/M
0120 C9         RET
```

```
0121 05          TAB     DB 5
0122 00          EOLCNT  DB 0
0123 00202020    DEST    DB 0,'           ',0,0  ;ELEVEN BLANKS
0131 00000000            DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0080 =           INBUF   EQU 80H
0144             OUTBUF  DS 128
01C4                     DS 40H
0203 =           NEWSTK  EQU $-1
0204             STKSAVE DS 2

0206 536F7572    MSSG1   DB 'Source file not on disk',0DH,0AH,'$'
0220 52656164    MSSG2   DB 'Read past end of file',0DH,0AH,'$'
0238 4E6F206D    MSSG3   DB 'No more directory space',0DH,0AH,'$'
0252 4469736B    MSSG4   DB 'Disk full',0DH,0AH,'$'
025E 4572726F    MSSG5   DB 'Error in extending file',0DH,0AH,'$'
0278 4E6F2073    MSSG6   DB 'No source file indicated',0DH,0AH,'$'
0293 4E6F2064    MSSG7   DB 'No destination file indicated',0DH,0AH,'$'
                 ;
                 GETNAMES
02B3 3A5D00              LDA SOURCE+1
02B6 FE20                CPI BL
02B8 CA0B04              JZ ERROR6
02BB 3A6D00              LDA SOURCE+17
02BE FE20                CPI BL
02C0 CA1104              JZ ERROR7
02C3 216C00              LXI H,SOURCE+16
02C6 112301              LXI D,DEST
02C9 060C                MVI B,12
02CB 7E          MOVNAME MOV A,M
02CC 12                  STAX D
02CD 23                  INX H
02CE 13                  INX D
02CF 05                  DCR B
02D0 C2CB02              JNZ MOVNAME
02D3 216800              LXI H,SOURCE+12
02D6 3E00                MVI A,0
02D8 0618                MVI B,24
02DA 77          ZEROS   MOV M,A
02DB 23                  INX H
02DC 05                  DCR B
02DD C2DA02              JNZ ZEROS
02E0 C9                  RET

                 OPENFILE
02E1 0E0F                MVI C,OPEN
02E3 115C00              LXI D,SOURCE
02E6 CD0500              CALL BDOS
02E9 FEFF                CPI 0FFH
02EB CAED03              JZ ERROR1
02EE 0E13                MVI C,DELETE
02F0 112301              LXI D,DEST
02F3 CD0500              CALL BDOS
02F6 0E16                MVI C,MAKE
02F8 112301              LXI D,DEST
02FB CD0500              CALL BDOS
02FE FEFF                CPI 0FFH
0300 CAF903              JZ ERROR3
0303 0E0F                MVI C,OPEN
0305 112301              LXI D,DEST
0308 CD0500              CALL BDOS
030B C9                  RET

                 READREC
030C F5C5D5E5            PUSH PSW ! PUSH B ! PUSH D ! PUSH H
0310 0E1A                MVI C,SETDMA
0312 118000              LXI D,INBUF
0315 CD0500              CALL BDOS
0318 0E14                MVI C,READFIL
031A 115C00              LXI D,SOURCE
031D CD0500              CALL BDOS
0320 FE01                CPI 1
0322 CAF303              JZ ERROR2
0325 E1D1C1F1            POP H ! POP D ! POP B ! POP PSW
0329 218000              LXI H,INBUF
032C 0680                MVI B,128
032E C9                  RET

                 MOVEREC
032F 0680                MVI B,128
0331 0E80                MVI C,128
0333 218000              LXI H,INBUF
0336 114401              LXI D,OUTBUF
0339 7E          NEWLINE MOV A,M
033A FE01                CPI 1
033C CAA603              JZ EOFMARK
033F 3E00                MVI A,0
0341 322101              STA TAB
0344 7E                  MOV A,M
0345 D606                SUI 6
0347 322201              STA EOLCNT
034A 78                  MOV A,B
034B D606                SUI 6
034D 47                  MOV B,A
034E 232323              INX H ! INX H ! INX H
0351 232323              INX H ! INX H ! INX H
0354 FC0C03              CM READREC
0357 CC0C03              CZ READREC
035A FCB203              CM RESTORENEG
035D 3A2101      OLDLINE LDA TAB
0360 FE00                CPI 0
0362 7E                  MOV A,M
0363 C28603              JNZ NOTAB
0366 FE3B                CPI 3BH
0368 C27103              JNZ ASTK
036B 322101              STA TAB
036E C38603              JMP NOTAB
0371 FE2A        ASTK    CPI '*'
0373 C27C03              JNZ BLANK
0376 322101              STA TAB
0379 C38603              JMP NOTAB
037C FE20        BLANK   CPI ' '
037E C28603              JNZ NOTAB
0381 3E09                MVI A,9
0383 322101              STA TAB
0386 12          NOTAB   STAX D
0387 23                  INX H
0388 13                  INX D
0389 05                  DCR B
038A CC0C03              CZ READREC
038D 0D                  DCR C
038E CCC003              CZ WRITEREC
0391 3A2201              LDA EOLCNT
0394 3D                  DCR A
0395 322201              STA EOLCNT
0398 C25D03              JNZ OLDLINE
039B 3E0A                MVI A,0AH
039D 12                  STAX D
039E 13                  INX D
039F 0D                  DCR C
03A0 CCC003              CZ WRITEREC
03A3 C33903              JMP NEWLINE
03A6 3E0A        EOFMARK MVI A,0AH
03A8 12                  STAX D
03A9 13                  INX D
03AA 0D                  DCR C
03AB CCC003              CZ WRITEREC
03AE 3E1A                MVI A,1AH
03B0 12                  STAX D
03B1 C9                  RET

                 RESTORENEG
03B2 2F                  CMA
03B3 C601                ADI 1
03B5 F5                  PUSH PSW
03B6 D5                  PUSH D
03B7 57                  MOV D,A
03B8 78                  MOV A,B
03B9 92                  SUB D
```

Continued on pg. 41

CPM-Pen / Pen-CPM

BY ROD HALLEN
Road Runner Ranch
P.O. Box 73
Tombstone, AZ 85638

The following letter written by Rod Hallen to Jim Gagne is a fitting introduction to the listings which follow.

Dear Dr. Gagne,

In March I wrote to thank you for your Electric Pencil to CPM routine (*DDJ* #33). I have found this to be one of my most used programs. Thanks Again.

Recently I acquired Digital Research's "MAC" macro assembler and was looking for a project to use it on. That's how I learn the ins and outs of any software that I add to my library. I decided to see if I could simplify and speed up your program. The enclosed PRN listings are the result.

The limited length of PEN-CPM and CPM-PEN are due to the macro routines which handle all disk I/O. Have you tried "MAC"? If not, it is well worth the price. Both of these programs were written in less than 3 hours time; an easy afternoon's work.

I wrote these as two separate programs since 95 per cent of my conversions are from Pencil to CPM. I write my assembly listings on the Pencil, convert to CPM, and test. If something isn't right (most of the time!), I reload the PCL file into Pencil and try again. I have also eliminated tab handling in PEN-CPM and line length considerations in both. I only use these programs for assembly language listings and never use a line longer than 72 characters.

You mentioned in your *DDJ* article that you were attempting to change the disk read and write to 512 byte blocks. This is extremeley simple with "MAC." Note the 1024 at the end of the INFILE and OUTFILE lines in the programs. This is the number of bytes read or written at one time. The only limit on this is the amount of memory that you have available for buffers. Almost everything that your program acccomplishes is handled automatically by the routines in the SEQIO (SEQuential I/O) libarary which comes with "MAC."

I've added one routine to SEQIO. LIB. That is MESS which is my message printer. It is described in PEN-CPM.DOC.

These two programs have accomplished many things for me. They have cut down on disk activity, reduced the conversion time, and eliminated a bug I couldn't get out of PENCPM. They have also made me aware of the capabilities of macro assembling.

The bug usually left garbage at the end of an assembly language listing that had been converted from Pencil to CPM. This garbage consisted of the last few lines of the listing repeated one or more times. This didn't, however, affect assembly because my assemblers stop at the first END OP code.

As proof of the increase in conversion speed, PEN-CPM takes exactly half as much time as PENCPM. The same is true in the opposite direction. I have one question about PENCPM. You save the old stack pointer and set up a new SP, which I think is a good idea for any program that will use the stack extensively, and then restore the old stack at the end. However, then you call for a CP/M warm start which restores the old stack pointer again. Why not a simple RET at the end? This returns to CP/M without disk activity and is quicker.

I am sending a copy of this letter and the enclosed programs to *DDJ* for possible publication. I am also sending a disk copy of both source and command files to the CP/M Users' Group for inclusion in one of their library disks. If you do not have "MAC" and you would like a copy of the command files, please send me a soft-sectored 8" diskette and $5 and I will record them for you. I find it necesssary to charge for this service since programming and writing are my only income.

Sincerely,

Rod Hallen

Listings

```
                ;
                ;
                ;* - - - - - - - - - - - - - - - - - - - - - - *
                ;*                                             *
                ;*                  - CPM-PEN -                *
                ;*                                             *
                ;*     ROUTINE TO TRANSLATE  CP/M FILES TO     *
                ;*          TO ELECTRIC PENCIL FORMAT          *
                ;*              (C) COPYRIGHT 1979             *
                ;* ROD HALLEN  BOX 73  TOMBSTONE, AZ  85638    *
                ;*                 26 JULY 1979                *
                ;*     THIS PROGRAM MAY BE USED FREELY BY      *
                ;*    COMPUTER HOBBYISTS FOR NON-COMMERCIAL    *
                ;*    APPLICATIONS.  IT MAY NOT BE USED FOR    *
                ;*     COMMERCIAL PURPOSES OR SOLD WITHOUT     *
                ;*      WRITTEN PERMISSION OF  THE AUTHOR.     *
                ;*                                             *
                ;*     ASSEMBLE WITH DIGITAL RESEARCH  "MAC"   *
                ;*      ASSEMBLER AND SEQIO LIBRARY.  INPUT    *
                ;*      FORMAT IS  "CPM-PEN X.Y" AND OUTPUT    *
                ;*          FILE WILL ALWAYS BE "X.PCL".       *
                ;*                                             *
                ;* - - - - - - - - - - - - - - - - - - - - - - *
                ;
                ;
                ;
0100                    ORG     100H
                ;
                        MACLIB  SEQIO   ;SEQUENTIAL I/O LIBRARY
                ;
0100 210000             LXI     H,0     ;SAVE OLD SP
0103 39                 DAD     SP      ;AND SET UP NEW SP
0104 228303             SHLD    OLDSP
0107 31C503             LXI     SP,NEWSP
                ;
010A            FILE    INFILE,INF,,1,,1024     ;SETUP PENCIL INPUT FILE
                ;
01F0            FILE    OUTFILE,OUTF,,1,PCL,1024    ;SETUP CPM OUTPUT FILE
                ;
02EE                    MESS    MES1    ;PRINT ACTIVITY MESSAGE
                ;
02F6 AF         CPMPN   XRA     A       ;CLEAR ACCUMULATOR
02F7 32C503             STA     CPOS    ;CLEAR CHAR POS COUNTER
02FA            CPMP1   GET     INF     ;GET CHAR FROM INPUT FILE
02FD FE1A               CPI     1AH     ;CPM END OF FILE ?
02FF CA1903             JZ      DONE    ;IF SO, TIME TO EXIT
0302 FE0A               CPI     0AH     ;IS IT END OF LINE ?
0304 CAF602             JZ      CPMPN   ;IF SO, IGNORE AND START NEW LINE
0307 FE09               CPI     9       ;IS IT HORIZ TAB ?
0309 CA6B03             JZ      EXTAB   ;GOTO EXPANSION
030C CD7F03             CALL    OUTPT   ;SEND IT
030F 3AC503             LDA     CPOS    ;GET CHAR POS
0312 3C                 INR     A       ;INCREMENT IT
0313 32C503             STA     CPOS    ;STORE IT
0316 C3FA02             JMP     CPMP1   ;NEXT CHARACTER
                ;
0319 AF         DONE    XRA     A       ;CLEAR ACCUMULATER
031A CD7F03             CALL    OUTPT   ;SEND PENCIL EOF
031D                    MESS    MES2    ;PRINT COMPLETION MESSAGE
0325                    FINIS   OUTF    ;CLOSE OUTPUT FILE
0366 2A8303             LHLD    OLDSP   ;GET OLD STACK POINTER
0369 F9                 SPHL            ;AND RESTORE IT
036A C9                 RET             ;RETURN TO CPM
                ;
036B 3E20       EXTAB   MVI     A,20H     ;LOAD SPACE
036D CD7F03             CALL    OUTPT   ;SEND IT
0370 3AC503             LDA     CPOS    ;GET CHAR POSITION
0373 3C                 INR     A       ;INCREMENT IT
0374 32C503             STA     CPOS    ;STORE IT
0377 E607               ANI     7       ;TAB COLUMN?
0379 CAFA02             JZ      CPMP1   ;NEXT CHARACTER
037C C36B03             JMP     EXTAB   ;MORE SPACE
                ;
037F            OUTPT   PUT     OUTF    ;SEND CHAR TO OUTPUT FILE
0382 C9                 RET
                ;
0383            OLDSP   DS      2       ;STORAGE FOR OLD STACK POINTER
0385                    DS      40H     ;NEW STACK LOCATION
                NEWSP                   ;NEW STACK POINTER
03C5 00         CPOS    DB      0       ;CHARACTER POSITION STORAGE
                ;
03C6 0D0A434F4EMES1     DB      CR,LF,'CONVERTING CP/M FILE TO ELECTRIC PENCIL '
03F0 464F524D41         DB      'FORMAT . . . . . ',CR,CR,LF,'$'
0404 0D0A3C2D2DMES2     DB      CR,LF,'<--- CONVERSION COMPLETE --->',CR,LF,'$'
                ;
                BUFFERS
                ;
0426                    END
```

The correct Command syntax for CPM-PEN is "CPM-PEN Filename.Filetype" and the converted file will be named "Filename.PCL. Command syntax for PEN-CPM is "PEN-CPM Filename.PCL Filename.XXX" and the converted file will be named "Filename.XXX". Filename for the output file in this case does not have to be the same as the PCL Filename. I sometimes use this feature to avoid writing over my original file.

The following addition to the SEQIO.LIB file supplied with Digital Research's "MAC" macro assembler will print messages pointed to by the D-E register pair. The message should end in a $. Correct syntax would be "MESS XXXX" with XXXX as the label identifying the beginning of the message. The message is entered into the program thus:

```
XXXX    DB      CR,LF,'THIS IS A MESSAGE. $'
```

This addition is used in PEN-CPM.ASM and CPM-PEN.ASM.

```
;
MESS    MACRO M         ;PRINT MESSAGE M
;
        LXI     D,M     ;POINT TO MESSAGE
        MVI     C,@MSG  ;LOAD CPM PRINT CODE
        CALL    BDOS    ;DISK PRIMITIVE ENTRY POINT
;
        ENDM
;
;
```

Rod Hallen
P. O. Box 73
Tombstone, AZ 85638

27 July 1979

Continued from pg. 38

```
0183:  03BA 47              MOV B,A
0184:  03BB D1              POP D
0185:  03BC F1              POP PSW
0186:  03BD 85              ADD L
0187:  03BE 6F              MOV L,A
0188:  03BF C9              RET
0189:
0190:                  WRITEREC
0191:  03C0 F5C5D5E5        PUSH PSW ! PUSH B ! PUSH D ! PUSH H
0192:  03C4 0E1A            MVI C,SETDMA
0193:  03C6 114401          LXI D,OUTBUF
0194:  03C9 CD0500          CALL BDOS
0195:  03CC 0E15            MVI C,WRTFILE
0196:  03CE 112301          LXI D,DEST
0197:  03D1 CD0500          CALL BDOS
0198:  03D4 FEFF            CPI 0FFH
0199:  03D6 CAF903          JZ ERROR3
0200:  03D9 FE02            CPI 2
0201:  03DB CAFF03          JZ ERROR4
0202:  03DE FE01            CPI 1
0203:  03E0 CA0504          JZ ERROR5
0204:  03E3 E1D1C1F1        POP H ! POP D ! POP B ! POP PSW
0205:  03E7 0E80            MVI C,128
0206:  03E9 114401          LXI D,OUTBUF
0207:  03EC C9              RET
0208:
0209:  03ED 110602     ERROR1  LXI D,MSSG1
0210:  03F0 CD2504             CALL MESSAGE
0211:  03F3 112002     ERROR2  LXI D,MSSG2
0212:  03F6 CD2504             CALL MESSAGE
0213:  03F9 113802     ERROR3  LXI D,MSSG3
0214:  03FC CD2504             CALL MESSAGE
0215:  03FF 115202     ERROR4  LXI D,MSSG4
0216:  0402 CD2504             CALL MESSAGE
0217:  0405 115E02     ERROR5  LXI D,MSSG5
0218:  0408 CD2504             CALL MESSAGE
0219:  040B 117802     ERROR6  LXI D,MSSG6
0220:  040E CD2504             CALL MESSAGE
0221:  0411 119302     ERROR7  LXI D,MSSG7
0222:  0414 CD2504             CALL MESSAGE
0223:
0224:                  CLOSEFILE
0225:  0417 0E10            MVI C,CLOSE
0226:  0419 112301          LXI D,DEST
0227:  041C CD0500          CALL BDOS
0228:  041F 0E0D            MVI C,RESET
0229:  0421 CD0500          CALL BDOS
0230:  0424 C9              RET
0231:
0232:                  MESSAGE
0233:  0425 0E09            MVI C,PRTBUF
0234:  0427 CD0500          CALL BDOS
0235:  042A 0E0D            MVI C,RESET
0236:  042C CD0500          CALL BDOS
0237:  042F 2A0402          LHLD STKSAVE
0238:  0432 F9              SPHL
0239:  0433 C9              RET                    ; RETURN TO CP/M
0240:
0241:  0434                 END
0242:
```

SYMBOL TABLE

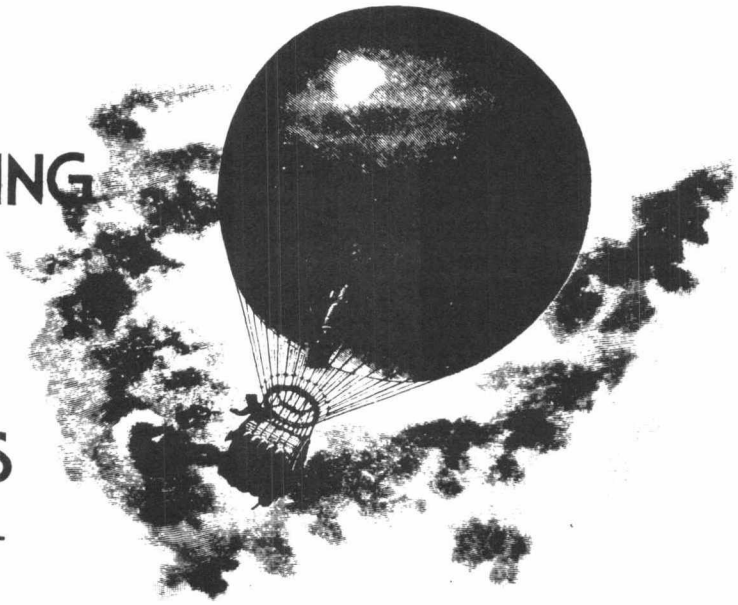
```
0371 ASTK        0005 BDOS        0020 BL          037C BLANK       0010 CLOSE
0417 CLOSEFILE   0013 DELETE      0123 DEST        03A6 EOFMARK     0122 EOLCNT
03ED ERROR1      03F3 ERROR2      03F9 ERROR3      03FF ERROR4      0405 ERROR5
040B ERROR6      0411 ERROR7      02B3 GETNAMES    0080 INBUF       0016 MAKE
0425 MESSAGE     032F MOVEREC     02CB MOVNAME     0206 MSSG1       0220 MSSG2
0238 MSSG3       0252 MSSG4       025E MSSG5       0278 MSSG6       0293 MSSG7
0339 NEWLINE     0203 NEWSTK      0386 NOTAB       035D OLDLINE     000F OPEN
02E1 OPENFILE    0144 OUTBUF      0009 PRTBUF      0014 READFIL     030C READREC
000D RESET       03B2 RESTORENEG  001A SETDMA      005C SOURCE      0204 STKSAVE
0121 TAB         03C0 WRITEREC    0015 WRTFILE     02DA ZEROS
```

```
                ;
                ;
                ;* - - - - - - - - - - - - - - - - - - - - - - - *
                ;*                                               *
                ;*                  - PEN-CPM -                  *
                ;*                                               *
                ;*    ROUTINE TO TRANSLATE ELECTRIC PENCIL       *
                ;*            FILES TO CP/M FORMAT               *
                ;*             (C) COPYRIGHT 1979                *
                ;*  ROD HALLEN  BOX 73  TOMBSTONE, AZ  85638     *
                ;*               26 JULY 1979                    *
                ;*     THIS PROGRAM MAY BE USED FREELY BY        *
                ;*    COMPUTER HOBBYISTS FOR NON-COMMERCIAL      *
                ;*    APPLICATIONS.  IT MAY NOT BE USED FOR      *
                ;*     COMMERCIAL PURPOSES OR SOLD WITHOUT       *
                ;*      WRITTEN PERMISSION OF  THE AUTHOR.       *
                ;*                                               *
                ;*     ASSEMBLE WITH DIGITAL RESEARCH  "MAC"     *
                ;*     ASSEMBLER AND SEQIO LIBRARY. COMMAND      *
                ;*        FORMAT IS "PEN-CPM X.PCL Y.Z"          *
                ;*          OUTPUT FILE WILL BE  "Y.Z".          *
                ;*                                               *
                ;* - - - - - - - - - - - - - - - - - - - - - - - *
                ;
                ;
                ;
0100                    ORG     100H
                ;
                        MACLIB  SEQIO   ;SEQUENTIAL I/O LIBRARY
                ;
0100 210000             LXI     H,0     ;SAVE OLD SP
0103 39                 DAD     SP      ;AND SET UP NEW SP
0104 226003             SHLD    OLDSP
0107 31A203             LXI     SP,NEWSP
                ;
010A            FILE    INFILE,INF,,1,,1024     ;SETUP PENCIL INPUT FILE
                ;
01F0            FILE    OUTFILE,OUTF,,2,,1024   ;SETUP CPM OUTPUT FILE
                ;
02EE                    MESS    MES1    ;PRINT ACTIVITY MESSAGE
                ;
02F6            PENCP   GET     INF     ;GET CHAR FROM INPUT FILE
02F9 FE00               CPI     0       ;PENCIL END OF FILE ?
02FB CA0E03             JZ      DONE    ;IF SO, TIME TO EXIT
02FE FE0D               CPI     0DH     ;IS IT END OF LINE ?
0300 CD5C03             CALL    OUTPT   ;SEND IT TO OUTPUT FILE
0303 C2F602             JNZ     PENCP   ;IF NOT EOL, SKIP LINE FEED
0306 3E0A               MVI     A,0AH   ;LOAD LINE FEED
0308 CD5C03             CALL    OUTPT   ;SEND IT
030B C3F602             JMP     PENCP   ;NEXT CHARACTER
                ;
030E            DONE    MESS    MES2    ;PRINT COMPLETION MESSAGE
0316                    FINIS   OUTF    ;CLOSE OUTPUT FILE
0357 2A6003             LHLD    OLDSP   ;GET OLD STACK POINTER
035A F9                 SPHL            ;AND RESTORE IT
035B C9                 RET             ;RETURN TO CPM
                ;
035C            OUTPT   PUT     OUTF    ;SEND CHAR TO OUTPUT FILE
035F C9                 RET
                ;
0360            OLDSP   DS      2       ;STORAGE FOR OLD STACK POINTER
0362                    DS      40H     ;NEW STACK LOCATION
                NEWSP                   ;NEW STACK POINTER
                ;
03A2 0D0A434F4EMES1     DB      CR,LF,'CONVERTING ELECTRIC PENCIL FILE TO '
03C7 43502F4D20         DB      'CP/M FORMAT . . . . . ',CR,CR,LF,'$'
03E0 0D0A3C2D2DMES2     DB      CR,LF,'<--- CONVERSION COMPLETE --->',CR,LF,'$'
                ;
                BUFFERS
                ;
0402                    END
```

* * * \

PROGRAMMING PASTIMES AND PLEASURES

BY CHARLES WETHERELL



The Programmer's Book of Rules

This month I would like to laud a singularly courageous act by a friend of mine. George Ledin, Jr. and his brother Victor have just published *A Programmer's Book of Rules* (Lifelong Learning Publications, 1979. $7.95). Before I tell you why I think George is so brave, let me describe the book. Then we shall anoint the hero.

Handbook and Sourcebook

The most striking thing about *Rules* is its layout. On the left of each double page are large boxes with a programming rule in big black type; for example:

Nest Loops Sparingly.

Often there will be short subsidiary rules and references to the source of the rule. On the right hand page there is an explanation box for each rule box. The explanation is usually English text, but it may contain a short program example, an input sample, or an output fragment. The explanations provide the reasons for the rules.

The book is broken into three major sections—one each for requirements analysis, coding style, and programming. The sections fall into chapters; with each chapter comes a bibliography giving exact sources for each rule in the chapter. At the end of the book there is a collected bibliography divided into useful topics. There is also a complete index of authors and an index of subject headings. *Rules* is valuable for its bibliographies alone.

When you first hold a copy of *Rules*, you will probably want to read it straight through—two hours will do nicely. Then you may well want to argue the rules with your friends. Finally you will probably want to write your own *Book of Rules*. In fact, I predict that many readers will rip the spine off *Rules*, hole-punch the pages, put the resulting handbook in a loose-leaf binder, and start adding rules of their own. George and Victor surely would approve. *Rules* not only suggests keeping historical notebooks, but the authors promise to revise *Rules* from time to time and ask their reader's help. So keep those cards and letters coming, folks.

Where Angels Fear To Tread

What makes the Ledins so heroic? On the day I received my copy of *Rules*, an acquaintance dropped into my office. He picked up the book, riffled the pages, glanced at a few rules, tossed it down, and said, "That can't be right; at least 75% of the rules are all wet." I fear that *Rules* is going to receive the same reaction from many old-timers (perhaps even from some of the quoted rule-makers). But such quick criticism is beside the point; let me explain why.

First, every programmer needs discipline—discipline in the monkish sense of a rule of life. (By the way, George Ledin is a professor at University of San Francisco, an eminent Catholic school. Make of that what you will.) Almost any rule is better than no rule at all. In fact, my master rule (or meta-rule, if you will) is

If you did something before and you have to
do it again, Do It The Same Way.

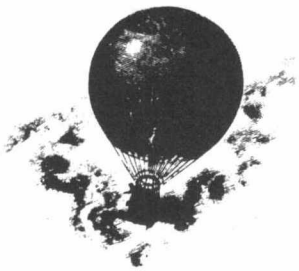
In other words, make rules and stick to them.

Second, George and Victor didn't make the rules up themselves. They distilled the experience of many other expert programmers. If you don't believe the explanation for a rule, they point you right to the original source, chapter and verse. You can go read the rulemaker's arguments and decide for yourself. One of the most important rules is "Learn From Experience, Yours And Others' ". *Rules* helps you do that.

© 1979, Charles Wetherell

Third, no rule is an absolute – neither " Brush after every meal" nor "Thou shalt not steal." Programming rules are very susceptible to exception. Criticism like the example at the beginning of this section is very likely based on experience of exceptions, but that doesn't mean the rules are all wrong. Judgment in application is needed. For example, I agree wholeheartedly with the book's rule "Avoid FORTRAN arithmetic IF," but . . . when I write a binary search, it is the perfect control structure. When you acquire the judgment and experience, you too can break the rules.

So I fear the Ledin's will receive much criticism and censure. But they have done a service and they must be praised. I have had a file of similar rules for five years and I have been afraid to print. Not they.



Hair Splitting And Head Shrinking

So that George's head doesn't swell too big from all this praise, I am now going to engage in a bit of criticism. Of course, the Preface says comments are welcomed. So George will now be noble for several paragraphs and take his licks without flinching.

Rules contains a lot of material on control structures. Yet I missed an overriding rule – one that says clean control has clean mathematical structure. Similarly, in the section on debugging and testing there is no mention that the goal is to write programs that are so clear and obvious that it is evident on their face they contain no errors. In a very important sense, debugging and testing are a waste of time; if your program were correct from the start, you wouldn't need to kill bugs. While I think the mathematical theory of programming is nowhere near developed enough to cope with large practical programs, programmers must always keep the ideal of the perfect program in mind. *Rules* does not emphasize this ideal enough.

There are a few rules I believe must have slipped by when George wasn't looking. "Minimize compilation time" is one such. Whatever for? Even very small computers have quite fast compilers nowadays, and anyway, who cares how long the compiler takes? Run-time, Si! Compile-time, No!

On the same quibble level, *Rules* suggests first that comments should explain the why and wherefore of each logical section of coding and that the code itself should express the actions necessary to achieve the object. Then *Rules* says one line of comment to 10 or 20 lines of code. I should have thought one to one was a much better ratio. Surely 5% comments are far too few. By the way, I do agree that a paragraph of comments should precede a paragraph of code; the two should generally not be mixed.

Throughout *Rules,* the Ledin's urge that clarity not be sacrificed to efficiency. They also recognize that sometimes improved efficiency is necessary. But nowhere can I find a mention of the way efficiency is best gained – wholesale replacement of algorithms and data structures by better ones. I am reminded of a friend who spent much effort delicately hand-coding an assembly language sort routine to extract every last microsecond of speed. No doubt he developed the fastest possible sort – using his algorithm. But along came a graduate student who coded a well-known better algorithm in ordinary FORTRAN. A factor of 3 speed-up and the new algorithm only used 40% of the data space. Finally, the new algorithm was simpler and cleaner. It often works that way. Speed and elegance go hand in hand because both depend on deep insight into the problem to be solved. So do follow *Rules'* precept to read the literature when you want more efficiency.

Hobbyists and Rules

In spite of the criticisms, I still think that *Dr. Dobb's* readers will find *Rules* profitable. My guess is that most of you are hobbyists and, even if involved with computers professionally, are not trained programmers. So you are probably having to learn all the painful lessons of programming for yourselves. *Rules* condenses much experience into palatable droplets of wisdom; it would be foolish to turn down such a healthy draught. Perhaps not all the ingredients will help you grow into better programmers, but like Mama's chicken soup, can it hurt?

Letters

CARRY FLAG STILL WAVES

Dear Suzanne:

We wouldn't want you not to have an 8080 pusher-popper routine to kick around anymore, so I would like to make this comment on the code submitted by John Bockelmann in *DDJ* #38. May I be the umpteenth programmer to point out it will always return with the carry flag cleared!! The DAD instruction used to get the old SP modifies the carry flag, and when DADing to 00, it will clear it. I don't have an optimum solution, but here is an idea of how we handle the problem in our Fast!tm software:

```
SHLD HLREG      ;save HL in RAM
PUSH PSW        ;save CARRY flag
                ;this uses 1 level of the old stack
LXI H,2         ;the 2 makes up for the PSCH PSW
DAD SP          ;HL = true stack
SHLD OLDSP      ;store old pointer in RAM
POP PSW         ;get it back
LXI SP, NEWSP   ;set new stack
```

Now the registers can be pushed in any order. What to do after that depends on whether this is a subroutine, part of inline code or whatever. Obviously, the code is not reentrant, requires RAM and is cumbersome. I am offering it as a stimulus to your readers to find a better way and to keep the ol' pusher-popper controversy going.

Sincerely,
Orv Balcom

Brown Dog Engineering
Box 427
Lomita, CA 90717

QUARREL WITH DAD

Dear *DDJ*:

I take exception to the routines submitted by M. John Bockelmann (*DDJ* #38) in the the "DAD SP" instruction found in the save routine will modify the setting of the carry flag before it is saved. Enclosed is a modified version of the routines that will cure the problem.

I enjoy your publication very much and consider it to be the best software journal for microprocessor programmers available today.

Sincerely,
Chester P. Quinn

3660 London Road
Chamblee, Georgia 30341

```
CP/M MACRO ASSEM 2.0    #001    SAVE & RESTORE MACHINE STATUS
                        TITLE   'SAVE & RESTORE MACHINE STATUS'
                ;
                ;  THIS ROUTINE WILL SAVE ALL REGISTERS IN THE 8080 AND INITIALIZE
                ;  THE STACK POINTER TO A LOCAL STACK.
                ;
0000 E3         SAVE:   XTHL            ;SAVE H&L
0001 D5                 PUSH    D       ;SAVE D&E
0002 C5                 PUSH    B       ;SAVE B&C
0003 F5                 PUSH    PSW     ;SAVE ACCUMULATOR & FLAGS
0004 EB                 XCHG            ;PUT RETURN ADDRESS IN D&E
0005 210000             LXI     H,0     ;GET AND SAVE THE CONTENTS OF THE STACK POINTER
0008 39                 DAD     SP
0009 222000             SHLD    SPHOLD
000C 314200             LXI     SP,LCLSTK       ;INITIALIZE THE LOCAL STACK
000F EB                 XCHG
0010 E9                 PCHL            ;PUT RETURN ADDRESS IN H&L AND GO THERE
                ;
                ;  THIS ROUTINE RESTORES THE MACHINE STATE USING THE SAVED
                ;  STACK POINTER ADDRESS "SPHOLD", AND THEN RETURNS TO THE CALLING
                ;  PROGRAM.
                ;
0011 2A2000     RESTR:  LHLD    SPHOLD  ;PUT SAVED STACK POINTER ADDRESS IN H&L
0014 F9                 SPHL            ;TRANSFER IT TO THE STACK POINTER
0015 F1                 POP     PSW     ;RESTORE ACCUMULATOR & FLAGS
0016 C1                 POP     B       ;RESTORE B&C
0017 D1                 POP     D       ;RESTORE D&E
0018 E1                 POP     H       ;RESTORE H&L
0019 C9                 RET             ;RETURN TO THE MAIN PROGRAM
                ;
                ;
                ;  THE FOLLOWING SUBROUTINE DEMONSTRATES THE USAGE OF THE TWO
                ;   SUBROUTINES SHOWN ABOVE.  THE MAIN PROGRAM CALLS THE ROUTINE CALLED
                ;  "LCLSUB".  "LCLSUB" THEN CALLS "SAVE" TO PRESERVE THE MACHINE STATE
                ;  AT THE TIME THAT IT IS CALLED.  WHEN "LCLSUB" HAS COMPLETED
                ;  PROCESSING, IT THEN JUMPS TO "RESTR" TO RESTORE THE MACHINE
                ;  STATE AND RETURN TO THE CALLING PROGRAM.
                ;
                ;
                ;       ANY LOCAL ROUTINE
                ;
001A CD0000     LCLSUB: CALL    SAVE    ;SAVE THE MACHINE STATE
                                        ;PERFORM ANY PROCESSING REQUIRED AT THIS
                                        ;POINT.
001D C31100             JMP     RESTR   ;LOCAL PROCESSING COMPLETE - RETURN TO CALLING
                                        ;PROGRAM WITH THE MACHINE STATE RESTORED
                ;
                ;       MAIN SYSTEM STACK HOLD AREA & LOCAL STACK
                ;
0020 0000       SPHOLD: DW      0       ;MAIN PROGRAM STACK ADDRESS HOLD AREA
0022                    DS      32      ;RESERVE 32 BYTES OF STACK
0042 =          LCLSTK  EQU     $       ;AND TAG THE TOP OF THE STACK
0042                    END
```

JULIAN DATE

Dr. Dobb's:

The letter by Jack Dekok in *DDJ* #39 reminded me that The Government, in its usual wisdom, has managed to confuse things. Julian date is a precisely defined astronomical term. According to the Explanatory Supplement to the *American Ephemeris and Almanac*, the Julian Day Number is the number of days since Noon, January 1, 4713 B.C. (Julian proleptic calendar). The Julian Date at any instant is the Julian day number, followed by the decimal fraction of the day elapsed since the preceding noon. Julian dates are always referred to as Greenwich Mean Time or Universal Time. Noon was chosen as the time a new date starts because the Greenwich astronomers did not like to have their date change in the middle of their night-time observations.

As Mr. Dekok says, Julian dates are much more convenient to use than conventional Gregorian calendar dates for the computation of intervals in days. This letter is being written at about 8 pm EDT, 25 September 1979 or JD2444142.5. It is easy to find that 1 January 1984 (JD2445700.5) will be upon us in only 1558 days. Unlike the governmental version (which is better called the Annual Day Number, or simply the Day number) the true Julian date does not require remembering which years are leap years. There are several good formulae for conversion of JD to calendar date and vice versa. A useful reference is *Matrix Magazine* Issue #3 (published by Michael and Margaret Erlewine, 1041 North Main Street, Ann Arbor MI 48104). Astronomers will find JD especially useful in generating positions of the Sun, Moon, and the planets with series expansions from the Explanatory Supplement. *Matrix* has also published useful formulae for such purposes.

I have no idea why the government created its own "Julian date." I suspect it was a case of someone who vaguely remembered AST 1 Lab . . . but not too well. If anyone knows more about the origins of the error, I would appreciate hearing from them .

Sincerely,
John P. Oliver

Astronomical Time Mechanisms
P.O. Box 12248, University Station
Gainesville, FL 32604

ADDITIONAL NORTHSTAR DOS PATCHES

Dear *DDJ*,

I would like to make some additions to Mr. A.M. Banks' Northstar DOS patches (*DDJ* #39, page 43). The changes to be made are as follows:

WAS:		CHANGE TO:	
	2601 FE 01		25DB FE 02
	2603 C2 11 26		25DD CA EE 29
	2611 CD 0C 27		25EB CD F6 29
	29F3 C3 14 26		29F3 C3 EE 25
	29F6 06 07		29F3 C3 EE 25
	29F8 CD 04 27		29F6 0E 07

Also to incorporate the 'GOTO' data change:

217A FE 01	217A FE XX
217C C2 89 21	217C D2 89 21

The 'XX' can be any number since I use different type numbers for electric pencil files, proc. tech. music files, and other BASIC files such as tiny BASIC. The reason for 'FE 02 CA EE 29' is that type 2 files use byte number 13 for the number of active blocks in a BASIC program. A BASIC statement such as 'A = CALL(8229.1)' will display the directory from a BASIC program.

Thanks again for the magazine and keep up the good work.

Sincerely,
Jim Gammell

PO Box 384
Berwick PA 18603

STRING ARRAYS IN NORTHSTAR BASIC

Dear *DDJ:*

This letter is in reply to a remark made in passing in one of the book reviews in the September 1979 issue of *Dr. Dobb's Journal.* At the end of the first paragraph in the second review on page 43, the remark was "Did you ever try to figure out how to handle string arrays in Northstar BASIC?" Well, I have, so I thought I would share these routines.

These examples are just to illustrate my approach; the details will probably have to be changed in an individual program, but they can be used as starting points. The idea is to store all of the string entries consecutively in a single long string variable and use an associated numeric array for pointers to the end of each string element.

The arrays can be set up by routine #1. If the program requires a string array with fixed data, the same loop can use READ/DATA statements instead of the INPUT statement. If the total number of entries is not known in advance, use a loop controlled by IF/THEN statements to check for an ending signal (an input of END, QUIT or something similar).

The required string element can be accessed by using the expression A$(N-1)+1,A(N)), with N being the number of the entry. This works because A(N - 1) is the pointer to the last character of the entry ahead of the required entry, so A(N - 1)+1 points to the first character and A(N) points to the last character of the required entry. Note that for the first entry, N=1 so A(N - 1) = A(0), this must be zero. Of course, it is automatically set to zero when the array is dimensioned, but it is a good idea to keep this requirement in mind anyway.

If this expression is required repeatedly (it probably will be) it is simpler to use a defined function such as DEF FNA$(N)=A$(A(N-1)+1,A(N)). This can be called by expressions such as PRINT FNA$(N) or T$=FNA$(N).

It is rather more difficult to dynamically alter the array string, but the defined function, shown as routine #2, will do it. This can be called by expressions such as X=FNA(T$,N), where T$ is the new string to be inserted and N is the number of the entry to be replaced. If the number of entries (N9 in this example) is unknown or otherwise not readily available, the pointer array can be set up with an extra entry at the end which is set to zero or negative. Then the FOR/NEXT loop and line 140 can be replaced with IF/THEN statements to check for this end marker in the pointer array.

I want to emphasize that these examples are intended strictly as starting points. Depending on the requirements in your program, they may have to be extended greatly. They do work in Northstar BASIC, but I don't know how much they need to be changed for other BASICs which may also require a similar approach to simulate string arrays. Also, I haven't shown the obvious additional requirements, such as properly dimensioning everything first.

Compared to a BASIC which directly handles string arrays, these routines are somewhat inconvenient and inefficient, but they do work. So now you can go ahead and copy those programs that you haven't copied before because they used string arrays! Good luck, and have fun!

A little further explanation may be helpful to those who are not familiar with the way the string functions are used in Northstar BASIC. First, dimensioning a string does not set up an array of strings, but sets up a single string. The specified dimension is the maximum length of this string. Substrings are invoked by either 1 or 2 subscripts, as in A$(2,5) or A$(4). The first example results in the second through the fifth characters of A$, and the second example results in the fourth through the last characters of A$. This same format can be used in assignment statements also. For example, assume A$ is ABCDEFG, the statement A$(3,5)="123" results in A$ now being AB123FG.

Sincerely,
Larry Hudson

2232 Alameda Avenue
Ventura, CA 93003

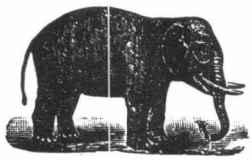
Hudson adds the following information in a note: "Until I got my Altair I had no experience or training with computers. Even after I got it, everything I have learned has been from books and my own playing around with it. BASIC is the only high-level language I am familiar with, but I do most of my programming in 8080 assembly language. I even wrote my own assembler, also the editor I am using to write this letter."

ROUTINE #1

```
100 A$=""\REM - set the array string to a null string
110 FOR I=1 TO N
120   INPUT T$
130   A$=A$+T$\A(I)=LEN(A$)
140 NEXT I
```

ROUTINE #2

```
100 DEF FNA(T$,N)
110   D=LEN(T$)+A(N-1)-A(N)\REM - D=difference btwn old & new
120   IF N=1 THEN 130 ELSE 140\REM - special handling for 1st
130   A$=T$+A$(A(N)+1)\GOTO 170
140   IF N=N9 THEN 150 ELSE 160\REM - spcl handling for last
150   A$=A$(1,A(N-1))+T$\GOTO 170
160   A$=A$(1,A(N-1))+T$+A$(A(N)+1)\REM - insert new string
170   FOR I=N TO N9\REM - N9=number of entries
180     A(I)=A(I)+D\REM - update pointer array
190   NEXT I\RETURN 0
200 FNEND
```



A CORRECTION TO KEAY'S TABLE

Dear Suzanne:

Your letter dated September 10 re the error in the Machine Language Program Displacement Table (*DDJ* #38) reached me a few days ago via my home address in Australia and I have just received a copy of the September issue of *DDJ*. In fact a second error has crept into the same paragraph. But they were not the fault of the Table. The Table gives correct results every time if the directions are followed properly.

The two errors in the paragraph beginning "Besides calculating program displacements . . ." baffle me. The article was checked God knows how many times after successive drafts were typed. But I must have slipped up in the final draft which I mailed to you. I haven't a copy of the typescript with me in Canada but I seem to recall inserting the word "(hex)" in the final version to make it clearer, which suggests the error got into the penultimate version which followed a thorough check of the hexadecimal arithmetic using a TI hex calculator by a colleague who owns one. Anyway, that offending paragraph, for which I must shoulder the blame, should contain 37B9 in place of 27B9 and in the last line 9 rather than 19 (hex) bytes. Here is a corrected version of the offending paragraph:

Besides calculating program displacements the table is useful for determining the length of program segments, subroutines, etc. For example, can a program extending from 37B9 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 equals 9 (hex) bytes to spare.

Yours sincerely,
Colin Keay,

Visiting Scientist,
Planetary Sciences Section

U.S. POSTAL SERVICE
STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION
(Required by 39 U.S.C. 3685)

1. TITLE OF PUBLICATION: Dr. Dobb's Journal of Computer Calisthenics & Orthodontia	A. PUBLICATION NO.: 3 0 7 6 9 0	2. DATE OF FILING: 28 Sep 79
3. FREQUENCY OF ISSUE: monthly except July and December	A. NO. OF ISSUES PUBLISHED ANNUALLY: 10	B. ANNUAL SUBSCRIPTION PRICE: $15.00

4. LOCATION OF KNOWN OFFICE OF PUBLICATION (Street, City, County, State and ZIP Code) (Not printers): 1263 El Camino Real, Menlo Park, CA 94025 (San Mateo County)

5. LOCATION OF THE HEADQUARTERS OR GENERAL BUSINESS OFFICES OF THE PUBLISHERS (Not printers): same as above

6. NAMES AND COMPLETE ADDRESSES OF PUBLISHER, EDITOR, AND MANAGING EDITOR

PUBLISHER (Name and Address): Willard J. Holden, 1263 El Camino Real, Menlo Park, CA 94025 (San Mateo County)

EDITOR (Name and Address): Suzanne Rodriguez (same)

MANAGING EDITOR (Name and Address): Suzanne Rodriguez (same)

7. OWNER (If owned by a corporation, its name and address must be stated and also immediately thereunder the names and addresses of stockholders owning or holding 1 percent or more of total amount of stock. If not owned by a corporation, the names and addresses of the individual owners must be given. If owned by a partnership or other unincorporated firm, its name and address, as well as that of each individual must be given.)

NAME	ADDRESS
People's Computer Company	1263 El Camino Real, Menlo Park, CA 94205

8. KNOWN BONDHOLDERS, MORTGAGEES, AND OTHER SECURITY HOLDERS OWNING OR HOLDING 1 PERCENT OR MORE OF TOTAL AMOUNT OF BONDS, MORTGAGES OR OTHER SECURITIES (If there are none, so state)

NAME	ADDRESS
none	

9. FOR COMPLETION BY NONPROFIT ORGANIZATIONS AUTHORIZED TO MAIL AT SPECIAL RATES (Section 132.122, PSM)
The purpose, function, and nonprofit status of this organization and the exempt status for Federal income tax purposes (Check one)

[X] HAVE NOT CHANGED DURING PRECEDING 12 MONTHS [ ] HAVE CHANGED DURING PRECEDING 12 MONTHS (If changed, publisher must submit explanation of change with this statement.)

10. EXTENT AND NATURE OF CIRCULATION	AVERAGE NO. COPIES EACH ISSUE DURING PRECEDING 12 MONTHS	ACTUAL NO. COPIES OF SINGLE ISSUE PUBLISHED NEAREST TO FILING DATE
A. TOTAL NO. COPIES PRINTED (Net Press Run)	14,725	13,700
B. PAID CIRCULATION 1. SALES THROUGH DEALERS AND CARRIERS, STREET VENDORS AND COUNTER SALES	4,350	3,081
2. MAIL SUBSCRIPTIONS	8,844	8,265
C. TOTAL PAID CIRCULATION (Sum of 10B1 and 10B2)	13,194	11,346
D. FREE DISTRIBUTION BY MAIL, CARRIER OR OTHER MEANS SAMPLES, COMPLIMENTARY, AND OTHER FREE COPIES	200	300
E. TOTAL DISTRIBUTION (Sum of C and D)	13,394	11,646
F. COPIES NOT DISTRIBUTED 1. OFFICE USE, LEFT OVER, UNACCOUNTED, SPOILED AFTER PRINTING	1,028	1,948
2. RETURNS FROM NEWS AGENTS	351	106
G. TOTAL (Sum of E, F1 and 2—should equal net press run shown in A)	14,725	13,700

11. I certify that the statements made by me above are correct and complete. SIGNATURE AND TITLE OF EDITOR, PUBLISHER, BUSINESS MANAGER, OR OWNER: Willard J. Holden

12. FOR COMPLETION BY PUBLISHERS MAILING AT THE REGULAR RATES (Section 132.121, Postal Service Manual)

39 U. S. C. 3626 provides in pertinent part: "No person who would have been entitled to mail matter under former section 4359 of this title shall mail such matter at the rates provided under this subsection unless he files annually with the Postal Service a written request for permission to mail matter at such rates."

In accordance with the provisions of this statute, I hereby request permission to mail the publication named in Item 1 at the phased postage rates presently authorized by 39 U. S. C. 3626.

SIGNATURE AND TITLE OF EDITOR, PUBLISHER, BUSINESS MANAGER, OR OWNER: Willard J. Holden PUBLISHER

PS Form 3526 Mar. 1977 (Page 1) (See instructions on reverse)

SAVE AN UNALTERED 8080

Editor:

Here is a routine that saves the state of an 8080, yet leaves it unaltered. As I mentioned in my last letter, such a routine might be useful in a debugger or break-point monitor where the current machine state must be obtained in a non-destructive manner.

Cheers,
Jon Bokelman

P.O. Box 1057
Menlo Park, CA 94025

```
        ORG     ROM     ;any ROM or RAM address
SAVE:   SHLD    SAVHL   ;save HL reg
        STA     SAVA    ;save contents of accumulator
        LXI     H,8000H ;use HL to reflect status of flag bits
        JM      TSTZ    ;test S flag bit, jump if 1
        MOV     H,L     ;H=0 means S=0
TSTZ:   JNZ     TSTP    ;test Z flag bit, jump if 0
        MVI     L,40H   ;L=40H means Z=1
TSTP:   JPO     TSTC    ;test P flag bit, jump if 0
        INX     H       ;L=04H means P=1
        INX     H
        INX     H       ;(INX alters no flags)
        INX     H
TSTC:   JNC     TSTH    ;test C flag bit, jump if 0
        INX     H       ;L=01H means C=1
;at this point H and L contain the 'testable' flag bits
;the N and H flag bits are obtained by deductive logic
TSTH:   MVI     A,240   ;value to test N and H flag bits
        DAA             ;execute DAA and see what happens
        ANI     82H     ;mask the significant result bits
        RAL             ;juggle to correct position
        RAL
        RAL
;accumulator now contains N and H flag bits
        ORA     L       ;add C, Z and P flag bits
        ORA     H       ;add S flag bit
        STA     SAVF    ;save reconstructed flag register
;all flags saved, now save other registers
        LXI     H,0     ;get all zeros-
        DAD     SP      ; to get SP register
        LXI     SP,SAVF ;(assume interrupts disabled)
        POP     PSW     ;restore PSW
        PUSH    PSW     ;and resave to fix bits 5, 3 and 1
        PUSH    B       ;save BC register pair
        PUSH    D       ;save DE register pair
        PUSH    H       ;save SP register (from DAD above)
        SPHL            ;restore original SP contents
        LHLD    SAVHL   ;restore original HL contents
;all regs saved, machine state unaltered
;continue in-line code or return
        ORG     RAM     ;any RAM address
SAVSP:  DS      2       ;saved SP
SAVDE:  DS      2       ;saved DE
SAVBC:  DS      2       ;saved BC
SAVF:   DS      1       ;saved Flags
SAVA:   DS      1       ;saved Accumulator
SAVHL   DS      2       ;saved HL
```

BASIC AIN'T *THAT* SLOW

Dear Suzanne:

In the October issue of *Dr. Dobb's* (#39) on page 30 there is an article by David Morganstein regarding using OSI BASIC with a modem. The statement was that BASIC is too slow to keep up with even 300 baud. This is not true. Declaring any numbers used in the first few lines of the program will help to speed things up somewhat. However rather than using PEEK to check the status use WAIT. Example:

```
10      P = 64512:M = 127:R = 13

800     REMEMBER TO INITIALIZE VARIABLES
810     Q$ = "" ""

900     WAITP,1
910     X = PEEK (P + 1) ANDM
920     IFX = RTHEN?:RETURN
930     IFF = 1THEN?CHR$(X);
940     Q$ = Q$ + CHR$ (X)
950     GOTO900
```

Putting this routine in the beginning of the program should also help. There are several routines in OSI BASIC that by doing one or two POKES allows you to write a string to memory rather than the CRT. By combining these features, BASIC can receive, store and print without missing any characters at greater than 300 baud. This is not to say that this would be preferable to assembler, only that it is possible.

Yours truly,
L. Barker

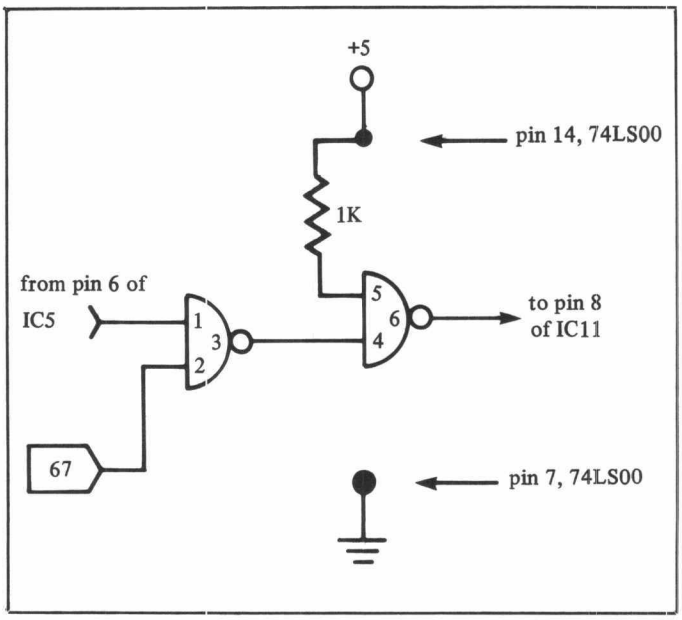
Chicago, Illinois

INSTALLING A "PHANTOM-DESELECT"

Dear Editor,

I recently found it necessary to install a "phantom-deselect" on one of the excellent GODBOUT ECONORAM II boards. The following circuit has worked well, with no problems.

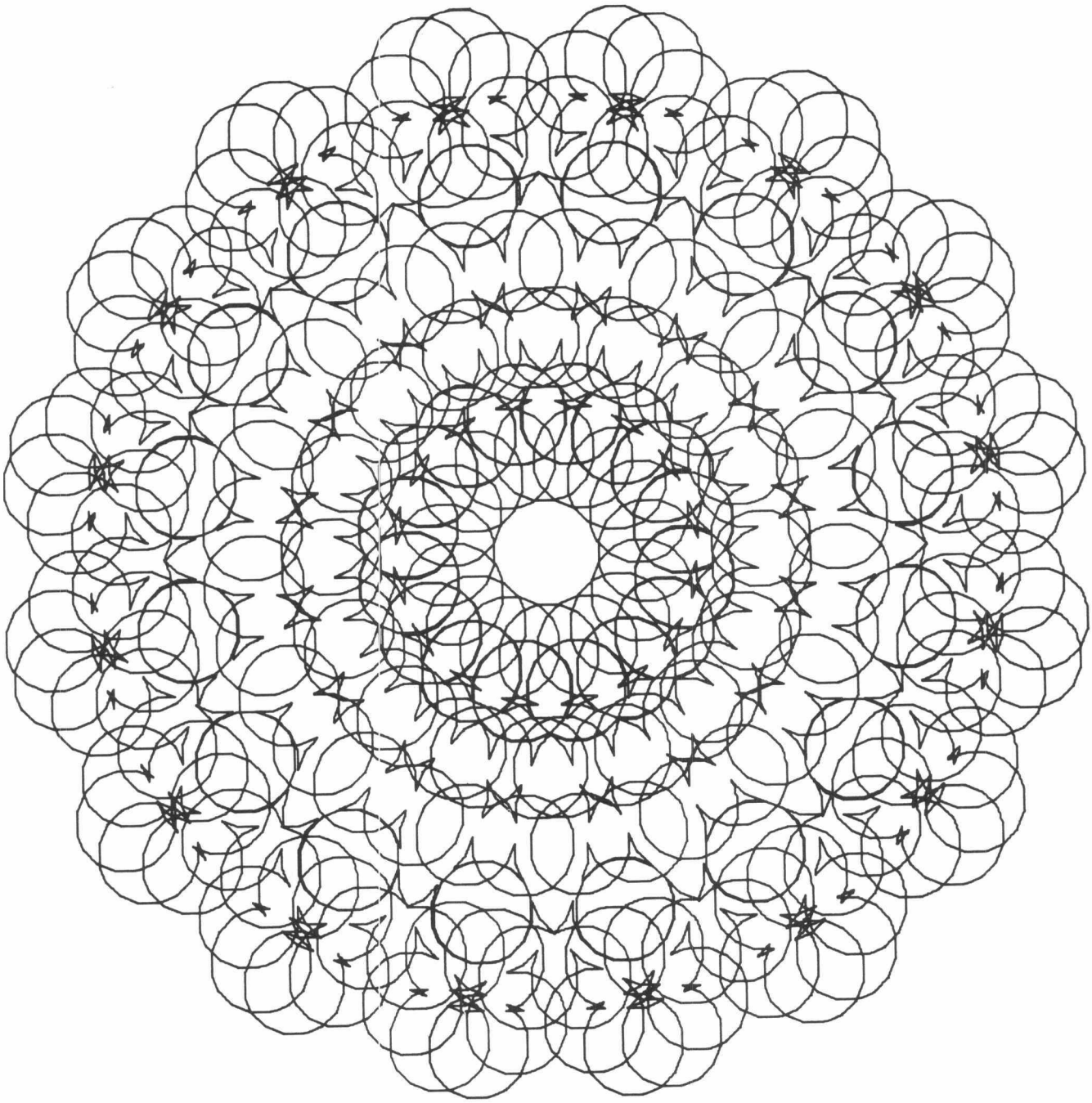
1. Cut trace from pin 6 of IC5.
2. Cement a 74LS00 (on its back) to the board next to IC11. (It fits nicely on the ECONORAM II logo).
3. Connect as follows:



Board now works with Tarbell floppy-disk controller.

Sincerely
Steve Newberry

Newberry Microsystems
24225 Summerhill
Los Altos, CA 94022



© David Dameron 1979

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 42 | February 1980 | Volume 5, Issue 2

A REFERENCE JOURNAL FOR USERS OF SMALL COMPUTERS

TI 9900: Fully Reentrant Formatter Operating Under TIPMX 5
Norman V. Walters

A Table Driven Assembler On CP/M 18
Darrel J. Van Buer

Feature Enhancements for the ADM-3A 15
John Wakerly

A User Interface to Apple-II Renumbering 26
Dewitt S. Brown

Converting Binary to Decimal 32
Dennis Allison

USCD PASCAL Terminal Parameters 34
Ronald G. Parsons

A Microprocessor-Assisted Graphics System 38
Griffith Hamlin and Thomas Crockett

... and, as always, even more

TI 9900 FULLY REENTRANT FORMATTER OPERATING UNDER TIPMX

BY NORMAN V. WALTERS
Software Engineer
R&D Department
Fairfield Industries, Inc.
Houston, TX 77042

Our company felt the need for a program that would allow assembly language programs operating in a real-time environment to display information to and request inputs from the user. The end result has been a powerful tool that is used in a large portion of our ongoing efforts. The code has been debugged over a period of six months in actual usage by a number of different users and should present no major "gotchas" to the unwary user.

There are five externally referenced routines that FRMTR calls during its execution. They are the following: B$RAM, CIO$$, ENTSM, PANIC, and RIO$. Since these called routines would vary from system to system, they are not given, but are instead described for the user to write for his or her own application.

B$RAM is the separator for stack and relative addressing. CIO$$ is the character I/O routine. The character to be input will be in the most significant byte of R7. The routine will input a character to the most significant byte of R7. The inputs and outputs are always ASCII. R7 should be the only register to change and then only on input. ENT$M is the workspace and stack allocator. PANIC generates an error message for the selected output device. The message does not depend on the formatter for generation of the message since it is showing an internal error of the formatter. If the user does not wish to generate this routine, the formatter code may be changed to

```
B  @ERR9
```

and the user can designate ERR9 as an internal error. RIO$ is the cursor I/O routine.

The system used for editing and assembly of this program was a TI 990/4 with dual floppies and 810 line printer. The program was debugged on TI 9900/101's with one or two Micropolis mini-floppies, 833ASR with dual cassettes, ADM-1A terminal and SCI line printer.

The code could probably be maximized in some areas, but we wished to get an operating program in use without incessant polishing. The following documentation is the full documentation I wrote to accompany the program. I hope that the combination of the documentation and the copious comments in the listing will enable you to understand FRMTR fully.

TIPMX FULLY REENTRANT FORMATTING ROUTINE

This routine processes and performs nine (9) different functions at the present time. Eight (8) opcodes are used, 0 through 7. The 4 most significant bits (B0-3) are used to designate the opcodes thus allowing room for further opcodes if needed. The 12 least significant bits (B4-15) are used for instructions in a particular opcode.

OPCODE	FUNCTION
0	1. END FORMATTING ROUTINE IF 12 LSB=0 2. FILL N COLUMNS WITH SPECIFIED CHARACTER.
1	PRINT (OUTPUT) INTEGER.
2	POSITION CURSOR ABSOLUTE (RELATIVE TO HOME POSITION).
3	INPUT (READ) INTEGER.
4	POSITION CURSOR RELATIVE TO LAST POSITION OF CURSOR.
5	INPUT/OUTPUT STRING.
6	1. REPEAT FOLLOWING FORMAT N TIMES. 2. END FORMAT REPEAT
7	LINE FEED CARRIAGE RETURN OR SPECIAL FUNCTION

For ease of use, the formatting instructions have been effectively microcoded, the user does not have to work with bits.

CALLING SEQUENCE OF FRMTR

```
LI      R11,FRMTR
BLWP    R10
DATA    OPLST,FMTLST,LUNS
JMP     ERR
```

NUMERICAL BASE

The routine is initialized to base 10. The base is changed by N or D fields (do not use both in the same formatting command) in format 1 or 3. That base will then be used until it is changed again.

NESTING OF FORMAT LOOPS

Nests up to eight (8) deep (determined by MAX in equate section) are permitted. Opcode six (6) is used to start and end a loop. Each loop MUST have a SEPARATE end repeat format, i.e. there must be as many end repeat opcodes as there are begin repeat opcodes.

DEFINITIONS OF CALLED FORMATTER SUBROUTINES

INCHAR - Reads a single ASCII character into the left-hand (B0-7) byte of R7.

OCHAR - Outputs a single ASCII character from the left-hand (B0-7) byte of R7.

BINBAS - Converts binary numbers to digital ASCII strings in base 2,8, 10, or 16. Determines the sign of the number and stores a plus (+) or minus (-) ASCII character at the start of the string. The subroutine uses the following registers for the following values:

REGISTER	VALUE
R5	WIDTH OF OUTPUT (NUMBER OF DIGITS) INCLUDING SIGN IF ANY
R6	ADDRESS OF BINARY NUMBER
R7	BASE (2,8,10,16)

The maximum width of a number is dictated by the base and the fact that only one word (2 bytes) of binary are considered. If the number is a signed number, the maximum width is reduced by one to allow for the sign.

At the end of the subroutine, the registers will have the following values:

REGISTER	VALUE
R5	WIDTH (SHOULD ALWAYS BE POSITIVE)
R6	ADDRESS OF START OF ASCII STRING
R7	STATUS 0=GOOD 1=ERROR

BASBIN - Converts ASCII digital strings (base 2,8,10, or 16) with or without signs (+ or -) into a binary word (2 bytes). The subroutine uses the following registers for the following values:

REGISTER	VALUE
R5	WIDTH OF STRING INCLUDING SIGN (+ or -) IF ANY
R6	ADDRESS OF START OF ASCII STRING
R7	CONVERSION BASE (2,8,10,16)

The subroutine rejects any width that would result in a binary number too large to be contained in one word (2 bytes).

At the end of the subroutine, the registers have the following values:

REGISTER	VALUE
R5	WIDTH (LESS 1 IF A SIGN WAS IN STRING)
R6	ADDRESS OF BINARY WORD
R7	STATUS 0=GOOD 1=ERROR

CURABS - Outputs a command to position the cursor at Row (line) Y and Column X relative to cursor home (upper left-hand corner of CRT). Cursor home is Row 0, Column 0. The subroutine uses the following registers for the following values:

REGISTER	VALUE
R6	ROW (LINE)
R7	COLUMN

CURREL - Outputs a command to position the cursor to Row (line) Y and Column X relative to the cursors present position. This subroutine uses the cursor read command to determine the present position of the cursor, therefore it can not be used with CRT's not having this command capability. The ADM-3A does not have this capability, the ADM-1 does. The subroutine uses the same registers and their values as the CURABS subroutine.

FORMAT OF LUNS FOR CALLING SEQUENCE

LUNS is used to designate the input and output devices to be used with the formatting routine. It will be consist of two bytes. The first byte will be the input device, the second byte will be the output device.

EXIT FROM FORMATTER WHEN ERROR IS DETECTED

Formatter will return control to the calling program with three data items supplied to the calling routine. RØ will return the error code which will designate the portion of the formatter that detected the error (see following table), R1 will contain the OPLIST address, and R2 will contain the FMTLST address.

RØ VALUE	ERROR AREA
1	INVALID OPCODE
2	REPEAT NESTING ERROR
3	INPUT CHARACTER SYNTAX ERROR
4	REPEAT VALUE MORE THAN MAXIMUM ALLOWED
5	OUTPUT CHARACTER SUBROUTINE
6	INPUT CHARACTER SUBROUTINE
7	CURSOR POSITION OUTPUT ROUTINE
8	CURSOR POSITION INPUT ROUTINE
9	RESERVED (NOT PRESENTLY USED)
10	ESCAPE SYMBOL DETECTED

NEEDED EQUATES FOR FORMATTER MICROCODE

The user will need to include in the calling routine the following equates:

```
     B      EQU    >20
  CRLF      EQU    >7000       OPCODE 7
     D      EQU    >40
ENDFRM      EQU    >0
ENDREP      EQU    >6000
     F      EQU    >800
     I      EQU    >20
 INPUT      EQU    >3000       OPCODE 3
     L      EQU    >80
  MOVE      EQU    >4000       OPCODE 4
     N      EQU    >80
     P      EQU    >800
 PRINT      EQU    >1000       OPCODE 1
     R      EQU    >80
   ROW      EQU    >780
REPEAT      EQU    >6000       OPCODE 6
     S      EQU    >400
   SET      EQU    >2000       OPCODE 2
STRING      EQU    >5000       OPCODE 5
     T      EQU    >400
   TAB      EQU    >0          OPCODE 0
     Y      EQU    >100
     Z      EQU    >800
```

EXAMPLE OF PROGRAM USING FORMATTER

Appendix A contains a test program that uses microcode to test Opcode 0 (TAB), 1 (PRINT), 2 (SET), 3 (INPUT), 4 (MOVE), 5 (STRING), 6 (REPEAT), and 7 (CRLF).

PROVISION FOR GETTING OPLIST, FMTLST, AND LUN ADDRESSES FROM STACK

If any of these pointers are to be gotten from the stack, the calling sequence for the formatter as shown on page 1 would be modified by adding +1 to the pointer that will be obtained from stack.

EXAMPLE: It is desired to obtain OPLST and LUNS from the stack and FMTLST from DSEG.

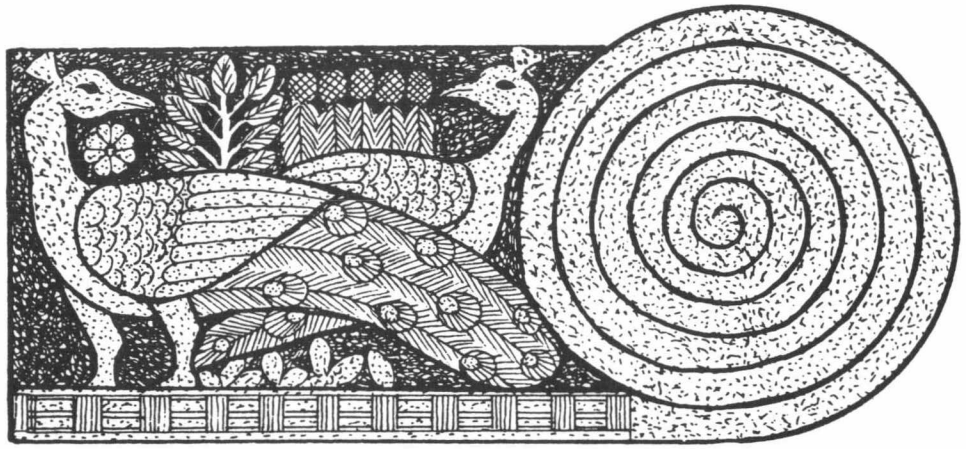
```
LI      R11,FRMTR
BLWP    R10
DATA    OPLST+1,FMTLST,LUNS+1
JMP     ERR
```

IF USER INPUTS TOO MANY DIGITS FOR ONE NUMBER

The formatter examines the length of the input number. If the maximum allowed is exceeded, the first character that exceeds the maximum is examined. If the character is an allowed terminator (i.e., COMMA, CR, SLASH, COLON, or PERIOD), the formatter will take a normal exit and process the next format. If the character is not a terminator, the formatter will wait until an allowed terminator is input. It will not accept any further numbers.

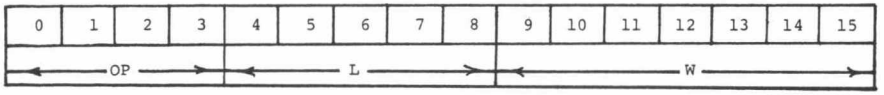
IF STRING LENGTH EXCEEDS FORMAT MAXIMUM

If the maximum is exceeded, the first character that exceeds the maximum is examined. If the character is an allowed terminator (i.e., CR, ESCAPE, or NULL), the formatter will take a normal exit and process the next format. If the character is not a terminator, the formatter will exit and remainder of string will be ignored.



IDENTIFICATION: OPCODE 0 FILL N COLUMNS WITH SPECIFIED CHARACTER

FORMAT:



FIELD	DESCRIPTION
OP	OPCODE 0
L	BINARY REPRESENTATION OF ASCII CHARACTER. THIS NUMBER = DECIMAL VALUE OF CHARACTER MINUS 32. BECAUSE OF FIVE (5) BIT FIELD, THERE ARE ONLY 32 CHARACTERS THAT CAN BE USED. SEE ASCII 20_{16} THROUGH $3F_{16}$.
W	THE NUMBER OF TIMES THE CHARACTER WILL BE OUTPUT. BECAUSE OF THE SEVEN (7) BIT FIELD, THE MAXIMUM NUMBER THAT CAN BE DONE WITH ONE OPCODE 0 INSTRUCTION IS 63 CHARACTERS. THE PLACE THAT A CHARACTER WILL START TO BE PRINTED WILL BE DICTATED BY THE POSITION OF THE CURSOR.

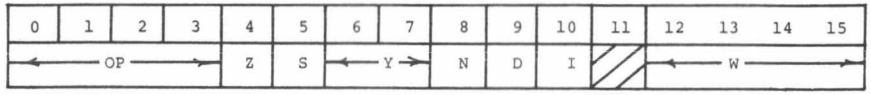
The microcodes for OPCODE 0 are:

MICROCODE	DESCRIPTION
ENDFRM CH*L+W+TAB EXAMPLE: 4*L+14+TAB	END FORMATTING CONVERSION ROUTINE. CH = DECIMAL VALUE OF CHARACTER DESIRED. SEE TABLE ON NEXT PAGE FOR CHARACTERS AND APPROPRIATE DECIMAL NUMBER TO BE ENTERED. L = REPRESENTS L FIELD. IT MUST BE SHOWN. W = DECIMAL VALUE OF TIMES THE CHARACTER WILL BE OUTPUT. IT MUST BE SOME NUMBER AT LEAST EQUAL TO 1 AND NOT MORE THAN 63. TAB - DESIGNATION OF OPCODE 0.

NUMBER TO BE ENTERED IN L FIELD (BASE 10)	CHARACTER REPRESENTED BY NUMBER
0	SPACE
1	!
2	"
3	#
4	$
5	%
6	&
7	'
8	(
9	)
10	*
11	+
12	,
13	-
14	.
15	/
16	0
17	1
18	2
19	3
20	4
21	5
22	6
23	7
24	8
25	9
26	:
27	;
28	<
29	=
30	>
31	?

IDENTIFICATION: OPCODE 1 PRINT INTEGER

FORMAT:



FIELD	DESCRIPTION
OP	OPCODE 1
Z	0 = LEADING ZERO SUPPRESSION. 1 = NO LEADING ZERO SUPPRESSION.
S	0 = UNSIGNED. 1 = SIGNED (SIGN WILL BE FIRST CHARACTER OUTPUT FROM ROUTINE).
Y	0 = BASE 10 1 = BASE 16 2 = BASE 2 3 = BASE 8 THIS CODE WILL NEED TO BE SPECIFIED IF A NEW RADIX IS SPECIFIED BY CODE N OR D.
N	0 = EXAMINE D CODE (NEXT FIELD IN TABLE). 1 = SET RADIX ONLY. IF N IS SELECTED, THE ONLY THING THAT THE OPCODE WILL DO IS TO CHANGE THE RADIX. IT WILL NOT OUTPUT A NUMBER. DO NOT SPECIFY N CODE AND D CODE IN SAME OPCODE WORD.
D	0 = USE PREVIOUS RADIX. 1 = SET RADIX PER CODE Y.
I	0 = ADVANCE TO NEXT OPERAND. 1 = INCREMENT CURRENT OPERAND BY 2.
W	THE MAXIMUM NUMBER OF DIGITS THAT CAN BE OUTPUT. THE MAXIMUM NUMBER ALLOWED IS A FUNCTION OF THE CHOSEN BASE AND THE FACT THE NUMBER IS STORED IN ONE (1) WORD (2 BYTES).

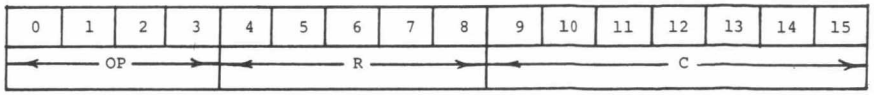
The microcode for OPCODE 1 is:

MICROCODE	DESCRIPTION
BR*Y+Z+S+N+D+I+W+PRINT EXAMPLE: 2*Y+Z+D+8+PRINT	BR = BASE REPRESENTATION. BASE 10 IS THE DEFAULT VALUE IF NOTHING IS SHOWN. IF BASE 10 IS CHOSEN OR IF THE RADIX IS NOT GOING TO BE CHANGED, BR*Y WILL NOT BE USED. THE VALUES TO USE ARE: BR = 1, BASE = 16 BR = 2, BASE = 2 BR = 3, BASE = 8 Y = RADIX MULTIPLIER. USED IN CONJUNCTION WITH BR TO SPECIFY BASE DESIRED. Z = NO LEADING ZERO SUPPRESSION DESIRED. DEFAULT (IF NOT SPECIFIED) IS LEADING ZERO SUPPRESSION. S = SIGNED NUMBER. DEFAULT (IF NOT SPECIFIED) IS UNSIGNED. N = SET RADIX ONLY. IF SPECIFIED, OPCODE WILL ONLY SET RADIX AS DEFINED BY BR*Y. DEFAULT (IF NOT SPECIFIED) IS EXAMINE D CODE. D = SET RADIX AS SPECIFIED BY BR*Y. DEFAULT (IF NOT SPECIFIED) IS USE PREVIOUS RADIX. I = USE SAME FORMAT BUT INCREMENT OPERAND VALUE BY 2. THIS ALLOWS DUMP OF MEMORY WHEN USED WITH OPCODE 6 (REPEAT FORMAT). IF DUMP IS GOING TO BE USED AS AN INNER LOOP WHICH WILL DUMP SAME SECTION OF MEMORY MORE THAN ONCE, STARTING POINT WILL HAVE TO BE REINITIALIZED EACH TIME. W = MAXIMUM NUMBER OF DIGITS THAT WILL BE ALLOWED. THIS WILL BE SOME NUMBER EQUAL TO OR MORE THAN ONE (1) AND NO MORE THAN THE DECIMAL MAXIMUM SHOWN BELOW FOR EACH BASE: BASE 2 — MAXIMUM 16 BASE 8 — MAXIMUM 5 BASE 10 — MAXIMUM 5 BASE 16 — MAXIMUM 4

MICROCODE	DESCRIPTION
BR*Y+Z+S+AHD+W+PRINT	PRINT = DESIGNATION OF OPCODE 1

IDENTIFICATION: OPCODE 2 SET CURSOR TO ABSOLUTE POSITION

FORMAT:



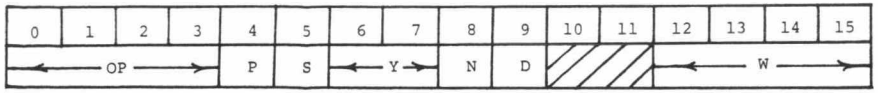
FIELD	DESCRIPTION
OP	OPCODE 2.
R	DECIMAL VALUE FOR THE NUMBER OF ROWS (LINES) WANTED. THE MAXIMUM ALLOWED IS 23, MINIMUM IS 0.
C	DECIMAL VALUE FOR THE NUMBER OF COLUMNS WANTED. THE MAXIMUM ALLOWED IS 79, MINIMUM IS 0.

The microcode for OPCODE 2 is:

MICROCODE	DESCRIPTION
NR*R+C+SET EXAMPLE: 15*R+24+SET	NR = DECIMAL NUMBER OF ROWS (LINES DESIRED. THIS WILL BE SOME VALUE AT LEAST 0 AND NO MORE THAN 23. HOME (TOP) ROW IS 0. R = ROW MULTIPLIER. USED IN CONJUNCTION WITH NR TO SPECIFY NUMBER OF ROWS DESIRED. C = DECIMAL NUMBER OF COLUMNS DESIRED. THIS WILL BE SOME NUMBER AT LEAST 0 AND NOT MORE THAN 79. HOME (LEFT) COLUMN IS 0. SET = DESIGNATION OF OPCODE 2.

IDENTIFICATION: OPCODE 3 INPUT INTEGER

FORMAT:



FIELD	DESCRIPTION
OP	OPCODE 3.
P	0 = LEAVE FIELD BLANK BEFORE INPUT 1 = FILL FIELD WITH * BEFORE INPUT
S	0 = DO NOT ALLOW ROOM FOR SIGN 1 = ALLOW ROOM FOR SIGN
Y	0 = BASE 10 1 = BASE 16 2 = BASE 2 3 = BASE 8 THIS CODE WILL NEED TO BE SPECIFIED IF A NEW RADIX IS SPECIFIED BY CODE N OR D.
N	0 = EXAMINE D CODE (NEXT FIELD IN TABLE). 1 = SET RADIX ONLY. IF N IS SELECTED, THE ONLY THING THAT THE OPCODE WILL DO IS TO CHANGE THE RADIX. IT WILL NOT INPUT A NUMBER. DO NOT SPECIFY N CODE AND D CODE IN THE SAME OPCODE WORD.
D	0 = USE PREVIOUS RADIX. 1 = SET RADIX PER CODE Y.
W	THE MAXIMUM NUMBER OF DIGITS THAT CAN BE INPUT. THE MAXIMUM NUMBER ALLOWED IS A FUNCTION OF THE CHOSEN BASE AND THE FACT THE NUMBER WILL BE STORED IN ONE (1) WORD (2 BYTES).

The microcode for OPCODE 3 is:

MICROCODE	DESCRIPTION
BR*Y+P+S+N+D+W-INPUT EXAMPLE: N+INPUT	BR = BASE REPRESENTATION, BASE 10 IS THE DEFAULT VALUE IF NOTHING IS SHOWN. IF BASE 10 IS CHOSEN OR IF THE RADIX IS NOT GOING TO BE CHANGED, BR*Y WILL NOT BE USED. THE VALUES TO USE ARE: BR — BASE 1 — 16 2 — 2 3 — 8 Y = RADIX MULTIPLIER. USED IN CONJUNCTION WITH BR TO SPFCIFY BASE DESIRED. P = FILL FIELD BEFORE INTEGERS WITH *. DEFAULT VALUE (IF NOT SPECIFIED) IS BLANKS BEFORE INTEGERS. S = SIGNED NUMBER. DEFAULT (IF NOT SPECIFIED) IS UNSIGNED. N = SET RADIX ONLY. IF SPECIFIED, OPCODE WILL ONLY SET RADIX AS DEFINED BY BR*Y. DEFAULT (IF NOT SPECIFIED) IS EXAMINE D CODE. D = SET RADIX AS SPECIFIED BY BR*Y. DEFAULT (IF NOT SPECIFIED) IS USE PREVIOUS RADIX. W = MAXIMUM NUMBER OF DIGITS THAT WILL BE ALLOWED. THIS WILL BE SOME NUMBER EQUAL TO OR MORE THAN ONE (1) AND NO MORE THAN THE DECIMAL MAXIMUM SHOWN BELOW FOR EACH BASE. BASE — MAXIMUM 2 — 16 8 — 5 10 — 5 16 — 4

MICROCODE	DESCRIPTION
BR*Y+P+S+N+D+W+INPUT	INPUT = DESIGNATION OF OPCODE 3.

IDENTIFICATION: OPCODE 4 SET CURSOR TO RELATIVE POSITION

FORMAT:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
← OP →				NR	← R →				NC	← C →					

FIELD	DESCRIPTION
OP	OPCODE 4.
NR	SIGN FOR ROWS. 0 = POSITIVE, 1 = NEGATIVE.
R	DECIMAL VALUE FOR THE NUMBER OF ROWS (LINES) MOVEMENT WANTED. THE MAXIMUM NUMBER OF ROWS THAT CAN BE MOVED IN ONE (1) OPCODE IS 15. 0 ROW IS THE ROW THAT THE CURSOR IS CURRENTLY ON. ENSURE THAT THE TOTAL OF OLD ROW + NEW ROW IS NOT A NEGATIVE NUMBER OR MORE THAN 24.
NC	SIGN FOR COLUMNS. 0 = POSITIVE, 1 = NEGATIVE.
C	DECIMAL VALUE FOR THE NUMBER OF COLUMNS MOVEMENT WANTED. THF MAXIMUM NUMBER OF COLUMNS THAT CAN BE MOVED IN ONE (1) OPCODE IS 63. 0 COLUMNS IS THE COLUMN THAT THE CURSOR IS CURRENTLY ON. ENSURE THAT THE TOTAL OF OLD COLUMN + NEW COLUMN IS NOT A NEGATIVE NUMBER OR MORE THAN 80.

The microcode for OPCODE 4 is:

MICROCODE	DESCRIPTION
NR*R+F+D+C=MOVE	NR = DECIMAL NUMBER NO LARGER THAN 15 ROWS (LINES) MOVEMENT FROM PRESENT CURSOR POSITION DESIRED. NOTE CAUTION IN ABOVE DISCUSSION OF R FIELD.
EXAMPLE:	
5*R+F+12+MOVE	R = ROW MULTIPLIER. USED IN CONJUNCTION WITH NR TO SPECIFY NUMBER OF ROWS MOVEMENT DESIRED.
	F = NEGATIVE ROW INDICATOR. DEFAULT (NOT SPECIFIED IS POSITIVE ROW.

MICROCODE	DESCRIPTION
NR*R+F+D+C+MOVE	C = DECIMAL NUMBER NOT LARGER THAN 63 COLUMNS MOVEMENT FROM PRESENT CURSOR POSITION DESIRED. NOTE CAUTION IN PREVIOUS DISCUSSION OF C FIELD.
	D = NEGATIVE COLUMN INDICATOR. DEFAULT (NOT SPECIFIED) IS POSITIVE COLUMN.
	MOVE = DESIGNATION OF OPCODE 4

IDENTIFICATION: OPCODE 5 STRING I/0

FORMAT:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
← OP →				F	T	← R →			D	← U →					

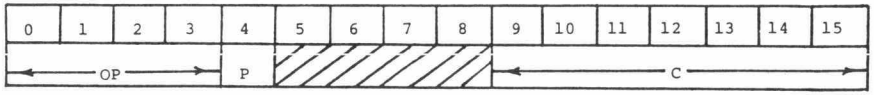
FIELD	DESCRIPTION
OP	OPCODE 5
F	0 = DATA IS FOLLOWING IN FORMAT LIST. THIS CAN ONLY BE USED FOR OUTPUT. 1 = DATA IS IN DATA AREA. FOR OUTPUT, THIS IS OPLIST. FOR INPUT, DATA WILL BE ACCESSED BY INCHAR SUBROUTINE.
T	0 = DATA WILL BE TERMINATED BY NULL CHARACTER FOR OUTPUT OR CR FOR INPUT. 1 = STRING LENGTH WILL BE SPECIFIED BY U FIELD (SEE DESCRIPTION BELOW). THIS IS USED FOR OUTPUT ONLY.
R	0 = NO STRING LENGTH MULTIPLIER. 1-7 = VALUE OF MULTIPLIER TO BE USED.
D	0 = OUTPUT STRING 1 = INPUT STRING. T FIELD CANNOT BE SPECIFIED IF THE D FIELD IS SPECIFIED.
U	DECIMAL VALUE FOR LENGTH OF STRING. IT MUST BE AT LEAST 1 AND NO MORE THAN 63.

The microcode for OPCODE 5 is:

MICROCODE	DESCRIPTION
NR*R+F+T+D+SL+STRING F+T+D+SL+STRING EXAMPLE: F+T+27+STRING	NR = DECIMAL NUMBER NO LARGER THAN 7. IT INDICATES WHAT NUMBER THE STRING LENGTH (SL) IS GOING TO BE MULTIPLIED BY. THIS NUMBER IS VALUE DESIRED MINUS 1. R = STRING LENGTH MULTIPLIER. USED IN CONJUNCTION WITH NR AND SL TO SPECIFY MAXIMUM STRING LENGTH. CAN BE USED ONLY WITH OUTPUT STRING AND CAN SPECIFY MAXIMUM OF 504 CHARACTERS. F = DATA IS DATA AREA. DEFAULT (IF NOT SPECIFIED) IS DATA FOLLOWS IN FORMAT LIST. T = SL GIVES DECIMAL VALUE FOR OUTPUT STRING LENGTH. DEFAULT (IF NOT SPECIFIED) IS NULL TERMINATES STRING. D = SL GIVES DECIMAL VALUE FOR INPUT STRING LENGTH. D CODE AND T CODE CANNOT BOTH BE SPECIFIED IN THE SAME OPCODE. DEFAULT (IF NOT SPECIFIED) IS OUTPUT STRING. SL = STRING LENGTH (ASCII CHARACTERS). MUST BE SOME VALUE AT LEAST ONE (1) AND NOT MORE THAN 63 IF D CODE OR T CODE IS SPECIFIED. IF NEITHER D CODE OR T CODE IS SPECIFIED, SL WILL BE 0. STRING = DESIGNATION FOR OPCODE 5

IDENTIFICATION: OPCODE 6 REPEAT FORMAT

FORMAT:



FIELD	DESCRIPTION
OP	OPCODE 6
P	0 = END REPEAT FORMAT 1 = REPEAT NEXT FORMAT(S) C FIELD NUMBER OF TIMES
C	THE DECIMAL NUMBER OF TIMES THE FORMAT IS TO BE REPEATED. THE UNSIGNED NUMBER MUST BE AT LEAST ONE (1) AND NOT MORE THAN 127 IF P CODE IS SPECIFIED. IF P CODE IS NOT SPECIFIED, C CANNOT BE SPECIFIED EITHER, THIS CONSTITUTES AN END REPEAT FORMAT.

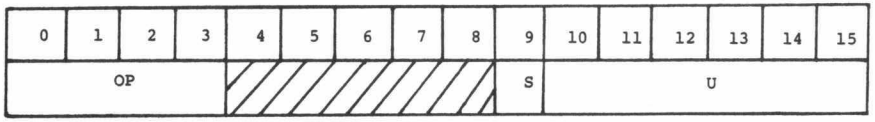
The Microcode for OPCODE 6 is:

MICROCODE	DESCRIPTION
ENDREP	THIS IS AN END REPEAT FORMAT OPCODE. THIS MUST MATCH EACH REPEAT FORMAT OPCODE ON A ONE TO ONE BASIS. THE SERIES OF FORMATS BETWEEN THE REPEAT FORMAT AND END REPEAT FORMAT OPCODES CONSTITUTES A LOOP.
P+NR+REPEAT EXAMPLE: P+36+REPEAT	P = REPEAT THE FOLLOWING FORMAT(S) NR DECIMAL NUMBER OF TIMES. THE REPEAT FUNCTION IS PERFORMED ON THE FORMATS FOLLOWING THIS OPCODE AND THE NEXT END REPEAT FORMAT OPCODE.

MICROCODE	DESCRIPTION
P+NR+REPEAT	NR = DECIMAL NUMBER OF TIMES FORMAT IS TO BE REPEATED. THE NUMBER WILL BE AT LEAST ONE (1) AND NOT MORE THAN 127. REPEAT = DESIGNATION FOR OPCODE 6

IDENTIFICATION: OPCODE 7 CARRIAGE RETURN LINE FEED

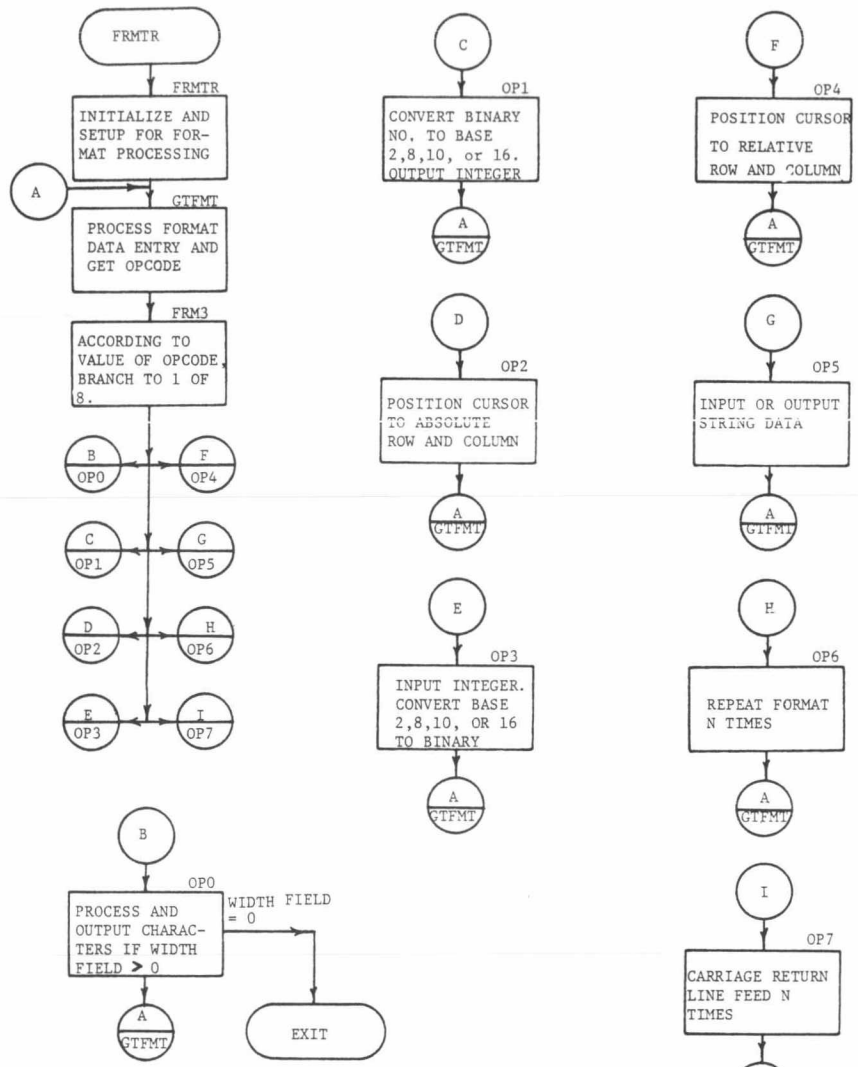
FORMAT:



FIELD	DESCRIPTION
OP	OPCODE 7
S	SPECIAL FUNCTION 0=CRLF, U SPECIFIES NO. OF TIMES. 1=SPECIAL FUNCTION, U SPECIFIES CODE.
U	IF CRLF IS TO BE DONE, NUMBER SHOWN WILL BE ONE LESS THAN NUMBER DESIRED. MAXIMUM THAT CAN BE DONE IN ONE INSTRUCTION IS 64; otherwise it is one of the following codes: 0=ERASE 1=PROTECT 2=UNPROTECT

The microcode for OPCODE 7 is:

MICROCODE	DESCRIPTION
U + CRLF EXAMPLE: 3 + CRLF	U = NUMBER OF TIMES CRLF IS TO BE DONE LESS ONE. IF NO U IS SPECIFIED, 1 CRLF WILL BE DONE. IF 1 U IS SPECIFIED, 2 CRLF WILL BE DONE, ETC. CRLF = DESIGNATION OF OPCODE 7



TOP LEVEL FLOW FRMTR

APPENDIX A

EXAMPLE PROGRAM CALLING FORMATTER

```
TESTER          TXMIRA

0001                        IDT  'TESTER'
0002                        DEF  IOER$
0003                        DEF  PSCL$$
0004                        REF  ENT$1
0005                        REF  FRMTR
0006                        REF  INITDB
0007                        REF  IOIN$
0008                  *                          THE FOLLOWING EQUATES MUST BE
0009                  *                          BE PUT IN ANY PROGRAM CALLING
0010                  *                          THE FORMATTER IF THE CALLING
0011                  *                          PROGRAM IS TO USE THE OPCODE
0012                  *                          MICROCODE AS SHOWN IN THE
0013                  *                          FORMATTER DOCUMENTATION.
0014                  *
0015                  *
0016                  *
0017        0020      B         EQU  >20
0018        7000      CRLF      EQU  >7000
0019        0040      D         EQU  >40
0020        0000      ENDFRM    EQU  0
0021        6000      ENDRPT    EQU  >6000
0022        0800      F         EQU  >800
0023        3000      INPUT     EQU  >3000              OPCODE 3
0024        0080      L         EQU  >80
0025        4000      MOVE      EQU  >4000              OPCODE 4
0026        0080      N         EQU  >80
0027        0800      P         EQU  >800
0028        1000      PRINT     EQU  >1000              OPCODE 1
0029        0080      R         EQU  >80
0030        6000      REPEAT    EQU  >6000              OPCODE 6
0031        0400      S         EQU  >400
0032        2000      SET       EQU  >2000              OPCODE 2
0033        5000      STRING    EQU  >5000              OPCODE 5
0034        0400      T         EQU  >400
0035        0000      TAB       EQU  >0                 OPCODE 0
0036        0100      Y         EQU  >100
0037        0800      Z         EQU  >800
0038        0000'     PSCL$$    EQU  $                  START FORMATTER TEST ROUTINE
0039  0000  0201                LI   R1,INITDB
      0002  0000
0040  0004  1301                JEQ  NODB               IS DEBUG WANTED?
0041  0006  0691                BL   *R1                 YES, START DEBUG
0042  0008  06A0      NODB      BL   @ENT$1             NO,DON'T DO DEBUG
      000A  0000
0043  000C  0028                DATA 40
0044  000E  06A0                BL   @IOIN$             INITIALIZE I/O SYSTEM
      0010  0000
0045  0012  020B                LI   R11,FRMTR
      0014  0000
0046  0016  040A                BLWP R10
0047  0018  006C"               DATA OPLST,FMTLST,LUNS
      001A  0000"
      001C  0098"
0048  001E  1001                JMP  ERR
0049  0020  045B      IOER$     RT
0050        0022'     ERR       EQU  $                  DUMMY ARGUMENT FOR NOW
0051  0022                      PEND
0052  0000                      DSEG
0053  0000  5C1C      FMTLST    DATA F+T+28+STRING        OUTPUT MAX 30 CHARACTERS
0054  0002  2101                DATA 2*R+1+SET          ABSOLUTE POSITION CURSOR TO
0055                  *                                 ROW 2, COLUMN 1.
0056  0004  583F                DATA F+63+STRING          OUTPUT MAX 63 CHARACTERS TER
0057                  *                                 BY NULL
0058  0006  228A                DATA 5*R+10+SET         ABSOLUTE POSITION CURSOR TO
0059                  *                                 ROW 5, COLUMN 10.
0060  0008  7000                DATA CRLF               CARRIAGE RETURN LINE FEED
0061  000A  540E                DATA T+14+STRING
0062  000C    4E                TEXT 'NOW WE GOT IT!'
0063  001A  4185                DATA 3*R+5+MOVE         RELATIVE POSITION CURSOR TO
0064                  *                                 ROW +3, COLUMN +5.
0065  001C  5445                DATA D+T+5+STRING         INPUT MAX 5 CHARACTERS TERM
```

```
0066                    *                                BY CARRIAGE RETURN (CR)
0067 001E 2681          DATA 13*R+1+SET                  ABSOLUTE POSITION CURSOR TO
0068                    *                                ROW 13, COLUMN 1.
0069 0020 0199          DATA 3*L+25+TAB                   PRINT 25 #
0070 0022 058A          DATA 11*L+10+TAB                  PRINT 10 +
0071 0024 0085          DATA L+5+TAB                      PRINT 5 !
0072 0026 2701          DATA 14*R+1+SET                  ABSOLUTE POSITION CURSOR TO
0073                    *                                ROW 14, COLUMN 1.
0074 0028 001E          DATA 30+TAB                       PRINT 30 SPACES
0075 002A 1944          DATA Y+Z+D+4+PRINT               OUTPUT HEX, NO LEADING 0 SUPP,
0076                    *                                  SET RADIX, UNSIGNED, MAX 4 DIG
0077 002C 1804          DATA Z+4+PRINT                     USE PREVIOUS RADIX, NO LEADING
0078                    *                                  0 SUPP, PRINT AND EXAMINE D
0079                    *                                  CODE, MAX 4 DIGITS
0080 002E 40DD          DATA 2*R-35+MOVE                 RELATIVE CURSOR POSITION TO
0081                    *                                ROW +1, COLUMN -35.
0082 0030 6810          DATA P+16+REPEAT                 REPEAT NEXT FORMAT 16 TIMES
0083 0032 1004          DATA 4+PRINT                       LEADING 0 SUPP, USE PREVIOUS
0084                    *                                  RADIX, UNSIGNED, MAX 4 DIGITS
0085 0034 6000          DATA ENDRPT                        END REPEAT FORMAT
0086 0036 2901          DATA 18*R+1+SET                  ABSOLUTE CURSOR POSITION TO
0087                    *                                ROW 18,COLUMN 1.
0088 0038 5C2A          DATA T+F+42+STRING                 OUTPUT STRING FROM OPLIST.
0089 003A 3045          DATA D+5+INPUT                   INPUT UP TO 5 NUMBERS FROM CRT.
0090 003C 2981          DATA 19*R+1+SET                  ABSOLUTE POSITION CURSOR TO
0091                    *                                ROW 19, COLUMN 1.
0092 003E 542A          DATA T+42+STRING                 OUTPUT FINAL MESSAGE.
0093 0040   49          TEXT 'I LOVE IT, WE HAVE GOT THIS THING WHIPPED!'
0094 006A 0000          DATA ENDFRM                      END FORMAT CONVERSION
0095 006C 009C" OPLST   DATA STRNG1                       FIRST OUTPUT MESSAGE
0096 006E 00B8"         DATA STRNG2                       SECOND OUTPUT MESSAGE
0097 0070 0124"         DATA STRNG3                      FIRST INPUT MESSAGE
0098 0072 0120"         DATA TABLE+32
0099 0074 0122"         DATA TABLE+34
0100 0076 0100"         DATA TABLE
0101 0078 0102"         DATA TABLE+2
0102 007A 0104"         DATA TABLE+4
0103 007C 0106"         DATA TABLE+6
0104 007E 0108"         DATA TABLE+8
0105 0080 010A"         DATA TABLE+10
0106 0082 010C"         DATA TABLE+12
0107 0084 010E"         DATA TABLE+14
0108 0086 0110"         DATA TABLE+16
0109 0088 0112"         DATA TABLE+18
0110 008A 0114"         DATA TABLE+20
0111 008C 0116"         DATA TABLE+22
0112 008E 0118"         DATA TABLE+24
0113 0090 011A"         DATA TABLE+26
0114 0092 011C"         DATA TABLE+28
0115 0094 011E"         DATA TABLE+30
0116 0096 00D6"         DATA STRNG4
0117 0098   01  LUNS    BYTE 1,1
     0099   01
0118 009A   01          BYTE 1,1
     009B   01
0119 009C   4E  STRNG1  TEXT 'NOW IS THE TIME FOR ALL GOOD'
0120 00B8   41  STRNG2  TEXT 'ALL GOOD MEN SIT IN SOFT SAND'
0121 00D5   00          BYTE 0
0122 00D6   48  STRNG4  TEXT 'HOW MANY TIMES DID YOU MAKE IT LAST NIGHT?'
0123 0100 1000  TABLE   DATA >1000
0124 0102 1001          DATA >1001
0125 0104 1002          DATA >1002
0126 0106 1003          DATA >1003
0127 0108 1004          DATA >1004
0128 010A 1005          DATA >1005
0129 010C 1006          DATA >1006
0130 010E 1007          DATA >1007
0131 0110 1008          DATA >1008
0132 0112 1009          DATA >1009
0133 0114 100A          DATA >100A
0134 0116 100B          DATA >100B
0135 0118 100C          DATA >100C
0136 011A 100D          DATA >100D
0137 011C 100E          DATA >100E
0138 011E 100F          DATA >100F
0139 0120 23CF          DATA >23CF
0140 0122 03CF          DATA >03CF
0141 0124       STRNG3  BSS  62
0142 0162               DEND
0143 0022       WSP     BSS  1000
0144                    END
```

```
   B        0020      CRLF     7000      D        0040      ENDFRM   0000
   ENDRPT   6000    E ENT$1    000A    ' ERR      0022      F        0800
 " FMTLST   0000    E FRMTR    0014    E INITDB   0002      INPUT    3000
 D IOER$    0020    E IOIN$    0010      L        0080    " LUNS     0098
   MOVE     4000      N        0080    ' NODB     0008    " OPLST    006C
   P        0800      PRINT    1000    D PSCL$$   0000      R        0080
   R0       0000      R1       0001      R10      000A      R11      000B
   R12      000C      R13      000D      R14      000E      R15      000F
   R2       0002      R3       0003      R4       0004      R5       0005
   R6       0006      R7       0007      R8       0008      R9       0009
   REPEAT   6000      S        0400      SET      2000      STRING   5000
 " STRNG1   009C    " STRNG2   00B8    " STRNG3   0124    " STRNG4   00D6
   T        0400      TAB      0000    " TABLE    0100    ' WSP      0022
   Y        0100      Z        0800

0000 ERRORS


TXXREF

 B        0017
 CRLF     0018      0060
 D        0019      0065  0075  0089
 ENDFRM   0020      0094
 ENDRPT   0021      0085
 ENT$1              0004  0042
 ERR      0050      0048
 F        0022      0053  0056  0088
 FMTLST   0053      0047
 FRMTR              0005  0045
 INITDB             0006  0039
 INPUT    0023      0089
 IOER$    0049      0002
 IOIN$              0007  0044
 L        0024      0069  0070  0071
 LUNS     0117      0047
 MOVE     0025      0063  0080
 N        0026
 NODB     0042      0040
 OPLST    0095      0047
 P        0027      0082
 PRINT    0028      0075  0077  0083
 PSCL$$   0038      0003
 R        0029      0054  0058  0063  0067  0072  0080  0086  0090
 R1                 0039  0041
 R10                0046
 R11                0045
 REPEAT   0030      0082
 S        0031
 SET      0032      0054  0058  0067  0072  0086  0090
 STRING   0033      0053  0056  0061  0065  0088  0092
 STRNG1   0119      0095
 STRNG2   0120      0096
 STRNG3   0141      0097
 STRNG4   0122      0116
 T        0034      0053  0061  0065  0088  0092
 TAB      0035      0069  0070  0071  0074
 TABLE    0123      0098  0099  0100  0101  0102  0103  0104  0105  0106
                    0107  0108  0109  0110  0111  0112  0113  0114  0115
 WSP      0143
 Y        0036      0075
 Z        0037      0075  0077

THERE ARE 0041 SYMBOLS
```

North Star Disk STOIC

Our November-December issue was barely in the mail when I was tracked down on the telephone by a fellow named Bob Fleming. "I've got some bad news for you," he said. "That STOIC program you've just published looks exactly like the original STOIC program written by Jon Sachs a couple of years ago in a joint Harvard-MIT engineering group."

I obtained a copy of Sachs' program and quickly ascertained that Fleming was right: it was almost word-for-word the same program. I called John Sachs, now at Data General in Massachusetts. He told me that he developed the program in 1978 at the Harvard-MIT Health Science and Technology Division's Biomedical Engineering Center. He said the program was available for anyone to use as long as credit for it remained with the Center.

B. W. Lee, the author of the piece which appeared in DDJ, *said, by way of explanation, that the later installments would have been quite different than the first. That may, or may not, be true, but we still felt, in all fairness, that Mr. Lee should have attributed credit to Harvard-MIT and not signed his name to work which he did not create. The fact that Mr. Lee was charging a fairly significant amount of money to distribute STOIC on tape has just about killed our interest in publishing anything in the future for which the reader will have to pay more than copying and mailing costs.*

We have withdrawn the future installments of Mr. Lee's STOIC article. Instead, we are including information here on ordering STOIC on tape direct from the Biomedical Engineering Center.

We apologize to any reader who has been inconvenienced by this unfortunate situation.

Procedure for Obtaining Copies of STOIC

An industry-standard magnetic tape (9-track, 800 BPI) containing the STOIC program listings along with written descriptions will be supplied upon request. An index of 8080 STOIC programs and the description of this magnetic tape follow:

STOIC TAPE DIRECTORY

PROG
NUMBER DESCRIPTION

0 STOIC BOOTSTRAP
1 STOIC KERNEL
2 BASIC STOIC DEFINITIONS
3 FILE SYSTEM
4 TEXT EDITOR
5 FLOATING POINT PACKAGE (4 ½ PLACE ACCURACY)
6 MISCELLANEOUS DEFINITIONS
7 FAST FOURIER TRANSFORM (FLOATING POINT)
8 LINE DRAWER AND GRAPHICS PACKAGE (DEVICE DEPENDENT)
9 INTERRUPT HANDLER (SYSTEM DEPENDENT)
10 FIXED POINT SIN.COS
11 RADIX EXCHANGE SORT
12 DOUBLE PRECISION ARITHMETIC PACKAGE

13 STOIC WRITE-UP PART 1
14 STOIC WRITE-UP PART 2
15 STOIC ASSEMBLER WRITE-UP
16 STOIC KERNEL WRITE-UP
17 STOIC FILE SYSTEM WRITE-UP
18 STOIC EDITOR WRITE-UP
19 STOIC FLOATING POINT PACKAGE WRITE-UP
20 STOIC INTERRUPT HANDLER WRITE-UP
21 STOIC BOOTSTRAP WRITE-UP
22 STOIC INTIAL LOAD INSTRUCTIONS

TAPE FORMAT

NO LABEL
9 TRACK, ODD PARITY, 800 BPI
FIXED SIZE BLOCKS: 2640 BYTES (DECIMAL)
EACH BLOCK IS 20 132 CHARACTER LINE IMAGES (ASCII)
LINES ARE BLANK FILLED WITH NO CARRIAGE RETURNS
TABS ARE NOT CONVERTED TO SPACES
LINES CONSISTING OF A FORM FEED FOLLOWED BY 131 BLANKS SEPARATE PAGES
INDIVIDUAL PROGRAMS ARE SEPARATED BY A LINE CONTAINING THE STRING "***EOF***".
ALL PROGRAMS ARE IN A SINGLE FILE
TWO COPIES OF THE FILE ARE INCLUDED ON THE TAPE. SEPARATED BY HARDWARE EOF AND TERMINATED BY TWO EOF'S.

PROCEDURE:

1) MAKE LISTINGS OF ALL PROGRAMS AND WRITE-UPS
2) READ ALL WRITE-UPS
3) FOLLOW THE DIRECTIONS IN THE INITIAL LOAD INSTRUCTIONS TO MODIFY PROGRAMS FOR YOUR SYSTEM

THE TAPE CONSISTS OF APPROXIMATELY 9000 (DECIMAL) LINES OF TEXT (PER FILE).

WRITE-UPS

STOIC.WU	DESCRIPTION OF HOW TO GET STOIC RUNNING ON A NEW MACHINE
STOIC1	STOIC WRITE-UP PART 1
STOIC2	STOIC WRITE-UP PART 2
ED.WU	TEXT EDITOR WRITE-UP
BOOT.WU	BOOTSTRAP WRITE-UP
PAPER	MANUSCRIPT OF STOIC PAPER
MODS	LIST OF ALL SOFTWARE REVISIONS
FP.WU	FLOATING POINT PACKAGE WRITE-UP
FS.WU	FILE SYSTEM WRITE-UP
RCV.WU	RCV WRITE-UP
LETTER1	DISTRIBUTION LETTER
COMPARISON	8080 SIGNED AND UNSIGNED COMPARISONS
KERNEL	KERNEL WRITE-UP
PROM.WU	PROM BURNING PROGRAM WRITE-UP
INTERRUPT.WU	INTERRUPT HANDLER WRITE-UP
CODE	ASSEMBLER WRITE-UP
SYSGEN.WU	INSTRUCTIONS FOR GENERATING A NEW BOOTSTRAP FILE AND RUNNING THE COMPRESSOR

Requests should include a brief description of the anticipated use of the STOIC programming system and should include a blank tape and self-addressed, stamped mailing container for its return. Address requests to:

Stephen K. Burns, Technical Director
Biomedical Engineering Center
for Clinical Instrumentation
MIT; Room 20-A 119
Cambridge, Mass. 02139

FEATURE ENHANCEMENTS for ADM-3A

By John Wakerly
Dept. of Electrical Engineering
Computer Systems Lab
Stanford University
Stanford, CA 94305

© 1979, by John F. Wakerly

A couple of months ago I picked up a maintenance manual for my Lear Siegler ADM-3A terminal. Among the SSI and silliness in the hand-drawn schematics, I discovered some nice ways to add some very useful features.

Automatic Repeat

The most useful addition is automatic repeat for every key: hold down the key long enough, and it starts repeating. Figure 1 shows the schematic for the modification. It requires one extra chip, a CD4093 quad CMOS Schmitt trigger NAND, to be added to the board. When I modified my terminal, I found it convenient to glue this chip in place upside-down on top of the 74LS00 at location K16 of the ADM-3A circuit board. The modification also uses a spare 74LS04 inverter already present in the terminal; the existing trace going to pin 1 of this chip must be cut as shown in Figure 2. By the way, if you can't identify pin 1 of an IC, then you shouldn't be doing these modifications.

The few additional components and wires may be soldered in place between the CD4093, the 74LS04, and nearby locations. Ground may be picked up from the foil that runs along the periphery of the circuit board or from pin 7 of the 74LS04; +5 is available at pin 14 of the 74LS04. Be sure to install the diode in the correct direction; the band on the diode corresponds to the straight line in the diode symbol. Also, the 4.7 capacitor should be a tantalum or electrolytic, and the end marked "+" should be connected to the diode-resistor junction as shown. The circuit may be tested off-line with the terminal in half-duplex mode (set the HDX/FDX switch to HDX).

Operation of the circuit is straightforward. The existing signal $\overline{\text{KEYDATA}}$ is low whenever any key is depressed, high otherwise. This signal is buffered through a spare 74LS04 inverter at location L16. The 74LS04 drives an RC network through a diode; any switching diode may be used. The voltage across the capacitor is sensed by the CD4093 Schmitt-trigger NAND, whose output is tied in parallel with the ADM-3A repeat key.

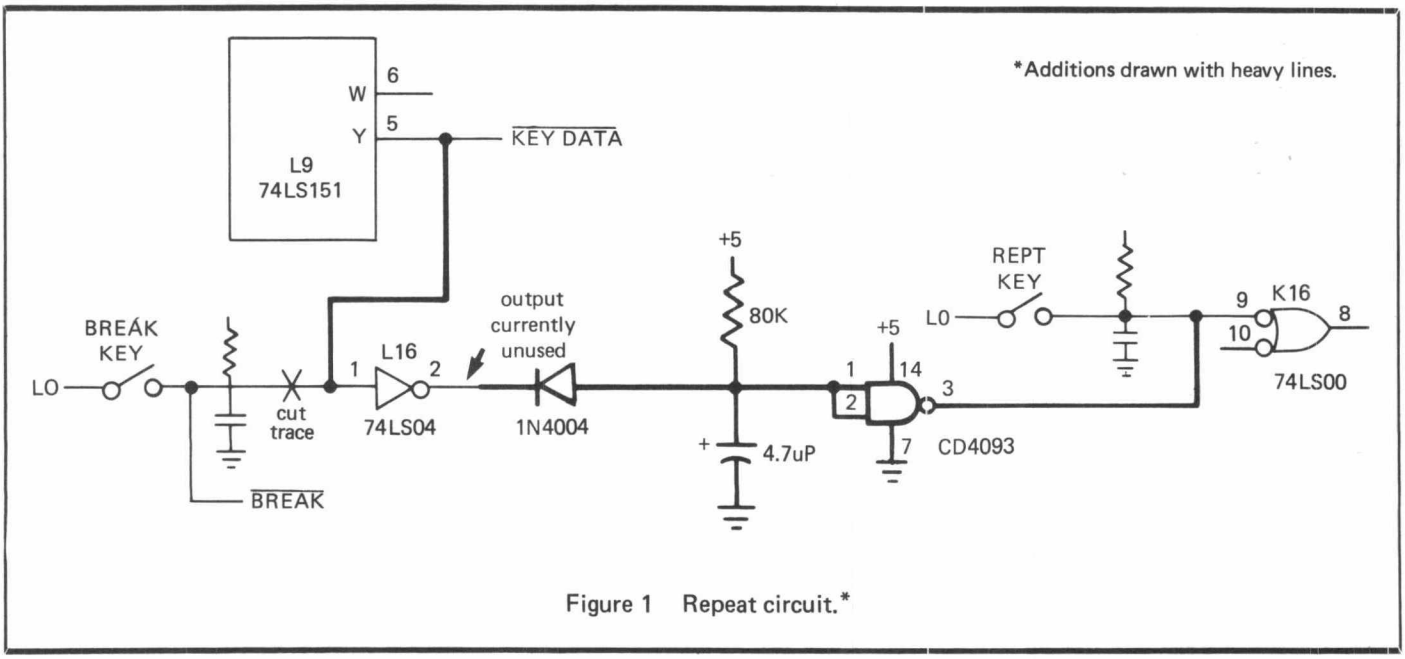


Figure 1 Repeat circuit.*

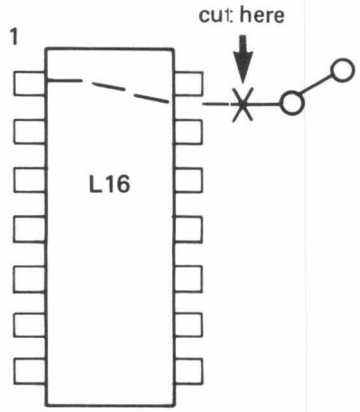


Figure 2 Trace cut at L16.

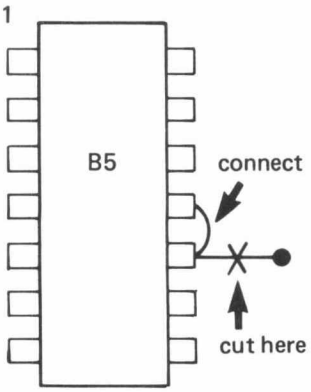


Figure 3 Trace cut at B5.

Normally, $\overline{\text{KEYDATA}}$ is high, the 74LS04 output is low, holding the capacitor discharged through the diode, and the CD4093 output is high. When any key is depressed, $\overline{\text{KEYDATA}}$ goes low, raising the output of the 74LS04 high, which allows the capacitor to charge through the resistor. If the key is held down long enough, the input switching threshold of the CD4093 will be crossed and its output will go low, mimicking the effect of holding down the repeat key. When the key is released, the capacitor immediately discharges through the diode and the 74LS04 low output, causing the repeat function to be released.

Some notes on this circuit are in order for serious logic designers. You may ask, why not get rid of the 74LS04 and just use the W output of the 74LS151, which is the complement of $\overline{\text{KEYDATA}}$? The problem is that Y is obtained internally in the 74LS151 by inverting W. If we used W, we would glitch $\overline{\text{KEYDATA}}$ for the rest of the terminal because of the nonstandard voltages that would be developed every time we discharged the capacitor through W. Another question: why not use another section of the CD4093 instead of the spare 74LS04? Answer: the CD4093 doesn't have enough low-state drive to quickly discharge the capacitor. Finally, why not use another spare 74LS04 instead of the CD4093? Answer: CMOS is required for its high input impedance (5M ohms or more) to allow the 80K-ohm resistor to determine the effective time constant; LSTTL input impedance is only about 20K ohms. A Schmitt trigger is required because of the slow input signal rise time.

The length of time a key must be held down before it starts to repeat is determined by the RC time constant. As a relatively fast typist, I found an 80K resistor and 4.7 uf capacitor to give an appropriate delay, about 400 milliseconds. More heavy-handed typists will want to increase these values to prevent unwanted key repetitions; doubling either component value doubles the delay.

Caps Lock

The next modification is to eliminate a useless feature and replace it with a useful one. If you really have found a use for the keyboard-lock feature of the ADM-3A, read no further. Otherwise, eliminate it by cutting the trace coming into the 74LS11 at location B5, pin 10, and connecting pins 10 and 11 of this IC together, as shown in Figure 3.

Now the $\overline{\text{KBLOCK}}$ signal can be used to enable and disable lower case characters. As shown in Figure 4, simply connect a wire from pin 9 of the 74LS113 at location B10 to pin 13 of the 74LS10 at location L10. This puts $\overline{\text{KBLOCK}}$ in parallel with the LC EN switch on the front panel. Then set the LC EN switch to the UC position, and the internal DISABLE/KBLOCK switch to the $\overline{\text{KBLOCK}}$ position. Now when you turn on the terminal, it will come up in uppercase only. Whenever the terminal receives a control-O, lower case will be enabled; receiving a control-N will return it to uppercase only. If you connect to pin 8 instead of 9 of the 74LS113, then the functions of control-N and control-O are reversed and the terminal will power-up to lowercase only. Setting the LC EN switch to LC EN unconditionally selects lower case. As before, these functions may be tested off-line with the terminal in half-duplex mode.

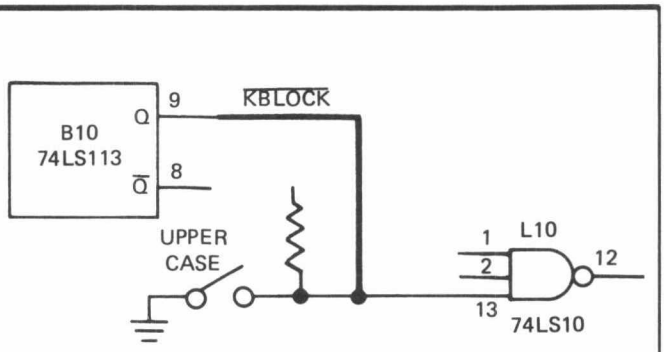


Figure 4 Caps lock circuit.

If your computer doesn't echo control characters when they are typed from the keyboard, you may want to provide some special system commands to enable and disable lower case. For example, I modified my computer's CRT I/O driver so that whenever it receives a NULL from the terminal (control-@ or control-space on the ADM-3A), it toggles an internal upper/lower mode flag and sends the appropriate control code to the terminal to put it in that mode. Furthermore, I automatically select uppercase-only when exiting to my ROM monitor, and enable lower case when entering the disk operating system.

31 Extra Flavors of Printing Characters

If you have an ADM-3A with the lower case option, you may notice that sometimes at power-up there are strange characters on the screen you can never generate yourself. Well, it turns out that the optional lower case ROM for the terminal has 64 characters in it, of which 32 are the normal lower-case character set. The other 32 characters correspond to the 32 ASCII control codes, 00 through 1F hex; 31 of these are printing characters, including some underlined letters and numbers, various arrows, a bell, and strange symbols. The terminal never stores these codes in the refresh memory when they are received from the computer; it simply performs the action, if any, required by the control code (clear screen, move cursor, etc.). Thus, the only time you see these characters is when they somehow get into the refresh memory at power up. What we need, then, is a way for the computer to tell the terminal to go ahead and store these codes.

As wired at the factory, the ADM-3A looks only at the low-order 7 bits of each received character, even if an 8-bit character is received. We can make use of the eighth bit to make the extra characters available. If the eighth bit is a 0, we treat a character between 00 and 1F hex as a normal control character; if it is a 1, we ignore the control function and store the character in the refresh memory instead. All this takes is one spare AND gate at location H5. As shown in Figure 5, cut the output trace from pin 10 of the 74LS02 at H3 and input trace from pin 10 of the 74LS25 at H1. Then install four new wires to make the circuit shown in Figure 6.

Continued on p. 25

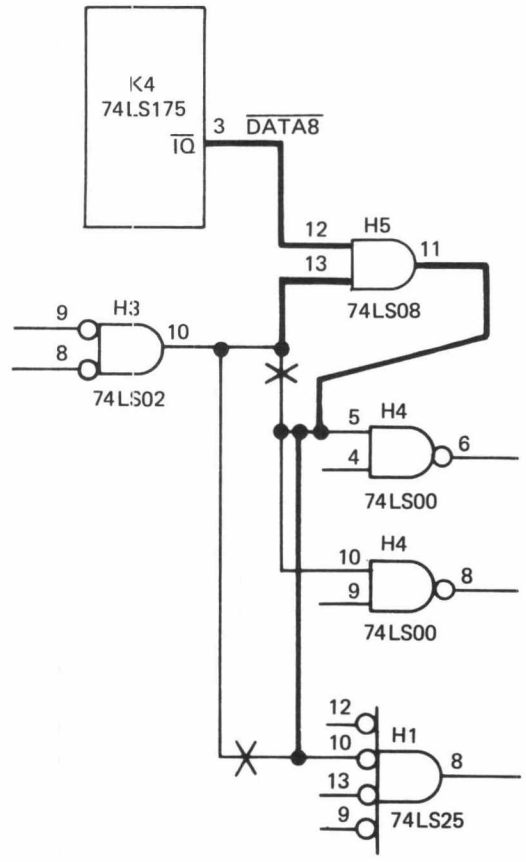


Figure 6 Extra-character circuit.

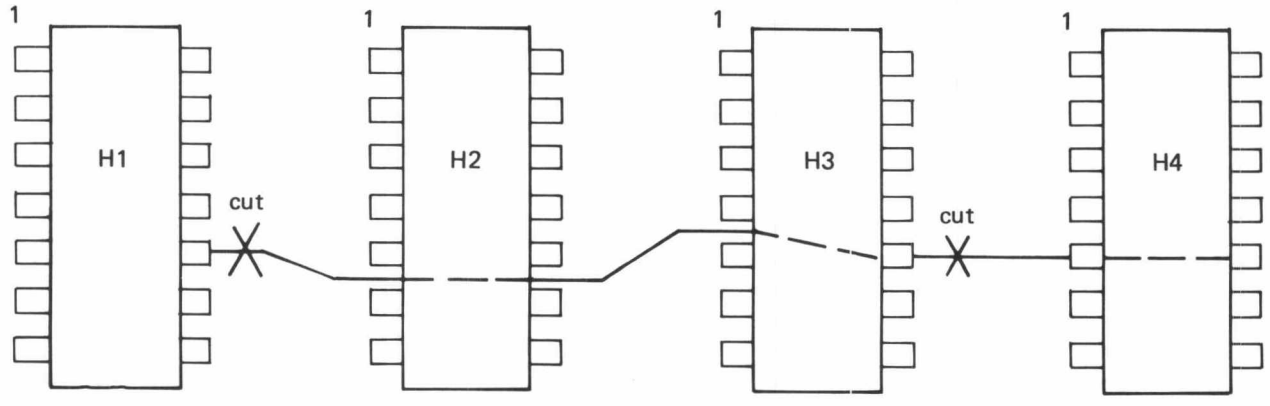


Figure 5 Trace cuts at H1 and H3.

A Table-Driven Assembler On CP/M

BY DARREL J. VAN BUER
1522 Brockton Avenue #5
Los Angeles, CA 90025

Darrel J. Van Buer is studying for a Ph.D. in CS at UCLA. He is a senior computer programmer analyst at SDC.

This article describes a table driven assembler I recently constructed to help with my investigation of the new one chip microcomputers. At present, my interest is in the Intel 8748 computer and the related Intel 8741 Universal Peripheral Interface, both with user programmable EPROMs. With the continued proliferation of new microprocessors, especially low end processors too small for resident utilities, I felt the best approach was a table driven cross-assembler. Since my main system is an 8085-based system running CP/M, I had a large choice of source languages for the assembler. I used ML-80 for several reasons, including availability, efficiency and structured programming features. ML-80 is a structured assembler/compiler written in 1975 to combine the advantages of assembler level machine control with a good set of control structures to produce readable and efficient code. I wrote it as a table driven compiler to make it easier to modify for use with other 8-bit machines. In particular, all components of an operation code except immediate operands are determined by table lookup alone. The target machines for the assembler listed here are the Intel 8748/8048/8035 and 8741/8041.

General Features of the Assembler

The assembler provides as basic features for all versions line parsing, symbol table management and expression evaluation. The line parser defines a source line as a series of fields separated by blanks and tabs and terminated by CR6LF. The first field is an optional label starting in column one, not over 6 characters long, optionally followed by a colon. The second field is an operation code not over 5 characters. The third field is an optional list of operands separated by commas. The fourth field treats the rest of the line as comments. Note that, for some operations, some of the optional fields may actually be required. To allow for longer labels and operations requires changing the declarations for OP, LABELL, LABELTAB and adjusting the loop counters in PARSELINE and GETNAMEVAL. The expression evaluator recognizes labels, numbers and '$' for the current value of the location counter (i.e., the address of the instruction containing $). Numbers may be in decimal, hexadecimal, octal or binary, denoted by a suffix of D, H, O, or B respectively. A number with no suffix is decimal. The first character of numbers must always be digits to distinguish them from labels. The expression evaluator also recognizes the operation symbols '+', '–', '>' and '<' for plus, minus, low byte and high byte respectively. When used as unary operations, the first operand is zero. Low byte is the less significant byte of the following value and high byte is the more significant byte. For example, >01234H is 34H and <01234H is 12H.

Customizing the Assembler

The behavior of the assembler is controlled by two main tables and two case statements (one in each pass). The first table in the listing is the operand table from label RR1 through RR31. This table is coded using macros RENTRY, RERROR, RNULL and RGO. The second table is in the declaration of OTAB immediately after the operand table. This table, constructed with the macro OENTRY, describes the operation codes for the assembler. Each entry in this table gives an operation code, e.g., 'ORG', the base value for this operation, e.g., '0', and names a starting point in the operand table, e.g., 'RR30'. Each operation is applied to a serial search of this table for an exact match (both length and contents), the operation value is saved and the operand pointer is used to start the search for operands. The operand table is serially searched from the starting point given by the operation table. The operand table contains several kinds of entries, some of which exist to make the table shorter by sharing portions. Entries created by the macro RENTRY result in comparison of the start of the operand field with the table entry. An RENTRY consists of match string, e.g. 'P1, #', an operation correction value, e.g., '49' meaning 49H, which is added to the opcode to produce the correct value for this operand, and a type code, e.g. '␣IM]', which drives the case clauses for operand format in the assembler. These types are generated by macros which expand to small numbers. The types which are currently defined are error, immediate byte follows, same page branches, full memory call/jump, pseudo operation and operand complete. The actual meaning of a type is defined by the case statements in PASS1 and PASS2 described below.

Entries created by the macro RNULL are for operations which have no operands. An entry or type RNULL is identical to RENTRY with null match string. Since the search is for a match on the first few characters of an operand, if one operand form is a prefix of another for the same operation (say P and P3) the longer one should come first, otherwise both P and P3 would match with P.

Entries created by the macro RGO are used to share parts of the operand lists. An RGO entry gives an opcode correction value and names another part of the operand table. When the table lookup routines encounter an RGO entry, the search

switches to the named part of the table after adding the correction value to the opcode. This correction allows for differences in the spacing of different opcodes. For example, the table entry RR4 ends with [RGO '60' '␣RR3]'] because most of the possible operands for ADD are the same as for XCH, saving 10 entries. RR3 is part of another space saving technique. RR3 is an internal label or the declaration for RR1 (and RR2), thus, if the search falls through the entry efter RR2, the effect is the same as ␣RGO '0' '␣RR3]'] without the overhead. RGO allows several variants to join since only one can fall through, and also allows other variants by using opcode corrections.

The last type of entry is generated by the RERROR macro. This macro signals that the operand field does not contain a valid entry. Note that every path through the operand table must end with RERROR or RNULL.

PASS1 Case Statement

This assembler is a traditional two pass assembler which resolves label values on the first pass and generates code on the second pass. These two passes are embodied in the procedures PASS1 and PASS2. In each of these procedures, a case statement is driven by the results of the operand table search. In pass one, the case alternatives do processing to resolve all labels. The basic function of pass one defines all labels as the current value of the location counter (the location counter is the assembler's analog of the processor's program counter). The main function of the case statement is to increment the location counter by the length of each instruction. The second function is to provide special processing for pseudo operations. Pseudo operation processing is driven by a second case statement. The driving value for this case statement is the opcode value since pseudo operations don't generate an operation in the usual sense. The case for EQU modifies the definition of its label to the value of the expression. The case for ORG resets the location counter to its expression. The DS case increments the location counter by the value of the expression.

PASS2 Case Statement

The main action in pass two is code generation by the different cases with appropriate location counter adjustment (normally done by EMIT1, the subroutine which emits a byte of code), again with special processing in a second case statement for pseudo-operations. The calls to MAGNITUDE-ERROR are made when a two byte value is to be emitted in one byte but is not the sign extension of the single byte.

8741/8748 Assembler Notes

The language currently recognized by the assembler is the union of the operations for the Intel 8741 and 8748, and the appropriate Intel User's Manuals are the primary reference on valid formats and operands. The operations are divided into four types for this assembler.

Type 1. This type is for operations with a single immediate operand data byte, for example ADD A, #5.

Type 2. This type is for the short conditional jump instructions which are limited to destinations on the same page (256 byte block) as the next instruction, for example DJNZ R1, LOOP.

Type 3. This type is for call and jump in which three address bits are part of the opcode and the remainder of the address is in a second byte.

Type 4. This is a dummy type for pseudo operations. These are

```
        ORG     expression      ;set location counter to ex-
                                 pression
label   EQU     expression      ;equate label to expression
␣lab]   DS      expression      ;reserve expression bytes
␣lab]   DB      expression      ;define byte with value ex-
                                 pression
        PRINT   ON              ;list assembly to printer
        PRINT   OFF             ;no listing (except errors)
```

Most of these are standard pseudo operations. The PRINT pseudo operation immediately affects the listing, making it possible to list only parts of an assembly. The initial state of the assembler is PRINT ON.

Type 5. This case is for simple opcodes with operands included within a single byte, for example MOV A, @R1.

Miscellaneous Considerations

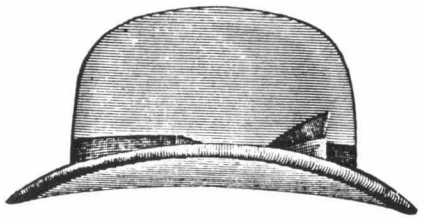
As compiled here, the assembler is limited to 255 labels by the table size declaration, but the search strategy is general enough for any length table. To simplify I/O, the assembler also presently uses the fact that there is only 1K of EPROM in an Intel 8748/8741, and first generates an in-memory array of the result and fianlly writes a binary file of the result. For larger memories, such as an 8080, EMIT1 needs to be revised to write to an output file, in a format such as Intel hex. Since I wrote this on my CP/M system, the I/O used is specific to CP/M but has been carefully encapsulated in only a few subroutines. Procedure GETFC returns successive characters from the input file, and thus is responsible for reading disk sectors and dribbling the contents out one character at a time. Procedures PRINTALINE and PRMSG call CP/M BDOS to type characters on the console. PRINTALINE lists lines of formatted assembly output, while PRMSG types error messages. The procedures ZTFCB, OPENR and OPENHEX also perform CP/M specific functions. These procedures all assume the existence of a CP/M initialized file control block which CP/M has parsed the required file name from the command invoking the assembler. For another DOS, it may be necessary to do this parsing separately, and change the format assumptions made by these routines. ZTFCB initializes a file control block prior to opening. OPENR opens a file control block to read the source file, and is called twice, once for each pass. OPENHEX creates a new file with suffix SCC and writes the memory image buffer of the assembled code out in binary format. Outside of these routines, the assembler makes no assumptions about the file structures available except that it is a sequence of bytes which can be read twice. Conversion to another DOS should be fairly easy, and conversion to a non-disk system should also be feasible.

Program Availability

The ML-80 compiler is part of the CP/M users group collection, and so is available from them. The PL/M source for ML-80 is available from the Intel Users' Group. As an alternative, for $10 I will provide an 8 inch, single density IBM format CP/M diskette which contains ML-80 and the source for the assembler in this article.

```
?

A>TYPE ASM874X.M80
[MACRO IM '1'] [MACRO EN'5'] [MACRO CJ '3']
[MACRO BRA '2'] [MACRO PSU '4']          /* 0 RESERVED FOR ERROR */
[MACRO RERROR '80H']
[MACRO RGO OPPLUS ENTRY '81H,[OPPLUS]H,.[ENTRY]']
[MACRO RENTRY ARGS OPPLUS TYPE '''[ARGS]'',0,[OPPLUS]H,[TYPE]']
[MACRO RNULL OPPLUS TYPE '0,[OPPLUS]H,[TYPE]']

DECLARE RR1 DATA([RENTRY 'P1,#' '49' '[IM]'],   /* ANL, ORL */
                 [RENTRY 'P2,#' '4A' '[IM]'],
                 [RENTRY 'BUS,#' '48' '[IM]'],
[MACRO RR2 'RR1(14)'] /* XRL + AN,OR,MOV */
                 [RENTRY 'A,#' '3' '[IM]'],
[MACRO RR3 'RR1(20)'] /* XCH + ANL,ORL,XRL,MOV,ADD,ADDC */
                 [RENTRY 'A,@R0' '0' '[EN]'],
                 [RENTRY 'A,@R1' '1' '[EN]'],
                 [RENTRY 'A,R0' '08' '[EN]'],
                 [RENTRY 'A,R1' '09' '[EN]'],
                 [RENTRY 'A,R2' '0A' '[EN]'],
                 [RENTRY 'A,R3' '0B' '[EN]'],
                 [RENTRY 'A,R4' '0C' '[EN]'],
                 [RENTRY 'A,R5' '0D' '[EN]'],
                 [RENTRY 'A,R6' '0E' '[EN]'],
                 [RENTRY 'A,R7' '0F' '[EN]'],
                 [RERROR]);
DECLARE RR4 DATA([RENTRY 'A,#' '3' '[IM]'],     /* ADD, ADDC */
                 [RGO '60' '[RR3]']);
DECLARE RR5 DATA([RENTRY 'A,P4' '4' '[EN]'],    /* MOVD */
                 [RENTRY 'A,P5' '5' '[EN]'],
                 [RENTRY 'A,P6' '6' '[EN]'],
                 [RENTRY 'A,P7' '7' '[EN]'],
[MACRO RR5A 'RR5(28)'] /* ANLD, ORLD + MOVD */
                 [RENTRY 'P4,A' '34' '[EN]'],
                 [RENTRY 'P5,A' '35' '[EN]'],
                 [RENTRY 'P6,A' '36' '[EN]'],
                 [RENTRY 'P7,A' '37' '[EN]'],
                 [RERROR]);
DECLARE RR6 DATA([RNULL '0' '[CJ]']);   /* CALL, JMP */
DECLARE RR7 DATA([RENTRY 'F1' '0A5' '[EN]'],    /* CLR, CPL */
                 [RENTRY 'C' '97' '[EN]'],
                 [RENTRY 'F0' '85' '[EN]'],
[MACRO RR8 'RR7(14)'] /* DA, RL, RLC, RR, RRC + CLR, CPL */
                 [RENTRY 'A' '27' '[EN]'],
                 [RERROR]);
DECLARE RR9 DATA([RNULL '0' '[EN]']);   /* NOP, RET */
DECLARE RR10 DATA([RENTRY 'DBB,A' '0' '[EN]'], /* OUT */
                 [RERROR]);
DECLARE RR11 DATA([RENTRY 'P1,A' '1' '[EN]'],   /* OUTL */
                 [RENTRY 'P2,A' '2' '[EN]'],
                 [RENTRY 'BUS,A' '0CA' '[EN]'],
                 [RERROR]);
DECLARE RR12 DATA([RENTRY 'RB0' '0' '[EN]'],    /* SEL */
                 [RENTRY 'RB1' '10' '[EN]'],
                 [RENTRY 'MB0' '20' '[EN]'],
                 [RENTRY 'MB1' '30' '[EN]'],
                 [RERROR]);
DECLARE RR13 DATA([RENTRY 'TCNT' '5' '[EN]'],   /* STOP */
                 [RERROR]);
DECLARE RR14 DATA([RENTRY 'CNT' '0' '[EN]'],    /* STRT */
                 [RENTRY 'T' '10' '[EN]'],
                 [RERROR]);
DECLARE RR15 DATA([RENTRY 'A,@R0' '0' '[EN]'], /* XCHD */
                 [RENTRY 'A,@R1' '1' '[EN]'],
                 [RERROR]);
DECLARE RR16 DATA([RNULL '0' '[BRA]']); /* CONDITIONAL JUMPS */
DECLARE RR17 DATA([RENTRY '@A' '0' '[EN]'],     /* JMPP */
                 [RERROR]);
DECLARE RR18 DATA([RENTRY 'A,#' '23' '[IM]'],   /* MOV */
                 [RENTRY 'A,PSW' '0C7' '[EN]'],
                 [RENTRY 'A,T' '42' '[EN]'],
                 [RENTRY 'STS,A' '90' '[EN]'],
                 [RENTRY 'PSW,A' '0D7' '[EN]'],
                 [RENTRY 'R0,A' '0A8' '[EN]'],
                 [RENTRY 'R1,A' '0A9' '[EN]'],
                 [RENTRY 'R2,A' '0AA' '[EN]'],
                 [RENTRY 'R3,A' '0AB' '[EN]'],
                 [RENTRY 'R4,A' '0AC' '[EN]'],
                 [RENTRY 'R5,A' '0AD' '[EN]'],
                 [RENTRY 'R6,A' '0AE' '[EN]'],
                 [RENTRY 'R7,A' '0AF' '[EN]'],
                 [RENTRY 'R0,#' '0B8' '[IM]'],
                 [RENTRY 'R1,#' '0B9' '[IM]'],
                 [RENTRY 'R2,#' '0BA' '[IM]'],
                 [RENTRY 'R3,#' '0BB' '[IM]'],
                 [RENTRY 'R4,#' '0BC' '[IM]'],
                 [RENTRY 'R5,#' '0BD' '[IM]'],
                 [RENTRY 'R6,#' '0BE' '[IM]'],
                 [RENTRY 'R7,#' '0BF' '[IM]'],
                 [RENTRY '@R0,A' '0A0' '[EN]'],
                 [RENTRY '@R1,A' '0A1' '[EN]'],
                 [RENTRY '@R0,#' '0B0' '[IM]'],
                 [RENTRY '@R1,#' '0B1' '[IM]'],
                 [RENTRY 'T,A' '62' '[EN]'],
                 [RGO '0F0' '[RR2]']);
DECLARE RR19 DATA([RENTRY 'A,@A' '0' '[EN]'],   /* MOVP, MOVP3 */
[MACRO RRXX 'RR19(7)'] /* UNDEFINED OPCODES */
                 [RERROR]);
DECLARE RR20 DATA([RENTRY '@R0' '0' '[EN]'],    /* INC */
                 [RENTRY '@R1' '1' '[EN]'],
                 [RENTRY 'A' '7' '[EN]'],
[MACRO RR21 'RR20(16)'] /* +INC, DEC */
                 [RENTRY 'R0' '8' '[EN]'],
                 [RENTRY 'R1' '9' '[EN]'],
                 [RENTRY 'R2' '0A' '[EN]'],
                 [RENTRY 'R3' '0B' '[EN]'],
                 [RENTRY 'R4' '0C' '[EN]'],
                 [RENTRY 'R5' '0D' '[EN]'],
                 [RENTRY 'R6' '0E' '[EN]'],
                 [RENTRY 'R7' '0F' '[EN]'],
                 [RERROR]);
DECLARE RR22 DATA([RENTRY 'A' '7' '[EN]'],      /* DEC */
                 [RGO '0C0' '[RR21]']);
DECLARE RR23 DATA([RENTRY 'DMA' '0E5' '[EN]'],  /* EN */
                 [RENTRY 'FLAGS' '0F5' '[EN]'],
[MACRO RR24 'RR23(14)'] /* DIS + EN */
                 [RENTRY 'I' '5' '[EN]'],
                 [RENTRY 'TCNTI' '25' '[EN]'],
                 [RERROR]);
DECLARE RR25 DATA([RENTRY 'R0,' '8' '[BRA]'],   /* DJNZ */
                 [RENTRY 'R1,' '9' '[BRA]'],
                 [RENTRY 'R2,' '0A' '[BRA]'],
                 [RENTRY 'R3,' '0B' '[BRA]'],
                 [RENTRY 'R4,' '0C' '[BRA]'],
                 [RENTRY 'R5,' '0D' '[BRA]'],
                 [RENTRY 'R6,' '0E' '[BRA]'],
                 [RENTRY 'R7,' '0F' '[BRA]'],
                 [RERROR]);
DECLARE RR26 DATA([RENTRY 'A,DBB' '22' '[EN]'], /* IN */
                 [RENTRY 'A,P1' '9' '[EN]'],
                 [RENTRY 'A,P2' '0A' '[EN]'],
                 [RERROR]);
DECLARE RR27 DATA([RENTRY 'CLK' '75' '[EN]'],   /* ENTO */
                 [RERROR]);
DECLARE RR28 DATA([RENTRY 'A,BUS' '8' '[EN]'],  /* INS */
                 [RERROR]);
DECLARE RR29 DATA([RENTRY 'A,@R0' '80' '[EN]'], /* MOVX */
                 [RENTRY 'A,@R1' '81' '[EN]'],
                 [RENTRY '@R0,A' '90' '[EN]'],
                 [RENTRY '@R1,A' '91' '[EN]'],
                 [RERROR]);
DECLARE RR30 DATA([RNULL '0' '[PSU]']);
DECLARE RR31 DATA([RENTRY 'ON' '0' '[PSU]'],
                 [RENTRY 'OFF' '1' '[PSU]'],
                 [RERROR]);
[MACRO OENTRY OP OPVAL OPERANDS '''[OP]'',0,[OPVAL]H,.[OPERANDS]']

DECLARE OTAB DATA(
[OENTRY 'ORG' '0' 'RR30'],
[OENTRY 'EQU' '1' 'RR30'],
[OENTRY 'DS' '2' 'RR30'],
[OENTRY 'DB' '3' 'RR30'],
[OENTRY 'ANL' '50' 'RR1'],
[OENTRY 'ORL' '40' 'RR1'],
[OENTRY 'XRL' '0D0' '[RR2]'],
[OENTRY 'XCH' '20' '[RR3]'],
[OENTRY 'ADD' '0' 'RR4'],
[OENTRY 'ADDC' '10' 'RR4'],
[OENTRY 'MOVD' '8' 'RR5'],
[OENTRY 'ANLD' '68' '[RR5A]'],
[OENTRY 'ORLD' '58' '[RR5A]'],
[OENTRY 'CALL' '14' 'RR6'],
[OENTRY 'JMP' '4' 'RR6'],
[OENTRY 'CLR' '0' 'RR7'],
[OENTRY 'CPL' '10' 'RR7'],
[OENTRY 'DA' '30' '[RR8]'],
[OENTRY 'NOP' '0' 'RR9'],
[OENTRY 'RET' '83' 'RR9'],
[OENTRY 'RETR' '93' 'RR9'],
[OENTRY 'OUT' '2' 'RR10'],
[OENTRY 'OUTL' '38' 'RR11'],
[OENTRY 'RL' '0C0' '[RR8]'],
[OENTRY 'RLC' '0D0' '[RR8]'],
[OENTRY 'RR' '50' '[RR8]'],
[OENTRY 'RRC' '40' '[RR8]'],
[OENTRY 'SWAP' '20' '[RR8]'],
[OENTRY 'SEL' '0C5' 'RR12'],
[OENTRY 'STOP' '60' 'RR13'],
[OENTRY 'STRT' '45' 'RR14'],
[OENTRY 'XCHD' '30' 'RR15'],
[OENTRY 'JMPP' '0B3' 'RR17'],
[OENTRY 'JB0' '12' 'RR16'],
[OENTRY 'JB1' '32' 'RR16'],
[OENTRY 'JB2' '52' 'RR16'],
[OENTRY 'JB3' '72' 'RR16'],
[OENTRY 'JB4' '92' 'RR16'],
[OENTRY 'JB5' '0B2' 'RR16'],
[OENTRY 'JB6' '0D2' 'RR16'],
[OENTRY 'JB7' '0F2' 'RR16'],
[OENTRY 'JC' '0F6' 'RR16'],
[OENTRY 'JF0' '0B6' 'RR16'],
[OENTRY 'JF1' '76' 'RR16'],
[OENTRY 'JNC' '0E6' 'RR16'],
[OENTRY 'JNIBF' '0D6' 'RR16'],
[OENTRY 'JNI' '86' 'RR16'],
[OENTRY 'JNT0' '26' 'RR16'],
[OENTRY 'JNT1' '46' 'RR16'],
[OENTRY 'JNZ' '96' 'RR16'],
[OENTRY 'JOBF' '86' 'RR16'],
[OENTRY 'JTF' '16' 'RR16'],
[OENTRY 'JT0' '36' 'RR16'],
[OENTRY 'JT1' '56' 'RR16'],
[OENTRY 'JZ' '0C6' 'RR16'],
[OENTRY 'MOV' '0' 'RR18'],
[OENTRY 'MOVP' '0A3' 'RR19'],
[OENTRY 'MOVP3' '0E3' 'RR19'],
[OENTRY 'INC' '10' 'RR20'],
[OENTRY 'DEC' '0' 'RR22'],
[OENTRY 'EN' '0' 'RR23'],
[OENTRY 'DIS' '10' '[RR24]'],
[OENTRY 'DJNZ' '0E0' 'RR25'],
[OENTRY 'IN' '0' 'RR26'],
[OENTRY 'ENT0' '0' 'RR27'],
[OENTRY 'INS' '0' 'RR28'],
[OENTRY 'MOVX' '0' 'RR29'],
[OENTRY 'PRINT' '4' 'RR31'],
0,.[RRXX]);
DECLARE PRNTLIN(120)BYTE;
DECLARE RANDLOC(2) BYTE,
        OP(6) BYTE,
        LABELL(7) BYTE;
DECLARE OPCODE BYTE,
        RANDLIST(2) BYTE;
```

```
[MACRO LF '0AH']        [MACRO EOF '1AH']       [MACRO BDOS '5']
[MACRO CR '0DH']        [MACRO TAB '9']
DECLARE IBUFP BYTE;
[MACRO DBUF '80H']      [MACRO IFCB '5CH']      [MACRO READNEXT '20']
[MACRO OPENF '15']      [MACRO CLOSEF '16']
DECLARE LISTOPT BYTE INITIAL(1);
DECLARE ERRPF BYTE, PASS BYTE INITIAL(0);

GETFC:PROCEDURE;
        STACK=HL;
        IF (A=IBUFP; A::128)ZERO THEN DO;
                IBUFP=(A=A-A);
                C=[READNEXT];
                DE=[IFCB];      CALL [BDOS];
                IF (A=A\A) !ZERO THEN DO;
                        HL=STACK;
                        A=[EOF];
                        RETURN;
                END;
        END;
        HL=.IBUFP;
        E=M(HL);
        M(HL)=M(HL)+1;
        D=0;    HL=[DBUF]+DE;
        A=M(HL);
        HL=STACK;
END;

[MACRO PUTC '2']
PRINTALINE:PROCEDURE;
        IF (A=ERRPF\A) ZERO & (HL=.PASS; A=LISTOPT & M(HL)) ZERO THEN RETURN;
        HL=.PRNTLIN;
        DO WHILE(A=M(HL); A::[LF]) !ZERO;
                STACK=HL;
                C=[PUTC];       E=A;    CALL [BDOS];
                HL=STACK+1;
        END;
        C=[PUTC];       E=A;    CALL [BDOS];
END;


GETLINE:PROCEDURE;
        HL=.PRNTLIN;    C=24;
        REPEAT;
                M(HL)=20H;      HL=HL+1;
        UNTIL (C=C-1) ZERO;
        REPEAT;
                CALL GETFC;
                M(HL)=A;
                HL=HL+1;
        UNTIL (A::[LF]) ZERO \ (A::[EOF])ZERO;
END;

[MACRO CONSTRING '9']
PRMSG:PROCEDURE;
        C=[CONSTRING];  CALL [BDOS];
        C=[PUTC];       E=[CR]; CALL [BDOS];
        C=[PUTC];       E=[LF]; CALL [BDOS];
END;

AERROR:PROCEDURE;
        STACK=PSW;      STACK=HL;       STACK=BC;
        CALL PRMSG;
        ERRPF=(A=1);
        BC=STACK;       HL=STACK;       PSW=STACK;
END;

SYNERR:PROCEDURE;
        DE=.'SYNTAX ERROR IN:$';
        CALL AERROR;
END;

PARSELINE:PROCEDURE;
        ERRPF=(A=A-A);  /* RESET FOR NEW LINE */
        REPEAT;
                CALL GETLINE;
```

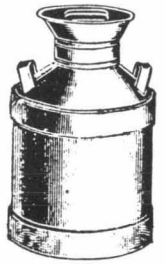


```
                IF (A::[EOF]) ZERO RETURN;
                IF (A=PRNTLIN(24)-3BH) ZERO CALL PRINTALINE/* COMMENT */;
        UNTIL (A=PRNTLIN(24);A::3BH) !ZERO;
        C=6;    HL=.OP;
        REPEAT; M(HL)=0;        HL=HL+1;        UNTIL (C=C-1) ZERO;
        C=7;    HL=.LABELL;
        REPEAT; M(HL)=0;        HL=HL+1;        UNTIL (C=C-1) ZERO;
        HL=.PRNTLIN(24);        DE=.LABELL;     C=8;
        DO WHILE (A=M(HL); A::21H) !CY
                & (A::3AH) !ZERO
                & (C=C-1) !ZERO;
                /* IE WHILE NOT CONTROL CHAR, BLANK, COLON OR TOO LONG*/
                M(DE)=A;
                HL=HL+1;        DE=DE+1;
        END;
        IF (A::3AH)!ZERO & (A::20H) !ZERO & (A::[TAB]) !ZERO THEN DO;
                /* LABEL TOO LONG OR IMPROPER TERMINATION */
                CALL SYNERR;
                DO WHILE(A=M(HL);A::21H) !CY & (A::3AH)!ZERO;
                        HL=HL+1;        /* SCAN TO BREAK */
                END;
        END;
        IF (A::3AH)ZERO THEN HL=HL+1;   /* IGNORE COLON */
        DO WHILE(A=M(HL);A::20H) ZERO \ (A::[TAB]) ZERO;        HL=HL+1;        END;
        DE=.OP; C=7;
        DO WHILE(A=M(HL); A::21H) !CY & (C=C-1) !ZERO;
                M(DE)=A;
                HL=HL+1;        DE=DE+1;
        END;
        IF(A::20H)!ZERO & (A::[TAB])!ZERO &(A::[CR])!ZERO THEN DO;
                CALL SYNERR;
                DO WHILE(A=M(HL);A::21H)!CY;    HL=HL+1;        END;
        END;
        DO WHILE(A=M(HL);A::20H) ZERO \ (A::[TAB]) ZERO;        HL=HL+1;        END;
        RANDLOC=HL;     /* SAVE ADDRESS OF OPERANDS */
END;

FINDOP:PROCEDURE;
        /* EACH ENTRY IN OTAB HAS FORMAT
        CHARS, 0, OPVALUE(1),OPERAND-TABLE-ADDRESS(2)           */
        DECLARE FOUND LABEL;
        HL=.OTAB;
        REPEAT;
                DE=.OP;
                DECLARE NEQ LABEL;
                REPEAT;
                        IF(A=M(DE) - M(HL)) !ZERO GOTO NEQ;
                        A=M(HL)\A;
                        HL=HL+1;        DE=DE+1;
                UNTIL /* A */ ZERO;
                GOTO FOUND;
        NEQ:    DO WHILE (A=M(HL)\A) !ZERO;     HL=HL+1;        END;
                HL=HL+(DE=4);
        UNTIL (A=M(HL)\A) ZERO;
FOUND:  /* HL -> CODE BYTE */
        OPCODE=(A=M(HL));
        HL=HL+1;        E=M(HL);
        HL=HL+1;        D=M(HL);
        HL==DE; RANDLIST=HL;
END;

DECLARE OPCLASS BYTE;
FINDRAND:PROCEDURE;
        HL=RANDLIST;    HL==DE;
        /* ENTRY IN OPERANT TABLES
        STRING,0,ADDED-VALUE,TYPE       */
        DO WHILE (A=M(DE); A::[RERROR])!ZERO;
                IF (A::81H /* RGO */)ZERO THEN DO;
                        DE=DE+1;        L=(A=M(DE));
                        OPCODE=(A=OPCODE+L);
                        DE=DE+1;        L=(A=M(DE));
                        DE=DE+1;        H=(A=M(DE));
                        HL==DE;
                END
                ELSE DO;        /* SEE IF MATCHES ENTRY */
                        DECLARE NEQ LABEL;
```

```
                        HL=RANDLOC;
                        DO WHILE (A=M(DE)\A) !ZERO;
                                IF(A::M(HL))!ZERO GOTO NEQ;
                                HL=HL+1;        DE=DE+1;
                        END;
                        RANDLOC=HL;     /* RESUME FROM HERE LATER */
                        DE=DE+1;        L=(A=M(DE));
                        OPCODE=(A=OPCODE+L);
                        DE=DE+1;        OPCLASS=(A=M(DE));
                        RETURN;
                NEQ:    REPEAT; DE=DE+1;        UNTIL(A=M(DE)\A)ZERO;
                        DE=DE+1,+1,+1;
                END;
        END;
        OPCLASS=(A=A-A);        /* ERROR CLASS */
END;

ASCTOHEX:PROCEDURE;     /*BC-> CHAR STRING, NUL TERMINATED */
ASCHX1:PROCEDURE;
        A=A-30H;
        IF(A::10) CY RETURN;
        A=A-7;
  END;
        HL=0;
        DO WHILE (A=M(BC)\A) !ZERO;
                HL=HL+HL,+HL,+HL,+HL;/* TIMES 16 */
                CALL ASCHX1;
                IF (A::16; CY= !CY) CY RETURN;
                L=(A=A+L);
                BC=BC+1;
        END;
END;

ASCTODEC:PROCEDURE;
        HL=0;
        DO WHILE (A=M(BC)\A) !ZERO;
                DE=HL;
                HL=HL+HL,+HL,+DE,+HL;   /* TIMES 10 */
                A=A-30H;
                IF(A::10; CY=!CY) CY RETURN;
                E=A;    D=0;
                HL=HL+DE;
                BC=BC+1;
        END;
END;

ASCTOOCT:PROCEDURE;
        HL=0;
        DO WHILE(A=M(BC)\A) !ZERO;
                HL=HL+HL,+HL,+HL;       /* TIMES 8 */
                A=A-30H;
                IF(A::8; CY=!CY) CY RETURN;
                L=(A=A+L);
                BC=BC+1;
        END;
END;

ASCTOBIN:PROCEDURE;
        HL=0;
        DO WHILE (A=M(BC)\A) !ZERO;
                HL=HL+HL;       /* TIMES 2 */
                A=A-30H;
                IF(A::2; CY=!CY) CY RETURN;
                L=(A=A+L);
                BC=BC+1;
        END;
END;

DECLARE LABELTAB([DEC 255*8]) BYTE INITIAL(0);
DECLARE THISLABAD(2) BYTE;

FINDLABEL:PROCEDURE;
        DECLARE FOUND LABEL;
        HL=.LABELTAB;
        DO WHILE (A=M(HL)\A) !ZERO;
                DECLARE NEQ LABEL;
```

```
                DE=.LABELL;
                C=6;    /* # CHARS IN LABEL */
                REPEAT;
                        IF (A=M(DE)-M(HL))!ZERO GOTO NEQ;
                        HL=HL+1;        DE=DE+1;
                UNTIL (C=C-1) ZERO;
                GOTO FOUND;
        NEQ:    REPEAT; HL=HL+1;        UNTIL (C=C-1) ZERO;
                HL=HL+1,+1;
        END;
        /* NOT IN LABEL TABLE, INSERT HERE */
        DE=.LABELL;
        C=6;
        REPEAT;
                M(HL)=(A=M(DE));
                HL=HL+1;        DE=DE+1;
        UNTIL (C=C-1) ZERO;
        THISLABAD=HL;
        M(HL)=81H;      HL=HL+1;        M(HL)=81H;      /* 8181H=UNINIALIZED */
        HL=HL+1;        M(HL)=(A=A-A);  /*A=0, NOT FOUND */
        RETURN;
FOUND:  THISLABAD=HL;
        A=A\1;  /* A!= ZERO, FOUND */
END;

DECLARE ASMPC(2) BYTE INITIAL(0,0);
DECLARE COREIM(1024) BYTE;

GETNUMBER:PROCEDURE;
        DECLARE NUMSTR(18) BYTE;
        DECLARE EXIT LABEL;
        HL=RANDLOC;     DE=.NUMSTR;     C=18;
        DO WHILE (C=C-1) !ZERO
                & (A=M(HL); A::30H) !CY /* A >= '0' */
                & (A::50H) CY;  /* A < P */
                IF (A :: 3AH) !CY & (A :: 41H) CY THEN GOTO EXIT;
                M(DE)=A;
                HL=HL+1;        DE=DE+1;
        END;
EXIT:
        M(DE)=(A=A-A);
        RANDLOC=HL;     DE=DE-1;        /* LAST CHAR */
        IF (A=C; A::17) ZERO CALL SYNERR;       /* NO NUMBER */
        A=M(DE);        BC=.NUMSTR;
        IF (A::48H) ZERO THEN DO;
                M(DE)=(A=A-A);  CALL ASCTOHEX;  END
        ELSE IF (A::4FH)ZERO THEN DO;
                M(DE)=(A=A-A);  CALL ASCTOOCT;  END
        ELSE IF (A::42H) ZERO THEN DO;
                M(DE)=(A=A-A);  CALL ASCTOBIN;  END
        ELSE CALL ASCTODEC;
        /* HL = VALUE */
END;

GETNAMEVAL:PROCEDURE;
        DECLARE EXIT LABEL;
ERRUNDEF:PROCEDURE;
        DE=.'UNDEFINED NAME IN EXPRESSION$';
        CALL AERROR;
END;
        A=A-A;  C=6;    DE=.LABELL(6);
                                        REPEAT;
                DE=DE-1;        M(DE)=A;
        UNTIL (C=C-1) ZERO;
        HL=RANDLOC;     C=7;
        DO WHILE (C=C-1) !ZERO
                & (A=M(HL); A::30H) !CY
                & (A::5BH) CY;
                IF (A::3AH) !CY & (A::41H) CY THEN GOTO EXIT;
                M(DE)=A;
                DE=DE+1;        HL=HL+1;
        END;
EXIT:   IF (A=C-6) ZERO THEN /* NO LEGAL IDENT FORCE ADVANCE */ HL=HL+1;
        RANDLOC=HL;
        CALL FINDLABEL;
        HL=THISLABAD;
```

```
        IF (A=M(HL);    A::81H) ZERO
                & (HL=HL+1; A::M(HL); HL=HL-1) ZERO THEN CALL ERRUNDEF;
        A=M(HL);        HL=HL+1;
        H=M(HL);        L=A;
END;

GETEXPR:PROCEDURE;
        DECLARE OPERAND(2) BYTE,        OPIND BYTE,     SIGN BYTE;
ASMERR:PROCEDURE;
        DE=.'ASM874X ERROR CAUSED IN:$';
        CALL AERROR;
END;
        OPERAND=(HL=0);
        OPIND=(A=1);
        SIGN=(A=2BH /* + */);
        DO WHILE (HL=RANDLOC;   A=M(HL);        A::21H) !CY
                & (A::2CH /* , */) !ZERO;
                IF (A :: 2BH /* + */) ZERO \ (A::3EH /* > */) ZERO
                        \(A::3CH /* < */)ZERO \(A::2DH /* - */) ZERO THEN DO;
                        SIGN=A;
                        A=OPIND;
                        IF (A::2) ZERO THEN CALL SYNERR
                        ELSE OPIND=(A=2);
                        RANDLOC=(HL=HL+1);
                END
                ELSE DO;
                        C=A;
                        IF(A=OPIND \ A) ZERO CALL SYNERR;
                        A=C;
                        IF (A::24H /* $ */)ZERO THEN DO;
                                RANDLOC=(HL=HL+1);      /* RESUME POINT */
                                HL=ASMPC;
                        END
                        ELSE IF (A::27H /* ' */) ZERO THEN DO;
                                HL=HL+1;        D=0;
                                IF (A=M(HL);    E=A;    A::27H) !ZERO THEN DO;
                                        HL=HL+1;
                                        IF(A=M(HL);     A::27H) !ZERO CALL SYNERR;
                                        RANDLOC=(HL=HL+1);
                                END
                                ELSE IF (HL=HL+1;       A=M(HL) -27H)!ZERO
                                        \(HL=HL+1;      A=M(HL)-27H)!ZERO
                                        THEN CALL SYNERR
                                ELSE RANDLOC=(HL=HL+1);
                                HL==DE;
                        END
                        ELSE IF (A::30H) CY THEN DO;
                                RANDLOC=(HL=HL+1);      /* ENSURE SCAN ADVANCES*/
                                CALL SYNERR;
                        END
                        ELSE IF (A::3AH) CY THEN DO;    /* A NUMBER */
                                IF (CALL GETNUMBER) CY CALL SYNERR;
                        END
                        ELSE DO;
                                CALL GETNAMEVAL;
                        END;
                        HL==DE;
                        HL=OPERAND;
                        OPIND=(A=A-A);
                        A=SIGN;
                        IF (A::2BH /* + */) ZERO THEN HL=HL+DE
                        ELSE IF(A::2DH /* - */)ZERO THEN DO;
                                L=(A=L-E);      H=(A=H--D);
                        END
                        ELSE IF (A::3EH /* > */) ZERO THEN DO;
                                L=E;    H=0;
                        END
                        ELSE IF (A::3CH /* < */)ZERO THEN DO;
                                L=D;    H=0;
                        END
                        ELSE CALL ASMERR;
                        OPERAND=HL;
                END;
        END /* WHILE */;
        IF (A=OPIND\A) !ZERO CALL SYNERR;
                HL=OPERAND;     /* RETURN VALUE */
                A=H\A:  /* SET FLAGS */
END;

DECLARE OHEXPTR(2) BYTE;

PLANTCH:PROCEDURE;
        STACK=HL;
        HL=OHEXPTR;
        M(HL)=A;
        OHEXPTR=(HL=HL+1);
        HL=STACK;
END;

BYTETOHEX:PROCEDURE;

RHEX:PROCEDURE;
        A=#(A=#(A=A & 0FH, + 90H), ++40H);
END;

LHEX:PROCEDURE;
        A=>(A=>(A=>(A=>A)));
        CALL RHEX;
END;

        STACK=PSW;
        CALL LHEX;      CALL PLANTCH;
        PSW=STACK;
        CALL RHEX;      CALL PLANTCH;
END;

ADDRTOHEX:PROCEDURE;
        A=H;    CALL BYTETOHEX;
        A=L;    CALL BYTETOHEX;
        A=20H;  CALL PLANTCH;
END;

EMIT1:PROCEDURE;        /* PUT A AT ASMPC IN COREIM */
        HL=ASMPC+(DE=.COREIM);
        M(HL)=A;
        ASMPC=(HL=ASMPC+1);
        CALL BYTETOHEX;
        A=20H;  CALL PLANTCH;
END;

[MACRO SETDMA '26']     [MACRO WRITEF '21']
[INT TFCB]      [TFCB := 5CH]
PUTHEX:PROCEDURE;
        HL=.COREIM;     D=8;    /* WRITE 8 SECTORS FROM COREIM */
        REPEAT;
                STACK=DE;       STACK=HL;
                HL==DE; C=[SETDMA];     CALL [BDOS];
                DE=[TFCB];      C=[WRITEF];     CALL [BDOS];
                HL=STACK+(DE=80H);
                DE=STACK;
        UNTIL (D=D-1) ZERO;
END;

CLOSEHEX:PROCEDURE;
        DE=[TFCB];      C=[CLOSEF];     CALL [BDOS];
END;

PASS1:PROCEDURE;

INSERTLABEL:PROCEDURE;
        CALL FINDLABEL;
        IF ZERO THEN DO;        /* NEW LABEL */
                HL=ASMPC;       HL==DE;
                HL=THISLABAD;
                M(HL)=E;        HL=HL+1;
                M(HL)=D;
        END
        ELSE DO;
                DE=.'DUPLICATED LABEL IN:$';
                CALL AERROR;
        END;
END;
```



```
NERROR:PROCEDURE;
        DE=.'UNRECOGNIZED OPERATION IN:$';
        CALL AERROR;
END;

EQUSYNERR:PROCEDURE;
        DE=.'EQUATE HAS NO LABEL$';
        CALL AERROR;
END;
        DECLARE TEMP(2) BYTE;
NXL:    CALL PARSELINE;
        IF (A=PRNTLIN(24); A::[EOF]) ZERO RETURN;
        IF (A=LABELL\A) !ZERO CALL INSERTLABEL;
        CALL FINDOP;
        CALL FINDRAND;
        L=(A=OPCLASS);  H=0;
        DO CASE HL;
                /* 0 */ CALL NERROR;
                /* 1 IMMEDIATE */ ASMPC=(HL=ASMPC+1,+1);
                /* 2 BRANCH */  ASMPC=(HL=ASMPC+1,+1);
                /* 3 CALL/JUMP */       ASMPC=(HL=ASMPC+1,+1);
                /* 4 PSEUDO OPS */      DO;
                        L=(A=OPCODE);   H=0;
                        DO CASE HL;
                                /* 0 ORG */     DO;
                                        CALL GETEXPR;
                                        ASMPC=HL;
                                END;
                                /* 1 EQU */     DO;
                                        TEMP=(HL=THISLABAD);
                                        IF (A=LABELL\A) ZERO CALL EQUSYNERR;
                                        CALL GETEXPR;   HL==DE;
                                        HL=TEMP;
                                        M(HL)=E;        HL=HL+1;
                                        M(HL)=D;
                                END;
                                /* 2 DS */      DO;
                                        CALL GETEXPR;   HL==DE;
                                        ASMPC=(HL=ASMPC+DE);
                                END;
                                /* 3 DB */      ASMPC=(HL=ASMPC+1);
                                /* 4 PRINT ON */        NOP;
                                /* 5 PRINT OFF */       NOP;
                        END;/* CASE */
                END;
                /* 5 SIMPLE OP */       ASMPC=(HL=ASMPC+1);
        END CASECLASS;
        CALL PRINTALINE;
        GOTO NXL;
END PASS1;

PASS2:PROCEDURE;
        DECLARE TEMP(2) BYTE;

MAGNITUDEERROR:PROCEDURE;
        DE=.'VALUE TOO LARGE FOR FIELD (TRUNCATED):$';
        CALL AERROR;
END;

PAGEERROR:PROCEDURE;
        DE=.'JUMP DESTINATION NOT ON SAME PAGE:$';
        CALL AERROR;
END;

        PASS=(A=1);
NXL:    CALL PARSELINE;
        OHEXPTR=(HL=.PRNTLIN);
        HL=ASMPC;       CALL ADDRTOHEX;
        IF (A=PRNTLIN(24); A::[EOF]) ZERO RETURN;
        CALL FINDOP;
        CALL FINDRAND;
        L=(A=OPCLASS);  H=0;
        DO CASE HL;
                /* 0 ERROR */   NOP;    /* MESSAGE IN PASS 1 */
                /* 1 IMMEDIATE */       DO;
                        A=OPCODE;       CALL EMIT1;
                        CALL GETEXPR;
                        IF (A=H\A) !ZERO & (A=A+1) !ZERO THEN CALL MAGNITUDEERROR;
                        A=L;    CALL EMIT1;
                END;
                /* 2 BRANCH ON CURRENT PAGE */  DO;
                        A=OPCODE;       CALL EMIT1;
                        CALL GETEXPR;
                        IF (A=ASMPC(1)-H) !ZERO CALL PAGEERROR;
                        A=L;    CALL EMIT1;
                END;
                /* 3 CALL/JUMP */       DO;
                        CALL GETEXPR;   STACK=HL;
                        IF (A=H & 0F8H) !ZERO CALL MAGNITUDEERROR;
                        D=(A=OPCODE);
                        A=>>(A=>>(A=>>(A=H & 7))), +D;  CALL EMIT1;
                        HL=STACK;       A=L;    CALL EMIT1;
                END;
                /* 4 PSUEDO OPS */      DO;
                        L=(A=OPCODE);   H=0;
                        DO CASE HL;
                                /* 0 ORG */     DO;
                                        CALL GETEXPR;   ASMPC=HL;
                                        CALL ADDRTOHEX;
                                END;
                                /* 1 EQU */     NOP;
                                /* 2 DS */      DO;
                                        CALL GETEXPR;   HL==DE;
                                        ASMPC=(HL=ASMPC+DE);
                                END;
                                /* 3 DB */      DO;
                                        CALL GETEXPR;
                                        IF (A=H\A) !ZERO & (A=A+1) !ZERO THEN CALL MAGNITUDEE
                                        A=L;    CALL EMIT1;
                                END;
                                /* 4 PRINT ON */        LISTOPT=(A=1);
                                /* 5 PRINT OFF */       LISTOPT=(A=A-A);
                        END;
                END;
                /* 5 SIMPLE OP */       DO;
                        A=OPCODE;       CALL EMIT1;
                END;
        END CASECLASS;
        CALL PRINTALINE;
        GOTO NXL;
END;

ZTFCB:PROCEDURE;
        M([TFCB])=(A=A\\A);
        M([HEX TFCB+12])=A;
        M([HEX TFCB+15])=A;
        M([HEX TFCB+32])=A;
END;

[MACRO CPM '0']

OPENR:PROCEDURE;
NOFILEERR:PROCEDURE;
        DE=.'FILE DOES NOT EXIST, ASSEMBLY ABORTED$';
        CALL PRMSG;
GOTO [CPM];
END;
        CALL ZTFCB;
        C=15;   DE=[TFCB];      CALL [BDOS];
        IF (A=A+1) ZERO CALL NOFILEERR;
        IBUFP=(A=128);
        ASMPC=(HL=0);
END;

OPENHEX:PROCEDURE;
CANTCREATEERR:PROCEDURE;
        DE=.'UNABLE TO CREATE .SCC FILE; ABORTING RUN$';
        CALL PRMSG;
GOTO [CPM];
END;

        CALL ZTFCB;
```

ERROR;

```
HL=[HEX TFCB+9];        DE=.'SCC';        /* NEW EXTENSION */
C=3;
REPEAT;
        M(HL)=(A=M(DE));        HL=HL+1;        DE=DE+1;
UNTIL (C=C-1) ZERO;
C=19;   DE=[TFCB];      CALL [BDOS];
C=22;   DE=[TFCB];      CALL [BDOS];
IF (A=A+1) ZERO CALL CANTCREATEERR;

END;

DECLARE LCLSTK(50) BYTE;
SP=.LCLSTK(50);
CALL OPENR;     CALL PASS1;
CALL OPENR;     CALL PASS2;
CALL OPENHEX;   CALL PUTHEX;    CALL CLOSEHEX;
GOTO [CPM];
EOF

A>^L

?

A>
A>ASM874X TEST
UNRECOGNIZED OPERATION IN:                      MOV     R0,X

SYNTAX ERROR IN:
SYNTAX ERROR IN:
SYNTAX ERROR IN:
VALUE TOO LARGE FOR FIELD (TRUNCATED):
0000 03 5F                                      ADD     A,#2*5
UNDEFINED NAME IN EXPRESSION
VALUE TOO LARGE FOR FIELD (TRUNCATED):
0002 B8 81                                      MOV     R0,#B00

A>TYPE TEST
        PRINT   OFF
        ADD     A,#2*5
        MOV     R0,#B00
        MOV     R0,X

A>ED TEST
*#AKE

BREAK "?" AT E
*E

A>ASM874X TEST
UNRECOGNIZED OPERATION IN:                      MOV     R0,X

SYNTAX ERROR IN:
SYNTAX ERROR IN:
SYNTAX ERROR IN:
VALUE TOO LARGE FOR FIELD (TRUNCATED):
0000 03 5F                                      ADD     A,#2*5
UNDEFINED NAME IN EXPRESSION
VALUE TOO LARGE FOR FIELD (TRUNCATED):
0002 B8 81                                      MOV     R0,#B00
0004                                            MOV     R0,X

A>^L
```

Continued from p. 17

Now the computer can display the 31 extra characters by sending codes 81 through 9F hex to the terminal (code 80 will print as a space). All we need now is a way to generate these codes from the terminal itself. Again, the terminal normally generates only a 7-bit code; bit 8 is set according to the value of the BIT8 0/1 switch when sending 8-bit data. As shown in Figure 7, we can instead use the otherwise useless HERE IS key to set bit 8 from the keyboard. The signal at pin 12 of the 74LS04 at location L16 is high whenever HERE IS is depressed. Simply connect this signal the UART bit 8 input (pin 33 of K5) and set the BIT8 0/1 switch to 1. Now bit 8 will be set to 1 only when HERE IS is depressed. Thus, pressing both CTRL and HERE IS will access the extra characters. If you are especially lazy, you can make HERE IS automatically invoke the CTRL function also, by connecting pin 13 of the 74LS02 at L12 to pin 9 of the 74LS04 at L16, and grounding pin 6 of the 74LS04 at L18.

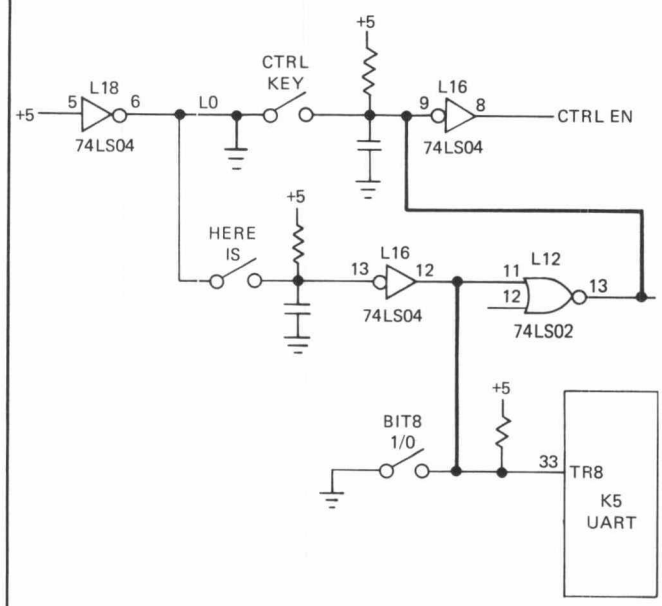


Figure 7 Modifications to set bit 8.

Acknowledgments

Thanks go to Dennis Allison for suggesting that I make these enhancements available to *DDJ*'s readers, and for serving as a guinea pig for the do-it-yourself instructions given above. This article was prepared using my ADM-3A . . . . . (note the use of the repeat key) and custom word-processing software running under Smoke Signal Broadcasting's DOS68 disk operating system for the Southwest Tech 6800 computer system.

A User Interface to ★ APPLE-II ★ Program Renumbering

BY DEWITT S. BROWN
1650 9th Ave. S.E.
Rochester, MN 55901

In DDJ #23, Steve Wozniak detailed the theory and code necessary to renumber BASIC programs on the APPLE-II. Although Wozniak's program was functionally good, I could not use it with the interface provided. In this article I would like to explain my solution. If you have an APPLE-II computer or are a forgetful, lazy user, read on.

The Goal

Renumber lines 100-110, to start at 150, spaced by 10.

Original User Interface

```
>BLOAD RENUMBER.ASM

>Poke 2,150 Mod 256
>Poke 3,150/256
>Poke 4,10 Mod 256
>Poke 5,10/256
>Poke 6,100 Mod 256
>Poke 7,100/256
>Poke 8,110 Mod 256
>Poke 9,110/256
>CALL 776
```

This is too much typing (about 160 keystrokes for this example) and too much remembering (the meaning of four locations in memory and two call addresses) for forgetful, lazy me.

Modified User Interface

An early solution to this problem used variables and an EXEC file to reduce the typing and remembering necessary.

Renumber Data Set Range

```
>NEWSTART = 150
>NEWINC = 10
>RSTART = 100
>REND = 110
>EXEC RENUM.RANGE
```

Renumber Whole Data Set

```
>NEWSTART = 150
>NEWINC = 10
>EXEC RENUM.ALL
```

This approach still requires 60 keystrokes and the remembering of four variable names and two exec file names (because meaningful names are used in place of numbers, the memory part is easier than the original user interface).

Ideal User Interface

Even with the modified user interface, the renumber program was too cumbersome to use often. I, therefore, decided to create the "ideal user interface" to the renumber program. Here are the qualities I set as "ideal."

1. Minimal typing required – only the actual input value should be entered.
2. Only one name to remember – that name would be meaningful to aid in remembering it.
3. Where defaults were used, they would be listed – the actual default would be selected with a simple "return."
4. Some degree of bad typing would be allowed – the user could correct mistakes until "return" entered.

5. Some degree of error checking would be implemented – numbers too big or non-numeric characters should be rejected.
6. Performance and storage impacts to the program being renumbered should be negligible – the renumber program is a tool, not a goal.

Here is the ideal user interface:

```
>EXEC RENUM
Commands in RENUM are
   HIMEM:-27648
   RENUM=-27648
BLOAD RENUM.ASM.A$9400.L$200
   CALL RENUM
```

On a cleared screen, APPLE responds

```
ENTER LINE NUMBER     DEFAULT

NEW START          100 ? 150 (RETURN)
NEW INCREMENT      100 ?  10 (RETURN)
RANGE START          0 ? 100 (RETURN)
RANGE END        65535 ? 110 (RETURN)
100 → 150                  ↑ Entered by user.
103 → 160
106 → 170
107 → 180
109 → 180
110 → 190
```

Subsequent renumbers could be done by typing 'CALL RENUM' to avoid diskette I/O delay (of course, continuing to type 'EXEC RENUM' would work as well – just slower).

This solution requires only 25 keystrokes (15 if all defaults taken) and requires the user to remember 1 easily-remembered name. Quantitatively, this user interface is 6 times better than the original interface.

Evaluation

The goals set down for the ideal user interface were met, except for the impact on storage. Compared with 190 bytes (X'BE') used originally, 512 bytes (X'200') are used. Also, the original version did not take storage from the program area, as this version does.

My solution is break a very large program into pieces to allow room for this version of renumber. The pieces could be connected using the 'chain' command. This solution encourages me to keep program development modular, as advocated in structured programming.

References

1. Auricchio, Rick. "An Apple-II Programmer's Guide," *The Best of MICRO*, Volume 1
2. Synertek. "SY 6500/MCS 6500 Microcomputer Family Programming Manual." Synertek, Santa Clara, CA (May, 1978)
3. Wozniak, Steve. "Renumbering & Appending BASIC Programs on the APPLE-II Computer." *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia,* Number 23 (March, 1978)
4. Wozniak, Steve. "SWEET16: The 6502 Dream Machine" *Byte Magazine,* (November, 1977)


```
1  ********************************************
2  *                                          *
3  * APPLE-II BASIC RENUMBER USER INTERFACE   *
4  *                                          *
5  * THIS ROUTINE INITIALIZES SWEET16 REGISTERS TO *
6  * DEFAULT VALUES, INVOKES THE CHECK/CONVERT *
7  * ROUTINE, AND THEN PASSES CONTROL TO WOZNIAK'S *
8  * SWEET16 RENUMBER ROUTINE.                *
9  *                                          *
10 * THIS ROUTINE IS WRITTEN IN 6502 ASSEMBLER *
11 * CODE.                                    *
12 *                                          *
13 * THIS ROUTINE CALLS THESE SUBROUTINES:    *
14 *   PUTSCRN-WRITE 26 BYTES TO THE SCREEN   *
15 *   CHKCNVT-ACCEPT INPUT, CHECK IT, AND CONVERT *
16 *      IT TO HEXADECIMAL VALUE             *
17 *   WOZRNMB-ORIGINAL WOZNIAK  RENUMBER CODE *
18 *                                          *
19 * THIS ROUTINE RUNS WITH APPLE II DOS REL 3.1 *
20 *                                          *
21 * PATCHED TO ALLOW FOR CHARACTER GENERATED BY DOS* *
22 *                                          *
23 * D. S. BROWN                   6/18/79 *  *
24 *                                          *
25 ********************************************
26
27 * EQUATES USED BY THIS ROUTINE
28
29 PATCHES  EQU  $9565      PATCH AREA TO BE USED
30 *                        TO CORRECT ANY
31 *                        PROBLEMS
32 MONRDCH  EQU  $FD35      ADDRESS OF MONITOR
33 *                        ROUTINE USED TO READ
34 *                        A CHARACTER
35 MONCLR   EQU  $FC58      ADDRESS OF MONITOR
36 *                        ROUTINE USED TO CLEAR
37 *                        THE SCREEN
38 MONCROUT EQU  $FD8E      ADDRESS OF MONITOR
39 *                        ROUTINE USED OUTPUT A
40 *                        CARRIAGE RETURN
41 MONCOUT  EQU  $FDED      ADDRESS OF MONITOR
42 *                        ROUTINE USED OUTPUT
43 *                        ACCUMULATER CHARACTER
44 RLOOUTS  EPZ  $14        LO-BYTE OF REGISTER
45 *                        HOLDING OUTSTRING
46 *                        ADDRESS
47 RHIOUTS  EPZ  $15        HI-BYTE OF REGISTER
48 *                        HOLDING OUTSTRING
49 *                        ADDRESS
50 OUTSLEN  EPZ  $1A        LENGTH OF OUTPUT
51 *                        STRING
52 ALOOUTS  EPZ  $7E        LO-BYTE OF OUTSTRING
53 *                        ADDRESS
54 AHIOUTS  EPZ  $95        HI-BYTE OF OUTSTRING
55 *                        ADDRESS
56 CMAX     EPZ  $FF        CONSTANT FOR MAXIMUM
57 *                        VALUE
58 CMIN     EPZ  $00        CONSTANT FOR MINIMUM
59 *                        VALUE
60 C100     EPZ  $64        CONSTANT FOR DECIMAL
61 *                        100
62 R1LO     EPZ  $02        LO-BYTE OF SWEET16
```

```
                        63  *                           REGISTER 1
                        64  R1HI       EPZ  $03         HI-BYTE OF SWEET16
                        65  *                           REGISTER 1
                        66  R2LO       EPZ  $04         LO-BYTE OF SWEET16
                        67  *                           REGISTER 2
                        68  R2HI       EPZ  $05         HI-BYTE OF SWEET16
                        69  *                           REGISTER 2
                        70  R3LO       EPZ  $06         LO-BYTE OF SWEET16
                        71  *                           REGISTER 3
                        72  R3HI       EPZ  $07         HI-BYTE OF SWEET16
                        73  *                           REGISTER 3
                        74  R4LO       EPZ  $08         LO-BYTE OF SWEET16
                        75  *                           REGISTER 4
                        76  R4HI       EPZ  $09         HI-BYTE OF SWEET16
                        77  *                           REGISTER 4
                        78  OUTRLO     EQU  $9572       LO-BYTE OF OUTPUT
                        79  *                           REGISTER NUMBER
                        80  OUTRHI     EQU  $9573       HI-BYTE OF OUTPUT
                        81  *                           REGISTER NUMBER
                        82  PROMPTC    EPZ  $33         ZERO-PAGE ADDRESS OF
                        83  *                           PROMPT CHARACTER
                        84
9400:  20 65 95         85            JSR  PATCHES      GO TO PATCH AREA
                        86
                        87  ********************************************
                        88  * PUT ADDRESS OF OUTPUT STRINGS INTO SWEET16 *
                        89  * REGISTER A TO BE USED AS A ZERO-PAGE ADDRESS *
                        90  ********************************************
                        91
9403:  A2 95            92            LDX  #AHIOUTS     HI-BYTE OF ADDRESS TO
9405:  86 15            93            STX  RHIOUTS      HI-BYTE OF REGISTER
9407:  A2 7E            94            LDX  #ALOOUTS     LO-BYTE OF ADDRESS TO
9409:  86 14            95            STX  RLOOUTS      LO-BYTE OF REGISTER
                        96
                        97  ********************************************
                        98  * SET DEFAULT VALUES FOR RENUMBER INTO SWEET16 *
                        99  * REGISTERS                                  *
                       100  ********************************************
                       101
940B:  A2 FF            102           LDX  #CMAX        MAXIMUM VALUE TO
940F:  86 09            103           STX  R4HI         SWEET16 REGISTER 4
9411:  86 08            104           STX  R4LO         LO AND HI BYTES
                       105
940D:  86 33            106           STX  PROMPTC      FF IS USED TO SET
                       107  *                           PROMPT CHARACTER TO ?
                       108
9413:  A2 00            109           LDX  #CMIN        MINIMUM VALUE TO
9415:  86 07            110           STX  R3HI         SWEET16 REGISTER 3
9417:  86 06            111           STX  R3LO         LO AND HI BYTES
9419:  86 05            112           STX  R2HI         SWEET16 REGISTER 2 HI
                       113  *                           BYTE
941B:  86 03            114           STX  R1HI         SWEET16 REGISTER 1 HI
                       115  *                           BYTE
                       116
941D:  8E 72 95         117           STX  OUTRLO       INITIAL REGISTER FOR
9420:  8E 73 95         118           STX  OUTRHI       OUTPUT IS REGISTER 0
                       119
9423:  A2 64            120           LDX  #C100        PUT DECIMAL 100 INTO
9425:  86 02            121           STX  R1LO         REGISTER 1 LO-BYTE
9427:  86 04            122           STX  R2LO         REGISTER 2 LO-BYTE
                       123
                       124  ********************************************
                       125  * PUT HEADER LINE ON SCREEN USING OUTPUT     *
                       126  * SUBROUTINE                                 *
                       127  ********************************************
                       128
9429:  20 3B 94        129           JSR  PUTSCRN      PUT HEADER TO SCREEN
                       130
                       131  ********************************************
                       132  * CALL CHECK/CONVERT ROUTINE FOR EACH SWEET16 *
                       133  * REGISTER TO BE SET UP FOR RENUMBER         *
                       134  ********************************************
                       135
942C:  20 4B 94        136           JSR  CHKCNVT      NEW START LINE NUMBER
942F:  20 4B 94        137           JSR  CHKCNVT      NEW INCREMENT
9432:  20 4B 94        138           JSR  CHKCNVT      START OF RANGE
9435:  20 4B 94        139           JSR  CHKCNVT      END OF RANGE
                       140
                       141  ********************************************
                       142  * REGISTERS ARE SET UP FOR RENUMBER BY WOZNIAK, *
                       143  * GO DO IT. (NO RETURN TO THIS CODE)         *
                       144  ********************************************
                       145
9438:  4C AF 94        146           JMP  WOZRNMB      FINALLY RENUMBER
                       147  ********************************************
                       148  * THIS SUBROUTINE PUTS LINES TO THE SCREEN BY *
                       149  * USING MONITOR ROUTINE WHICH PRINTS ACCUMULATER *
                       150  ********************************************
                       151
943B:  20 8E FD        152  PUTSCRN   JSR  MONCROUT    CARRIAGE RETURN OUT
943E:  A0 00           153           LDY  #CMIN        INITIALIZE INDEX
9440:  B1 14           154  LOOP2     LDA  (RLOOUTS),Y  OUT-BYTE TO ACCUM.
9442:  20 ED FD        155           JSR  MONCOUT      ACCUMULATER OUT
9445:  C8              156           INY               INDEX TO NEXT BYTE
9446:  C0 1A           157           CPY  #OUTSLEN     COMPARE WITH MAX=26
9448:  30 F6           158           BMI  LOOP2        IF NOT MAX, CONTINUE
944A:  60              159           RTS               ELSE, RETURN
                       160  ********************************************
                       161  *                                          *
                       162  * APPLE-II BASIC RENUMBER USER INTERFACE   *
                       163  *                                          *
                       164  * THIS SUBROUTINE PUTS A MESSAGE TO THE SCREEN *
                       165  * FROM THE OUTPUT STRING STORAGE AREA.  THIS *
                       166  * OUTPUT IS A PROMPT FOR THE USER TO ENTER DATA. *
                       167  *                                          *
                       168  * THE USER RESPONSE IS ENTERED, THEN CHECKED FOR *
                       169  * LENGTH.  LENGTH=0 IS A NULL RETURN; LENGTH>5 *
                       170  * IS AN ERROR.                             *
                       171  *                                          *
                       172  * IF ALL IS OK WITH INPUT LENGTH, THE SWEET16 *
                       173  * CODE IS ENTERED TO DO FURTHER CHECKING OF THE *
                       174  * INPUT AND CONVERSION TO HEXADECIMAL      *
                       175  *                                          *
                       176  * D. S. BROWN                    6/18/79   *
                       177  *                                          *
                       178  ********************************************
                       179
                       180  * EQUATES USED BY THIS SUBROUTINE
                       181
                       182  RPWR16    EPZ  $8           REGISTER TO POINT TO
                       183  *                           POWERS OF 16 VECTOR
```

```
                        184  APWR16     EQU  $9574      ADDRESS OF POWERS OF
                        185  *                         16 VECTOR
                        186  RINP       EPZ  $6        REGISTER POINTING TO
                        187  *                         INPUT BYTE
                        188  AINP       EQU  $0200     ADDRESS OF INPUT AREA
                        189  RTOTAL     EPZ  $C        REGISTER WHICH HOLDS
                        190  *                         TOTAL NOW CONVERTED
                        191  ZERO       EQU  $0000     ZERO CONSTANT
                        192  RINLEN     EPZ  $5        UPON INVOCATION, THE
                        193  *                         LENGTH OF THE INPUT
                        194  *                         FIELD IS IN THIS
                        195  *                         REGISTER
                        196  R1PWR16    EPZ  $7        REGISTER HOLDS THE
                        197  *                         CURRENT POWER OF 16
                        198  INMASK     EQU  $00B0     MASK TO CONVERT ASCII
                        199  *                         BYTE TO DECIMAL
                        200  INCHECK    EQU  $000A     INPUT BYTE MUST BE IN
                        201  *                         RANGE FROM 0000 TO
                        202  *                         0009 AFTER CONVERSION
                        203  *                         FROM ASCII TO DECIMAL
                        204  RINCHK     EPZ  $B        REGISTER USED TO
                        205  *                         CHECK INPUT
                        206  RINBYTE    EPZ  $9        REGISTER USED TO HOLD
                        207  *                         ONE CONVERTED INPUT
                        208  *                         BYTE FOR ADD INDEX
                        209  MONERR     EQU  $FF2D     ADDRESS OF MONITOR
                        210  *                         ERROR ROUTINE
                        211  MONSW16    EQU  $F689     ADDRESS OF MONITOR
                        212  *                         SWEET16 INTERPRETER
                        213  MONINP     EPU  $FD6A     ADDRESS OF MONITOR
                        214  *                         INPUT ROUTINE
                        215  NULLLRET   EPZ  $00       NULL RETURN LENGTH
                        216  TOOLONG    EPZ  $06       TOO LONG LENGTH
                        217  REGLEN     EPZ  $02       REGISTER LENGTH
                        218  RSVD1      EPZ  $0A       SWEET16 REGISTER A IS
                        219  *                         RESERVED FOR USE BY
                        220  *                         OTHER ROUTINE
                        221  SW16ACC    EPZ  $00       SWEET16 ACCUMULATER
                        222  OUTADD     EQU  $9572     ADDRESS OF OUTPUT
                        223  *                         ADDRESS
                        224  ROUTADD    EPZ  $02       REGISTER USED TO
                        225  *                         ADDRESS OUTPUT AREA
                        226  R5LO       EPZ  $0A       LO-BYTE OF SWEET16
                        227  *                         REGISTER 5
                        228  R5HI       EPZ  $0B       HI-BYTE OF SWEET16
                        229  *                         REGISTER 5
                        230
                        231
944B: A5 14             232  CHKCNVT    LDA  RLOOUTS   GET OUTSTRING ADDRESS
944D: 18                233               CLC         PREPARE FOR ADD
944E: 69 1A             234               ADC  #OUTSLEN INDEX TO THIS STRING
9450: 85 14             235               STA  RLOOUTS  SAVE NEW ADDRESS
9452: 20 3B 94          236               JSR  PUTSCRN  PUT OUT OUTPUT STRING
9455: 20 6A FD          237               JSR  MONINP   GET INPUT
9458: E0 00             238               CPX  #NULLRET CHECK FOR NULL RETURN
945A: D0 0A             239               BNE  NOTNULL  NE => NOTNULL
945C: AD 72 95          240               LDA  OUTRLO   GET OUT REGISTER PTR
945F: 18                241               CLC         PREPARE FOR ADD
9460: 69 02             242               ADC  #REGLEN  INCREMENT REG LENGTH
9462: 8D 72 95          243               STA  OUTRLO   SAVE NEW OUT REG PTR
9465: 60                244               RTS         RETURN TO CALLER

9466: E0 06             245  NOTNULL    CPX  #TOOLONG  CHECK FOR TOO LONG
9468: 10 2D             246               BPL  ERR6502  PL => ERR6502
946A: 86 0A             247               STX  $R5LO    STORE X IN R5 LO-BYTE
946C: A2 00             248               LDX  #CMIN    GET CONSTANT OF 0
946E: 86 0B             249               STX  $R5HI    STORE 0 IN R5 HI-BYTE
                        250  ***************************************************
                        251  *                                                 *
                        252  * APPLE-II BASIC RENUMBER USER INTERFACE          *
                        253  *                                                 *
                        254  * THIS SUBROUTINE ACCEPTS INPUT OF LENGTH X AT    *
                        255  * LOCATION 0200 AND CONVERTS IT TO HEXADECIMAL    *
                        256  * REPRESENTATION.                                 *
                        257  *                                                 *
                        258  * THE HEXADEXIMAL OUTPUT IS STORED AT ADDRESS+2,  *
                        259  * WHERE 'ADDRESS' IS THE VALUE STORED AT 9572/3.  *
                        260  *                                                 *
                        261  * THIS PUTS THE OUTPUT IN THE APPROPRIATE         *
                        262  * SWEET16 REGISTERS FOR THE WOZNIAK RENUMBER      *
                        263  * ROUTINE.                                        *
                        264  *                                                 *
                        265  * THIS ROUTINE IS WRITTEN IN SWEET16 INTERPRETER  *
                        266  * CODE.                                           *
                        267  *                                                 *
                        268  * SWEET16 REGISTER USAGE:                         *
                        269  *    0 - ACCUMULATER                              *
                        270  *    1 - NEW START LINE NUMBER                    *
                        271  *    2 - NEW LINE NUMBER INCREMENT                *
                        272  *    3 - RANGE START                              *
                        273  *    4 - RANGE END                                *
                        274  *    5 - LENGTH OF INPUT                          *
                        275  *    6 - ADDRESS OF INPUT                         *
                        276  *    7 - HOLDS A POWER OF 16                      *
                        277  *    8 - HOLDS POWERS OF 16 VECTOR ADDRESS        *
                        278  *    9 - INPUT BYTE USED AS AN INDEX              *
                        279  *    A - RESERVED FOR USE BY PRINT ROUTINE        *
                        280  *    B - USED TO CHECK INPUT BYTE                 *
                        281  *    C - HOLDS TOTAL ACCUMULATED HEX DIGITS       *
                        282  *                                                 *
                        283  * D. S. BROWN                          6/18/79  *
                        284  *                                                 *
                        285  ***************************************************
9470: 20 89 F6          286               JSR  MONSW16  START SWEET16 CODE
9473: 18 74 95          287               SET  RPWR16,APWR16 GET ADDRESS OF THE
                        288  *                         POWERS OF 16 VECTOR
                        289  *                         INTO REGISTER
9476: 16 00 02          290               SET  RINP,AINP GET ADDRESS OF START
                        291  *                         OF INPUT AREA INTO
                        292  *                         REGISTER
9479: 1C 00 00          293               SET  RTOTAL,ZERO INITIALIZE TOTAL
947C: 25                294               LD   RINLEN   INPUT LENGTH TO ACC.
947D: A6                295               ADD  RINP     THE START OF THE
                        296  *                         INPUT AREA + THE
                        297  *                         LENGTH OF THE INPUT -
                        298  *                         1 IS THE RIGHTMOST
                        299  *                         BYTE OF THE INPUT
947E: 36                300               ST   RINP     STORE ADDRESS IN
                        301  *                         REGISTER TO GET INPUT
947F: 68                302  LOOP1      LDD  @RPWR16  GET POWER OF 16 AND
                        303  *                         DECREMENT REGISTER
9480: 37                304               ST   R1PWR16  PUT POWER OF 16 INTO
                        305  *                         REGISTER FOR ADD
```

```
9481:  86        306             POP   @RINP         GET BYTE OF INPUT AND
                 307  *                              DECREMENT REGISTER
9482:  1B B0 00  308             SET   RINCHK,INMASK PUT MASK TO CHECK
                 309  *                              INPUT INTO REGISTER
9485:  BB        310             SUB   RINCHK        SUBTRACT MASK FROM
                 311  *                              INPUT BYTE
9486:  05 0E     312             BM    ERR           MINUS => NOT NUMERIC
9488:  39        313             ST    RINBYTE       SAVE CONVERTED INPUT
9489:  1B 0A 00  314             SET   RINCHK,INCHECK PUT MAXIMUM DECIMAL
                 315  *                              NUMBER INTO REGISTER
948C:  BB        316             SUB   RINCHK        SUBTRACT MAXIMUM FROM
                 317  *                              ACTUAL
948D:  04 07     318             BP    ERR           PLUS => NOT NUMERIC
948F:  2C        319             LD    RTOTAL        GET CURRENT TOTAL
9490:  F9        320  ADDPWR     DCR   RINBYTE       DECREMENT ADDS-TO-DO
9491:  08 0A     321             BM1   NEXTIN        M1 => NO MORE TO ADD
9493:  A7        322             ADD   R1PWR16       ADD POWER OF 16 TO
                 323  *                              TOTAL
9494:  02 FA     324             BNC   ADDPWR        CARRY => INPUT NUMBER
                 325  *                              GREATER THAN 65535
9496:  00        326  ERR        RTN                 EXIT SWEET16 CODE
                 327
9497:  68        328  ERR6502    PLA                 GET SUBROUTINE RETURN
9498:  68        329             PLA                 ADDRESS OFF STACK
9499:  20 2D FF  330             JSR   MONERR        TELL USER OF ERROR
                 331  *                              USING MONITOR CODE
949C:  60        332             RTS                 RETURN TO USER
949D:  3C        333  NEXTIN     ST    RTOTAL        SAVE TOTAL
949E:  F5        334             DCR   RINLEN        ONE MORE DIGIT
                 335  *                              CONVERTED TO DECIMAL
949F:  07 DE     336             BNZ   LOOP1         NZ => INPUT LEFT
94A1:  18 72 95  337             SET   ROUTADD,OUTADD GET ADDRESS OF
                 338  *                              OUTPUT REGISTER
94A4:  68        339             LDD   @ROUTADD      GET OUTPUT REGISTER
                 340  *                              VALUE
94A5:  E0        341             INR   SW16ACC       INCREMENT TO NEXT
94A6:  E0        342             INR   SW16ACC       REGISTER BY ADDING 2
94A7:  F8        343             DCR   ROUTADD       RESTORE ADDRESS OF
94A8:  F8        344             DCR   ROUTADD       OUTPUT REGISTER
94A9:  78        345             STD   @ROURTAD      SAVE NEW OUTPUT
                 346  *                              REGISTER VALUE
94AA:  38        347             ST    ROUTADD       STORE REGISTER NUMBER
                 348  *                              INTO REGISTER
94AB:  2C        349             LD    RTOTAL        GET TOTAL AGAIN
94AC:  78        350             STD   @ROUTADD      PUT TOTAL INTO
                 351  *                              SWEET16 REGISTER
94AD:  00        352  RTN        RTN                 EXIT SWEET16
94AE:  60        353             RTS                 RETURN TO CALLER
```

```
354   TITLE 'APPLE-II BASIC RENUMBER/APPEND SUBROUTINES'
355  ************************
356  *                      *
357  *   APPLE-II BASIC     *
358  *      RENUMBER        *
359  *     SUBROUTINES      *
360  *                      *
361  *       RENUMBER       *
362  *  NEW INITIAL (2,3)   *
363  *  NEW INCR (4,5)      *
364  *  RANGE BEG (6,7)     *
365  *  RANGE END (8,9)     *
366  *                      *
367  *   WOZ       11/16/77 *
368  * APPLE COMPUTER INC.  *
369  *                      *
370  ************************
371
372    SUBTTL 6502 EQUATES
373  ROL         EPZ   $0          LOW-ORDER SW16 BYTE
374  ROH         EPZ   $1            HI-ORDER.
375  R11L        EPZ   $16         LOW-ORDER SW16 R11
376  *                             BYTE
377  R11H        EPZ   $17           HI-ORDER.
378  HIMEM       EPZ   $4C         BASIC HIMEM POINTER.
379  PVL         EPZ   $CC         BASIC VAR POINTER.
380  MEMFULL     EQU   $E36B       BASIC MEM FULL ERROR.
381  PRDEC       EQU   $E51B       BASOC DECIMAL PRINT
382  *                             SUBROUTINE.
383  RANGERR     EQU   $EE68       BASIC RANGE ERROR.
384  SW16        EQU   $F689       SWEET16 ENTRY.
385  CROUT       EQU   $FD8E       CAR RET SUBR.
386  COUT        EQU   $FDED       CHAR OUT SUBR.
387
388    SUBTTL SWEET16 EQUATES
389  ACC         EPZ   $0          SWEET16 ACCUMULATOR.
390  NEWLOW      EPZ   $1          NEW INITIAL LNO.
391  NEWINCR     EPZ   $2          NEW LNO INCR.
392  LNLOW       EPZ   $3          LOW LNO OF RENUM
393  *                             RANGE.
394  LNHI        EPZ   $4          HI LNO OF RENUM RANGE
395  TBLSTRT     EPZ   $5          LNO TABLE START
396  TBLNDX1     EPZ   $6          PASS 1 LNO TBL INDEX.
397  TBLIM       EPZ   $7          LNO TABLE LIMIT.
398  SCR8        EPZ   $8          SCRATCH REG.
399  HMEM        EPZ   $8          HIMEM (END OF PRGM).
400  SCR9        EPZ   $9          SCRATCH REG.
401  PRGNDX      EPZ   $9          PASS 1 PROG INDEX.
402  PRGNDX1     EPZ   $A            ALSO PROG INDEX.
403  NEWLN       EPZ   $B          NEXT 'NEW LNO'.
404  NEWLN1      EPZ   $C          PRIOR 'NEW LNO'
405  *                             ASSIGNMENT.
406  TBLND       EPZ   $6          PASS 2 LNO TABLE END.
407  PRGNDX2     EPZ   $7          PASS 2 PROG INDEX.
408  CHR0        EPZ   $9          ASCII '0'.
409  CHRA        EPZ   $A          ASCII 'A'.
410  MODE        EPZ   $C          CONST/LNO MODE.
411  TBLNDX2     EPZ   $B          LNO TBL IDX FOR
412  *                             UPDATE.
413  OLDLN       EPZ   $D          OLD LNO FOR UPDATE.
414  STRCON      EPZ   $B          BASIC STR CON TOKEN.
```

Continued on p. 50

```
                        415  REM       EPZ  $C          BASIC REM TOKEN.
                        416  R13       EPZ  $D          SWEET16 REG 13 (CPR).
                        417  THEN      EPZ  $D          BASIC THEN TOKEN
                        418  LIST      EPZ  $D          BASIC LIST TOKEN
                        419
                        420   SUBTTL APPLE-II BASIC RENUMBER SUBROUTINE- PASS 1
94AF: 20 89 F6          421  WOZRNMB   JSR  SW16
94B2: 18 4C 00          422            SET  SCR8,HIMEM
94B5: 68                423            LDD  @SCR8       BASIC HIMEM POINTER
94B6: 38                424            ST   HMEM          TO HMEM.
94B7: 19 CE 00          425            SET  SCR9,PVL+2
94BA: C9                426            POPD @SCR9       BASIC VAR PTR TO
94BB: 35                427            ST   TBLSTRT       TBLSTRT AND TBLNDX1
94BC: 36                428            ST   TBLNDX1
94BD: 21                429            LD   NEWLOW      COPY NEWLOW (INITIAL)
94BE: 3B                430            ST   NEWLN         TO NEWLN.
94BF: 3C                431            ST   NEWLN1
94B0: C9                432            POPD @SCR9       BASIC PROG PTR
94C1: 37                433            ST   TBLIM          TO TBLIM
94C2: 39                434            ST   PRGNDX         AND PRGNDX.
94C3: 29                435  PASS1     LD   PRGNDX
94C4: D8                436            CPR  HMEM        IF PRGNDX >= HMEM
94C5: 03 46             437            BC   PASS2         THEN PASS 1 DONE.
94C7: 3A                438            ST   PRGNDX1
94C8: 26                439            LD   TBLNDX1
94C9: E0                440            INR  ACC         IF < 2 BYTES AVAIL IN
94CA: D7                441            CPR TBLIM          LNO TABLE THEN RET
94CB: 03 38             442            BC   MERR          WITH 'MEM FULL' MSG
94CD: 4A                443            LD   @PRGNDX1
94CE: A9                444            ADD  PRGNDX      ADD LEN BYTE TO
94CF: 39                445            ST   PRGNDX        PROG INDEX.
94D0: 6A                446            LDD  @PRGNDX1    LINE NUMBER.
94D1: D3                447            CPR  LNLOW       IF < LNLOW THEN
94D2: 02 2A             448            BNC  P1B           GO TO P1B.
94D4: D4                449            CPR  LNHI        IF > LNHI THEN
94D5: 02 02             450            BNC  P1A           GO TO P1C.
94D7: 07 30             451            BNZ  P1C
94D9: 76                452  P1A       STD  @TBLNDX1    ADD TO LNO TABLE.
94DA: 00                453            RTN
94DB: A5 01             454            LDA  R0H         *** 6502 CODE ***
94DD: A6 00             455            LDX  R0L
94DF: 20 1B E5          456            JSR  PRDEC       PRINT OLD LNO'->'NEW
94E2: A9 AD             457            LDA  #$AD          (R0,R11) IN DECIMAL
94E4: 20 ED FD          458            JSR  COUT
94E7: A9 BE             459            LDA  #$BE
94E8: 20 ED FD          460            JSR  COUT
94EC: A5 17             461            LDA  R11H
94EE: A6 16             462            LDX  R11L
94F0: 20 1B E5          463            JSR  PRDEC
94F3: 20 8E FD          464            JSR  CROUT
94F6: 20 8C F6          465            JSR  SW16+3      *** END 6502 CODE ***
94F9: 2B                466            LD   NEWLN
94FA: 3C                467            ST   NEWLN1      COPY NEWLN TO NEWLN1
94FB: A2                468            ADD  NEWINCR       AND INCR NEWLN BY
94FC: 3B                469            ST   NEWLN         NEWINCR.
94FD: 0D                470            NUL              (WILL SKIP NEXT INST)
94FE: D1                471  P1B       CPR  NEWLOW      IF LOW LNO < NEWLOW
94FF: 02 C2             472            BNC  PASS1         THEN RANGE ERR.
9501: 00                473  RERR      RTN              PRINT 'RANGE ERR' MSG
9502: 4C 63 EE          474            JMP  RANGERR       AND RETURN.
9505: 00                475  MERR      RTN              PRINT 'MEM FULL' MSG
9506: 4C 6B E3          476            JMP  MEMFULL       AND RETURN.
9509: EC                477  P1C       INR  NEWLN1      IF HI LNO <= MOST
                        478  *                             RECENT NEWLN
950A: DC                479            CPR  NEWLN1        THEN RANGE ERROR
950B: 02 F4             480            BNC  RERR
950D: 19 B0 00          481  PASS2     SET  CHR0,$B0    ASCII '0'.
9510: 1A C1 00          482            SET  CHRA,$C1    ASCII 'A'.
9513: 27                483  P2A       LD   PRGNDX2
9514: D8                484            CPR  HMEM        IF PROG INDEX = HIMEM
9515: 03 4C             485            BC   DONE          THEN DONE PASS 2.
9517: E7                486            INR  PRGNDX2     SKIP LEN BYTE.
9518: 67                487            LDD  @PRGNDX2    LINE NUMBER.
9519: 3D                488  UPDATE    ST   OLDLN       SAVE OLD LNO.
951A: 25                489            LD   TBLSTRT
951B: 3B                490            ST   TBLNDX2     INIT LNO TABLE INDEX.
951C: 21                491            LD   NEWLOW      INIT NEWLN1 TO NEWLOW
951D: 1C 00 00          492            SET  NEWLN1,0    (WILL SKIP NEXT 2
                        493  *                             INSTRUCTIONS)
                        494            ORG  *-2
951E: 2C                495  UD2       LD   NEWLN1
951F: A2                496            ADD  NEWINCR     ADD INCR TO NEWLN1
9520: 3C                497            ST   NEWLN1
9521: 2B                498            LD   TBLNDX2     IF LNO TBL IDX=TBLND
9522: B6                499            SUB  TBLND         THEN DONE SCANNING
9523: 03 07             500            BC   UD3           LNO TABLE.
9525: 6B                501            LDD  @TBLNDX2    NEXT LNO FROM LNO TBL
9526: BD                502            SUB  OLDLN       LOOP TO UD2 IF NOT
9527: 07 F5             503            BNZ  UD2           SAME AS OLDLN.
9529: C7                504            POPD @PRGNDX2    REPLACE OLD LNO WITH
952A: 2C                505            LD   NEWLN1        CORRESPOND NEW LNO
952B: 77                506            STD  @PRGNDX2
952C: 1B 28 00          507  UD3       SET  STRCON,$28  STR CON TOKEN
952F: 1C 00 00          508            SET  MODE.0      (SKIPS NEXT 2 INSTRS)
                        509            ORG  *-2
9530: 67                510  GOTCON    LDD  @PRGNDX2
9531: FC                511            DCR  MODE        IF MODE = 0 THEN
9533: 08 E5             512            BMI  UPDATE        UPDATE LNO REF.
9534: 47                513  ITEM      LD   @PRGNDX2    BASIC ITEM.
9535: D9                514            CPR  CHR0
9536: 02 09             515            BNC  CHKTOK      CHECK TOKEN FOR SPEC
9538: DA                516            CPR  CHRA        IF >='0' AND <'A'
9539: 02 F5             517            BNC  GOTCON        THEN SKIP CONSTANT
                        518  *                             OR UPDATE
953B: F7                519  SKPASC    DCR  PRGNDX2
953C: 67                520            LDD  @PRGNDX2    SKIP ALL NEG BYTES OF
953D: 05 FC             521            BM   SKPASC        STR CON, REM OR NAM
953F: F7                522            DCR  PRGNDX2
9540: 47                523            LD   @PRGNDX2
9541: DB                524  CHKTOK    CPR  STRCON      STR CON TOKEN?
9542: 06 F7             525            BZ   SKPASC        YES,SKIP SUBSEQNCE
9544: 1C 5D 00          526            SET  REM,$5D
9547: DC                527            CPR  REM         REM TOKEN?
9548: 06 F1             528            BZ   SKPASC        YES,SKIP SUBSEQNCE
954A: 08 13             529            BMI  CONTST      GOSUB, LOOK FOR LNO.
954C: FD                530            DCR  R13
954D: FD                531            DCR  R13         (TOKEN $5F IS GOTO)
954E: 06 0F             532            BZ   CONTST      THEN LNO,LOOK FOR LNO
9550: 1D 24 00          533            SET  THEN,$24
9553: DD                534            CPR  THEN
9554: 06 09             535            BZ   CONTST      THEN LNO,LOOK FOR LNO
9556: F0                536            DCR  ACC
```

(Continued on page 33)

CONVERTING BINARY TO DECIMAL

BY DENNIS ALLISON
169 Spruce Street
Atherton, CA 94025

Dennis R. Allison is an independent consultant and part-time lecturer at Stanford University's Computer Systems Laboratory. His research interests include programming languages and their relationship to problem solving and creativity. He is presently President and Chairman of the Board of Trustees of the People's Computer Company. He is a member of ACM, SIAM, and IEEE and serves as an Associate Technical Editor of the IEEE Computer Society's general membership publication, Computer Magazine.

Every system needs to have some mechanism for outputting data in a form easily useable by humans. With our ten-finger heritage, the easy-to-generate hex, octal, and binary displays are inadequate, particularly for non-specialists. This note describes a conversion routine suitable for use with microprocessors as it does not use either multiplication or division. For a more comprehensive treatment see Knuth's *Seminumerical Algorithms* (Addison-Wesley, Reading, MA, 1969).

There are several slightly different problems which have to be solved. For a decimal output to a terminal, we normally want a zero-suppressed output to have the number 2 print a "2" and not "00002". On the other hand, if the output is to go to a LED display which normally outputs hex, the conversion must be to packed decimal, not characters, and should not be zero suppressed. The idea is pretty much the same for each, however.

The routine depends upon being able to compute both the quotient and the remainder after dividing by 10, is the rightmost digit. And n/10 contains all the digits to the right. This observation suggests the routine

```
prdec   (   n ) int n. {
            int q, r; /* locals for quotient and remainder */
            if (n < 0) { putchar('-'); n = -n; }
            q = n/10;
            r = n % 10;
            if (q != 0) prdec(q); /* print digits to the left */
            putchar(r + '0');
}
```

written in the programming language C, which performs a binary to decimal conversion with zero suppression. The procedure is recursive, so local variables are allocated on the stack. The digits are constructed right to left and printed left to right. The subroutine stack is used to reverse the converted number. Signed numbers are converted to unsigned numbers following output of a sign.

If one does not want to suppress zeros, one can add another parameter to the call, the number of digits desired. If one does that the program becomes

```
prdec   (   n, dig) int n,dig; {
            int q,r; /*locals for quotient and remainder */
            if (dig <= 0) return;
            if (n < 0) { putchar('-'); n = -n; }
            q = n/10;
            r = n % 10;
            prdec(q, dig-1); /* print digits to left */
            putchar(r + '0');
}
```

If one wants a packed decimal representation for positive numbers only, as would be the case for a LED display, the call to character output routine, putchar, should be replaced with a call to a packing routine which simply masks off the decimal digit and shifts it left four bits. One way to do this would be to have a global 16-bit variable and use a C routine of the form

```
pack ( dig ) int dig; {
            packed = (packed < <4) & (dig & 017 ));
```

Unfortunately both these routines fail for the largest possible negative number because of the way the PDP-11 handles the remaindering process. The division must be unsigned for the algorithm to work. Remember that 2's complement numbers have an asymmetric range skewed to the negative side. For example, 2's complement 4-bit numbers represent values from –8 to +7. And negating the largest possible negative number gives the largest possible negative number.

If we are going to implement this procedure, we need a way to do the division and remaindering operation. Strangely enough, we do the division using multiplication by 0.1. The fraction 1/10 has a binary representation

1/10 =
0.0000110011001100110011001100110011001100 . . .

which may be thought of as a sum of fractions

1/10 = 1/16 + 1/32 + 1/256 + 1/512 + . . .

The fraction will never represent the correct value. If we truncate the representation at any point it will always be too small. If alpha is an approximation to 0.1 which is too small by 1/w, one can write

(alpha + 1/w) * u = u/10

which suggests that

alpha * u = u/10 – u/w

Now the error in the representation, 1/w will always be less than or equal to the smallest value represented in the fraction alpha. The value of the ratio u/w will always be less than or equal to one. Thus, we can write a division by ten routine which also computes the remainder and uses only shifts and adds. Using the C notation of >> for right shift one has

```
int rem; /* global to catch value of remainder */
div10 ( x ) int x; {
   int u,v;
   u=(x>>1) + (x>>2) + (x>>5) + (x>>6) + (x>>9) +
      (x>>10) + (x>>13) + (x>>14);
   u = >>3;
   v = mul10(u);
   if ( (rem = x – v) > = 10) {u++; rem = rem – 10;}
   return (u);
}
```

This program uses right shifts to implement division so it won't work for negative numbers. The same problem with the largest negative number will also appear unless the right shift is implemented as a logical right shift rather than an arithmetic right shift which sign extends.

The second to the last line corrects the quotient and remainder when the value computed is in error by one.

One now needs to have a simple multiplication by ten routine. This can be done with a few temps and additions. In C the routine looks like

```
mul10(x) int x; {
   int t;
   x = x + x; t = x + x; t = t + t;
   return (t+x);
}
```

which works by noting that 10 * x = 8 * x + 2 * x and using addition to do the multiplication by two.

Mapping these routines into machine language is left as an exercise for the reader. The only problem is how to represent local variables in the recursive procedures. For most microprocessors local variables may be implemented on the subroutine stack. At subroutine entry the stack pointer is decremented by the number of bytes needed for local variables. Locals are then accessed by indexing from the stack pointer. Just prior to subroutine return the stack pointer must be incremented to readjust it so the return address is at the top of stack.

And, of course, there are equivalent non-recursive programs which use exactly the same ideas but keep their own stacks. They require slightly more code and data space but may run faster.

(continued from page 31)

```
9557:  06 BA     537          BZ   P2A          EOL (TOKEN $01)?
9559:  1D 74 00  538          SET  LIST,$74
955C:  BD        539          SUB  LIST         SET MODE = 0 IF LIST
955D:  09 01     540          BNMI CONTS2         OR LIST COMMA ($73)
955F:  B0        541  CONTST  SUB  ACC          CLEAR MODE FOR LNO
9560:  3C        542  CONTS2  ST   MODE           UPDATE CHECK.
9561:  01 D1     543          BR   ITEM         CHECK NEXT BASIC ITEM
9563:  00        544  DONE    RTN               RETURN.
9564:  60        545          RTS
                 546  *
                 547  * PATCH AREA: LABEL IS PATCHES
                 548  *
9565:  20 35 FD  549          JSR  MONRDCH      CLEAR EXTRA INPUT
                 550  *                         CHARACTER FROM EXEC
9568:  20 FC 58  551          JSR  MONCLR       CLEAR THE SCREEN
956B:  60        552          RTS               RETURN.
                 553  *
                 554  * CONSTANTS AND DATA AREAS
                 555  *
                 556          ORG  $9572
9572:  00 00     557  OUTREG  HEX  00,00        OUTPUT REGISTER NUMBER
 574:  01 00     558  PWR16   HEX  01,00        16**0
9576:  0A 00     559          HEX  0A,00        16**1
9578:  64 00     560          HEX  64,00        16**2
957A:  E8 03     561          HEX  E8,03        16**3
957C:  10 27     562          HEX  10,27        16**4
957E:  C5 CE D4  563  OUTCONS CHAR 'ENTER LINE NUMBER DEFAULT'
9597:  8D        564          HEX  8D           CARRIAGE RETURN
9598:  E0 E0 CE  565          CHAR '  NEW START      100   '
95B2:  E0 E0 CE  566          CHAR '  NEW INCREMENT  100   '
95CC:  E0 E0 D2  567          CHAR '  RANGE START      0   '
95E6:  E0 E0 D2  568          CHAR '  RANGE END     65535   '
```

UCSD PASCAL Terminal Parameters

BY RONALD G. PARSONS
9001 Laurel Grove Dr.
Austin, TX 78758

The PASCAL system from the University of California at San Diego has a very useful system generation procedure by which the physical parameters of the user's screen and terminal may be defined to the system. These parameters include the keyboard cursor keys, the screen cursor movement characters, and such entities as the characters to clear the screen and erase to the end of the line. These parameters are defined for the system by using a supplied program SETUP and are stored on a file named SYSTEM.MISCINFO.

In order to write programs which are as independent as possible of the user's screen and keyboard configuration, these parameters must be known to the program. For example, if you wish to write a program which must be able to move the cursor around on the screen and always stay inside the user's screen whether it is 64 X 16 or 80 X 24, the cursor controls and screen size must be known. The record definition and initialization procedure below may be used to make these parameters known to a user-written program. The main procedure illustrates the use of the parameters in the record. There is only one oddly defined parameter—"verticalmdelay"— which tells the number of nulls to be sent after a vertical cursor move. This variable is defined here as character, but should be used as an integer. The statement

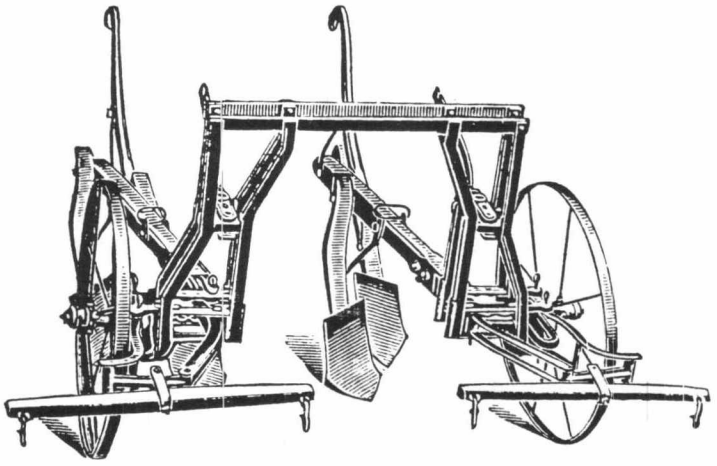
```
intverticalmdelay := ord(verticalmdelay);
```

will transform the quantity to an integer.

I hope the way UCSD PASCAL lets the user define his system parameters to the operating system will be widely copied. I wish UCSD had been consistent in their use of these parameters in setting the length of their prompt lines which often overflow a 64 character screen line.

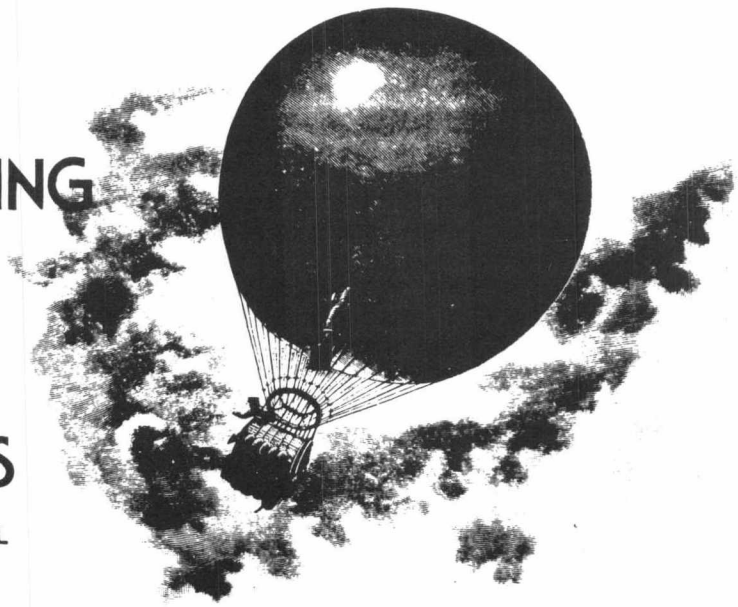
```
(*$S+*)
Program testmiscinfo;
(*************************************************************)
(*                                                           *)
(*      SYSTEM.MISCINFO parameter record definition          *)
(*      Author: Ronald G. Parsons                            *)
(*      Date: August 21, 1979                                *)
(*                                                           *)
(* Defines terminal and screen parameters from the file      *)
(* produced by SETUP for UCSD Pascal Version I.5.            *)
(*                                                           *)
(*                                                           *)
(* The following variables are associated with               *)
(* the SETUP defined parameter.                              *)
(* The full name of these variables is minfobuffer.xxxxx     *)
(* e.g.,                                                     *)
(*       minfobuffer.screenheight                            *)
(* or                                                        *)
(*       with minfobuffer do ... screenheight ... end;       *)
(*                                                           *)
(* ----- The following variables are type Boolean.           *)
(* hasclock            HAS CLOCK                             *)
(* has8510A            HAS 8501A                             *)
(* haslowercase        HAS LOWER CASE                        *)
(* hasrandom           HAS RANDOM CURSOR ADDRESSING          *)
(* hasslow             HAS SLOW TERMINAL                     *)
(* student             STUDENT                               *)
(* premcup             PREFIX [MOVE CURSOR UP]               *)
(* premcright          PREFIX [MOVE CURSOR RIGHT]            *)
(* preereol            PREFIX [ERASE TO END OF LINE]         *)
(* preereos            PREFIX [ERASE TO END OF SCREEN]       *)
(* premchome           PREFIX [MOVE CURSOR HOME]             *)
(* predcharacter       PREFIX [DELETE CHARACTER]             *)
(* preerscreen         PREFIX [ERASE SCREEN]                 *)
(* preerline           PREFIX [ERASE LINE]                   *)
(* prekmcright         PREFIX [KEY FOR MOVING CURSOR RIGHT]  *)
(* prekmcleft          PREFIX [KEY FOR MOVING CURSOR LEFT]   *)
(* prekmcup            PREFIX [KEY FOR MOVING CURSOR UP]     *)
(* prekmcdown          PREFIX [KEY FOR MOVING CURSOR DOWN]   *)
(* prenonprintchar     PREFIX [NON-PRINTING CHARACTER]       *)
(* prekeystop          PREFIX [KEY FOR STOP]                 *)
(* prekeybreak         PREFIX [KEY FOR BREAK]                *)
(* prekeyflush         PREFIX [KEY FOR FLUSH]                *)
(* prekeyendfile       PREFIX [KEY TO END FILE]              *)
(* preedescapekey      PREFIX [EDITOR 'ESCAPE' KEY]          *)
(* prekeydline         PREFIX [KEY TO DELETE LINE]           *)
(* prekeydcharacter    PREFIX [KEY TO DELETE CHARACTER]      *)
(* preedacceptkey      PREFIX [EDITOR 'ACCEPT' KEY]          *)
(*                                                           *)
(* ----- The following variables are type integer.           *)
(* screenheight        SCREEN HEIGHT                         *)
(* screenwidth         SCREEN WIDTH                          *)
(*                                                           *)
```

```
(* ----- The following variables are type char.              *)
(* leadintoscreen       LEAD-IN TO SCREEN                    *)
(* movecursorhome       MOVE CURSOR HOME                     *)
(* eraseeos             ERASE TO END OF SCREEN               *)
(* eraseeol             ERASE TO END OF LINE                 *)
(* movecright           MOVE CURSOR RIGHT                    *)
(* movecup              MOVE CURSOR UP                       *)
(* backspace            BACKSPACE                            *)
(* verticalmdelay       VERTICAL MOVE DELAY                  *)
(* eraseline            ERASE LINE                           *)
(* erasescreen          ERASE SCREEN                         *)
(* keytomcup            KEY TO MOVE CURSOR UP                *)
(* keytomcdown          KEY TO MOVE CURSOR DOWN              *)
(* keytomcleft          KEY TO MOVE CURSOR LEFT              *)
(* keytomcright         KEY TO MOVE CURSOR RIGHT             *)
(* keytoendfile         KEY TO END FILE                      *)
(* keyforflush          KEY FOR FLUSH                        *)
(* keyforbreak          KEY FOR BREAK                        *)
(* keyforstop           KEY FOR STOP                         *)
(* keytodcharacter      KEY TO DELETE CHARACTER              *)
(* nonprintchar         NON-PRINTING CHARACTER               *)
(* keytodline           KEY TO DELETE LINE                   *)
(* editorescapekey      EDITOR 'ESCAPE' KEY                  *)
(* leadincharacter      LEAD-IN CHARACTER FROM KEYBOARD      *)
(* editoracceptkey      EDITOR 'ACCEPT' KEY                  *)
(*                                                           *)
(*************************************************************)

type
   miscinforec = packed record
      dummy1 : packed array [0..28] of integer;
      dummy2,dummy3,student,hasslow,hasrandom,haslowercase,
      has8510A,hasclock : Boolean;
      dummy4 : integer;
      erasescreen,eraseline,verticalmdelay,backspace,
         movecup,movecright,eraseeol,eraseeos,movecursorhome,
         leadintoscreen : char;
      preerline,preerscreen,predcharacter,premchome,
         preereos,preereol,premcright,premcup : Boolean;
      screenwidth,screenheight : integer;
      editoracceptkey,leadincharacter,editorescapekey,
         keytodline,nonprintchar,keytodcharacter,
         keyforstop,keyforbreak,keyforflush,keytoendfile,
         keytomcright,keytomcleft,keytomcdown,keytomcup : char;
      dummy5 : integer;
      dummy6,dummy7,preedacceptkey,prekeydcharacter,
         prekeydline,preedescapekey,prekeyendfile,
         prekeyflush,prekeybreak,prekeystop,prenonprintchar,
         prekmcdown,prekmcup,prekmcleft,prekmcright : Boolean;
      dummy8 : packed array [48..95] of integer;
      end;

var
   minfobuffer : miscinforec;
procedure minfoinit; (* initialize *)
var
   minfofile : file of miscinforec;
begin
   reset (minfofile,'*SYSTEM.MISCINFO');
   get (minfofile);
   minfobuffer := minfofile ;
   close (minfofile);
end;

begin
   minfoinit;
   with minfobuffer do
   begin
      writeln('Lead-in to screen = ',ord(leadintoscreen));
      writeln('Cursor home = ',ord(movecursorhome));
      writeln('Erase to end-of-screen = ',ord(eraseeos));
      writeln('Erase to end-of-line = ',ord(eraseeol));
      writeln('Move cursor right = ',ord(movecright));
      writeln('Move cursor up = ',ord(movecup));
      writeln('Key to end file = ',ord(keytoendfile));
      writeln('Key to move cursor up = ',ord(keytomcup));
      write('Has clock = ');
      if hasclock then writeln('T') else writeln('F');
      write('Has 8510A = ');
      if has8510A then writeln('T') else writeln('F');
      write('Has lowercase = ');
      if haslowercase then writeln('T') else writeln('F');
      write('Has random = ');
      if hasrandom then writeln('T') else writeln('F');
      write('Has slow = ');
      if hasslow then writeln('T') else writeln('F');
      write('Student = ');
      if student then writeln('T') else writeln('F');
      write('PRE [key to move cursor up] = ');
      if prekmcup then writeln('T') else writeln('F');
      writeln;
      writeln(screenheight,' ',screenwidth);
      gotoxy((screenwidth div 2),(screenheight div 2));
      (* go to center of screen *)
      write('Input "return" to clear screen');
      readln;
      write(erasescreen);
   end;
end.
```



PROGRAMMING PASTIMES AND PLEASURES

BY CHARLES WETHERELL



© 1979, Charles Wetherell

Four Famous Programs

Programs are not always interesting because of their output; sometimes the program itself is the object of interest. This month I am going to describe four famous programs which fall into this class. Even if you don't actually write versions for your own computer, thinking through each program will surely reward you with deeper understanding of your computer.

I Got Plenty of Nothin'

The first problem is to clear memory to zero. Easy enough, you say—just hit the CLEAR button. But the rules are a little more complicated. All of writable memory must be set to zero at program completion; as an added fillip, all registers should be zero as well. The program must execute normally without using any input/output device and without any operator intervention. If possible, the program should halt normally. As an example, I know of one computer which used a store double word zero instruction to wipe out the last two words of a loop through memory. An even harder problem is to store a preselected bit pattern throughout memory. You may find this program is not possible for some bit patterns.

Stamina

Now consider the problem of writing the longest running program you can. The same rules apply—no I/O, no operator intervention, normal program execution. How about infinite loops? Well, my definition of a loop may be a little different than yours. I will say the program loops when the same main memory configuration occurs for the second time. The repetition must be exact, bit by bit. To make things simple, you may specify any starting memory configuration you like. Your program must reside in main memory and count as part of the memory pattern. Strangely enough, almost every computer has the same solution for this problem; I will suggest it at the end of the column.

Introspection

For the third problem, consider a program which can describe itself. Specifically, the introspective program must print out an exact listing of its source coding. You are not allowed to use any tricks. For example, you may not reread the source input. Some compilers and interpreters leave copies of source lying around in memory. You may not reach in and print such a source. The only resource you are allowed is the program itself. A memory dump does not constitute a source listing.

The introspective program is particularly interesting because it is always possible. In any language—FORTRAN, BASIC, PASCAL, assembly—as long as the language is "sufficiently" powerful—the program will exist. Every normal programming language is powerful enough; see the column on Ackermann's Function (*DDJ* # 37) to find out how powerful is "powerful." The existence is a consequence of the Recursion Theorem. Fortunately, you can solve this problem without using any mathematics. Hint: the solution hinges on the proper use of quotation.

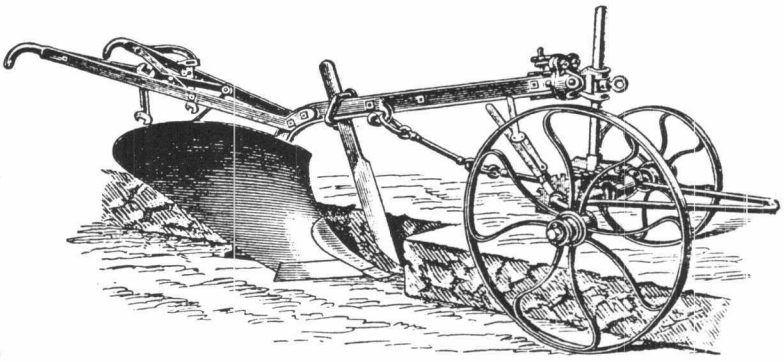
A Pox On You

The last program is the computer virus. Before you do any work on viruses, you better be sure that you either have a "clean room" or can stand to take the consequences. If you are certain, I'll go on.

The virus is a program which implants itself into another program, subverts the host's operations, creates multiple copies of itself, and then implants the copies into other victim programs. Obviously, once a virus gets loose in a system, the system will soon spend all its time breeding more viruses. If the fever attacks some common utility like the compiler or file manager, the plague may kill the system immediately. Eradication of a successful virus could be very difficult.

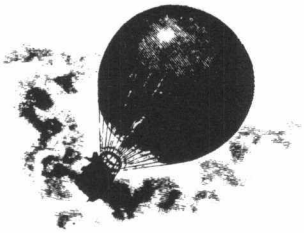
I know of one rumored virus. Back in the early days of computing, the 7094 operating system stored all object programs waiting to be punched onto a single tape. The virus would read the last object file off the tape, modify it to include a copy of the virus, and then write the object back onto the tape. When the object, which would be a single routine from some other program, was loaded and executed, the virus would propagate again. Indeed, since object decks could be physically loaded with several programs, the virus might be propagated by the unsuspecting owner of the subroutine—a sort of Typhoid Mary. Security features on most modern systems would defeat this ploy, but it is thinking about such pathologies that makes designers improve security. I might add that releasing a virus would certainly be unethical, if not criminal.

One other rumored virus caused a genetic defect in its host. The creator of a timesharing system also had a hand in writing the compiler for the language in which the entire system was written. He modified the compiler so that whenever the SIGN-ON program was compiled, the compiled SIGN-ON procedure would always recognize his name. This way he could run on any version of the system anywhere. Not only that, but if this defect was removed from the compiler source code, the compiler automatically reinserted the flaw in the compiler object modules when the old (flawed) compiler—the only one available, of course—was used to recompile the new (clean?!) compiler sources. I heard this tale far removed from the perpetrator and I suspect it to be apocryphal. But have you checked you compiler lately?



Hint

The longest running program generally treats all of memory as a giant binary counter and simply increments constantly by 1. When the counter overflows, all memory configurations (almost) will have been counted off. A sophisticated version will also move the program around in memory to get more configurations (only one move is necessary—why?). Don't try this program and expect to see it finish; on a computer with n bits of memory, the program will run for at least 2^n or so cycles.



A MICROPROCESSOR-ASSISTED GRAPHICS SYSTEM

BY GRIFFITH HAMLIN, JR. AND THOMAS CROCKETT
Los Alamos Scientific Laboratory
Los Alamos, New Mexico 87544

ABSTRACT

GRAF80 is a small scale graphics system designed to demonstrate some of the capabilities offered by microprocessor-assisted graphics. As such, it employs both hardware and software techniques. The system implements in a microprocessor a subset of the features of the SIGGRAPH-ACM CORE (1) graphics package, including picture segmentation and a subset of the output primitives and primitive attributes.

INTRODUCTION

GRAF80 is a graphics system that uses inexpensive microprocessors to enhance the capabilities of a Tektronix storage tube display. For a fraction of the cost of a Tektronix 4014, several features can be added that considerably increase its usefulness. These features include a selective erase using the segmented display file concept of the SIGGRAPH CORE graphics package, and the ability to display a limited number of vectors in refresh mode using a high-bandwidth hardware interface between a microprocessor and the Tektronix 4014.

HARDWARE

The hardware consists of an IMSAI 8080 microprocessor, which forms the heart of the system; a Tektronix 4014 direct-view storage tube terminal, which is used as the graphical output device; and a host computer. The IMSAI 8080 acts as a dedicated device driver, and has software that supports picture refreshing and segmentation. Any one of several other available 8-bit microprocessors could be used as well. Several possibilities exist for interconnecting the microprocessor, Tektronix terminal, and host computer. Figure 1 shows one possibility that uses commonly available equipment.

The Tektronix 4041 has a built-in serial interface that will support serial communications up to 9600 baud with the microprocessor. This rate should be sufficient to provide interactive access to images stored in the microprocessor's segmented display file. It is not sufficient to refresh images using the write-through mode of the 4014. To remedy this, we have constructed a simple parallel interface to the 4014 and connected it as shown in Figure 2. This interface consists of only nine 16-pin integrated circuits mounted and wire-wrapped on a blank Tektronix plug-in card. This provides one-way byte parallel transmission at the Tektronix 4014 MINIBUS® speed (9 μs/byte) from the microprocessor to the Tektronix 4014. Communication from the Tektronix 4014 uses the normal serial interface in the 4014 and is connected directly to the host computer.

Any machine capable of producing GRAF80 commands may be the host computer. This could be the same or another microprocessor, a minicomputer, or a large mainframe. In our case a PRIME-300 minicomputer is used.

© 1978, SIGGRAPH-ACM. Reprinted by permission.

SOFTWARE

Software for GRAF80 consists of two main components, one on each computer. The microprocessor component receives graphic commands, stores them in a segmented data structure, and interprets them to produce either a refreshed or a stored image on the 4014. The host computer software consists of a package of Fortran subroutines that issue the commands.

Microprocessor Software

Software for the microprocessor is written in PL/M, a high-level language designed for microprocessors (2). The software is designed to intercept and interpret a set of primitive graphical commands. It requires 4000 bytes of microprocessor memory plus buffers for the segmented display data structure generated from the commands. These commands consist of move and draw graphic-output primitives, along with commands to OPEN, CLOSE, DISPLAY, and ERASE individual image segments. The complete list is given in Appendix A. They were selected from the SIGGRAPH-ACM CORE graphics package.

The main program consists of a loop that continuously monitors an I/O subtask and a graphics subtask. (The term "task" is used to indicate that these two processes could be executed in parallel if properly synchronized, although they have been implemented sequentially here. The net effect would be similar in either case.) When the main loop determines that a particular task has work to be done, it transfers control to that task, which runs to completion and returns to the main loop.

Input to the microprocessor is interrupt driven. A simple interrupt routine receives a character and stores it in a circular buffer. The job of the I/O task is to empty the buffer and evaluate its contents. In order to allow ordinary (nongraphical or graphical) use of the terminal, a "transparent" mode is defined. Transparent mode defaults until a special graphics notify character (ASCII ETB) is received. The I/O task examines each character and sends it directly to the terminal until an ETB is encountered, at which time further data is considered to be graphical. The graphical input is filtered according to a predefined transmission format (Appendix B) and stored as a picture segment. When the input buffer is emptied, control returns to the main loop.

The graphics task serves as an interpreter, converting the commands stored by the I/O task into device-dependent instructions which are sent to the 4014 to generate the picture. By redefining this task, different devices may be handled within the framework of a single system. Each call to this device driver interprets a single picture segment; repeated calls with the same segment allow the refresh capability. The graphics task returns to the main loop after once interpreting each segment that needs to be displayed.

Picture segmentation is handled at a level above the device driver. A number of graphics commands are used for manipulating segments. The I/O task intercepts these and takes appropriate action, such as setting flags, allocating storage, etc. The actual picture commands (or "primitives") are stored in the microprocessor's free memory as one or more 64-byte blocks, linked together by a forward chain, one chain per segment. A set of memory management routines is available to manipulate this data structure. Memory allocation is automatic and dynamic, implying that each segment receives only as much memory as it requires. The total amount of information that may be displayed simultaneously is thus limited by the amount of memory available to the microprocessor. Although the current implementation uses only real memory, this need not be the case; nonrefreshed segments could use an auxiliary disk, but system performance would be somewhat degraded. Currently up to 16 segments may be defined.

Commands for manipulating segments are listed below.

OPEN(N,R)	Opens segment N. All subsequent graphic primitives become part of the segment until a matching CLOSE is encountered. R = refresh status; 0 = no refresh; 1 = refresh.
CLOSE(N)	Closes segment N. No further information may be added to this segment.
DISPLAY(N)	Causes segment N to be displayed.
ERASE(N)	Removes segment N from the display, but the segment may be redisplayed later.
DELETE(N)	Deletes segment from microprocessor memory.

Open segments may not be nested or overlapping; no more than one segment may be open at any time. No portion of a segment is visible until an explicit DISPLAY command is given, and a currently open segment cannot be displayed. This differs from the SIGGRAPH-ACM CORE system which specifies that portions of the open segment become immediately visible as they are defined. Due to the existence of both stored and refreshed segments in our implementation, we departed from the SIGGRAPH CORE here in order to considerably simplify the microprocessor software.

Erasing a segment merely removes it from the display; it remains available in the microprocessor memory for further use. A DELETE command destroys the segment entirely.

Refreshing, as pointed out, is accomplished by the graphics task; successive iterations of the main program loop refresh the segment. Since the I/O task usually is not busy, the graphics task becomes the limiting factor in providing flicker-free refresh. Within that task two considerations limit the refresh capacity:

1. the rate at which graphic commands can be interpreted into Tektronix code by the microprocessor,
2. the drawing speed of the Tektronix 4014 (about 190 μs/in. which is rather slow for refresh purposes).

I/O time is negligible because of the parallel interface. Currently the flicker-free refresh capacity is approximately 75 characters or 65 inches of 2-inch vectors (including both dark and written vectors). This is about half the capacity of the Tektronix 4014 and is limited in our current implementation by the microprocessor software. Different software, discussed in the last section, should remedy this situation.

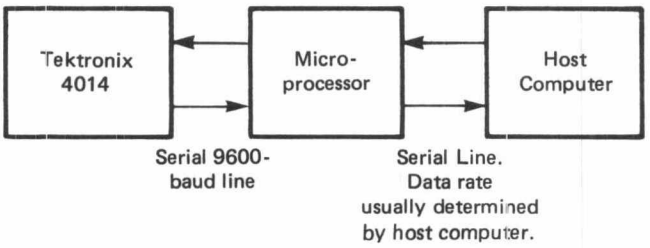


Fig. 1. Basic hardware configuration.

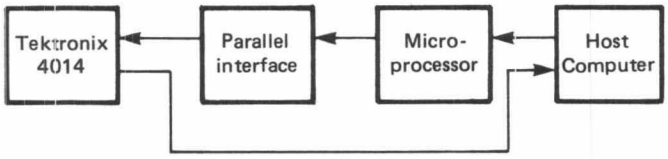


Fig. 2. GRAF80 hardware configuration.

Nonrefresh segments differ in that they are drawn only once, and not on each successive loop iteration. Normally such segments will be drawn in storage mode, but this does not have to be true. The primitive attribute REFRESH determines the mode the Tektronix 4014 uses when generating characters and vectors; therefore it is possible to draw non-refreshed information that immediately disappears and to draw storage-mode information that is continually refreshed (not recommended). The general assumption, however, is that refreshed segments will contain refresh commands and non-refresh segments will contain storage commands.

Selective erase is implemented somewhat differently for each type of segment. Refreshed segments merely have their display flag turned off. Nonrefresh (storage) segments that are erased cause the entire display to be erased, and the display flag for that segment is turned off; all other segments are flagged as not having been drawn. The result is that everything but the erased segment is redrawn on the next pass of the main loop. The redraw time is very short; the only objection being the screen erase flash that precedes it. A typical mesh plot containing about 1500 ¼-inch vectors takes about one second to redraw.

Host Software

A package of Fortran interface subroutines on our PRIME-300 minicomputer allows a user's application program to generate commands for the GRAF80 system. Access is allowed to each segment-level and primitive-level command. The subroutines generate the proper transmission formats, perform data conversions where necessary, and send GRAF80 commands out the communications line to the microprocessor. All coordinates used at this level are integers, which run from 0-4095. This corresponds directly to Tektronix 4014 enhanced graphics screen coordinates, although any arbitrary units could have been chosen. Appendix C contains documentation of these routines.

CONCLUSIONS

From the host computer's viewpoint the system appears as a virtual device, with built-in functions for picture segmentation and refreshing. As such, it would be most useful when interfaced to a higher-level graphics system. This higher-level system could be an existing graphics package, modified to accept GRAF80 as another device type and to include segment manipulation features, or it could be a new system built on top of GRAF80. In the latter case, an implementation of the SIGGRAPH-ACM CORE system would be facilitated since GRAF80 has several of the CORE functions already available.

Our 8080 microprocessor is only lightly loaded with the current GRAF80 software (except when refreshing). Thus we believe that much of the CORE system could be handled by the microprocessor with additional software. Currently we have designed, but not yet implemented on our microprocessor, the image transformation features of the CORE system and most of the input functions. We chose these functions for microprocessor implementation since they can provide fast interactive feedback to the user when placed on a processor with a high-speed connection to the display device. Also, they involve only fixed point coordinates. Today's microprocessors handle floating point computations only slowly in software. We have left the transformations from floating point (world coordinates) to fixed point (normalized device coordinates) to our minicomputer which has floating point instructions.

As mentioned above, the limiting refreshing rate for GRAF80 could be improved through revision of the data structures. The graphics task could be modified to store Tektronix 4014 code in a separate buffer, rather than display it directly. A third refresh task would then display this buffer, with no interpretive overhead. A small test program on our microprocessor that used this technique was able to refresh approximately 140 inches of vectors, about twice as much as GRAF80 can currently refresh. However, our experience so far has been that the segmenting features (which provide selective erase) are more valuable for most users than the refresh capability.

The handful of users with engineering applications who have been exposed to GRAF80 have generally been quite pleased with its capabilities. This is in part due to their previous use of an overloaded mainframe (a not uncommon occurrence) with a slow (300 to 2400 baud) data rate line to their terminal. These conditions favor our use of local intelligence and sending only changes in images instead of sending the entire image each time a small change is made.

REFERENCES

1. Computer Graphics (11, 3) Fall 1977.
2. *A Guide to PL/M Programming*, Intel Corp., Santa Clara, California, 1973.

Letters

AN ADDITION TO 8080 PILOT

I have added a "pause" command to my implementation of the pilot interpreter (see *DDJ* #13 and #14). The command "PAUSE:" delays approximately 1 second before the program execution continues. "PAUSE:N" delays N seconds, and N may be a number or numeric variable (similar to the INMAX: command.

What makes the command useful is that the delay is interrupted by the pressing of any key and the program then continues with the Y/N switch set for "N" and a new variable "[" (shift-K) set to the number of seconds that expired before the key was pressed. If the delay is not interrupted however, the Y/N switch is set to "Y".

Sincerely,
Robert Wilcox

920 N. Washington St.
Owosso, MI 48867

AN EXAMPLE WILL SHOW HOW THE COMMAND CAN BE USED:

```
R:EXAMPLE OF A PILOT PROGRAM USING THE 'PAUSE:' COMMAND
*BEGIN
T:WHAT IS YOUR NAME?
T:YOU HAVE 5 SECONDS TO ANSWER.
PAUSE:5
JY:*TOOSLOW
A:$N
T:YOU ANSWERED "$N" AFTER A #[ SECOND DELAY.
J:*ASK
*TOOSLOW
T:YOUR 5 SECONDS ARE UP!
*ASK T:WANT TO TRY AGAIN?
A:
M:Y
JY:BEGIN
E:
```

THE ROUTINE TO BE ADDED TO PILOT IN ORDER TO IMPLEMENT THE PAUSE: COMMAND FOLLOWS, ALONG WITH THE MODIFICATIONS MADE TO THE "LETTER" AND "INITV" ROUTINES THAT ARE NECESSARY IN ORDER TO ADD THE "[" VARIABLE.

ALSO, THE COMMAND MUST BE ADDED TO "CTLST":

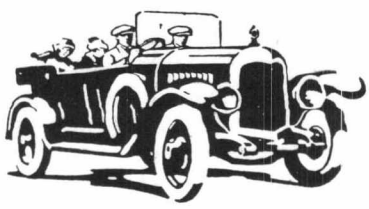
```
        DB      'PAUSE',ØDH
        DW      PAUSE
```

AND THE NVAR STORAGE AREA MUST BE CHANGED TO:

```
NVAR:   DS      55
```

I HOPE THAT THIS ADDITION TO PILOT WILL BE USEFUL AND I WOULD BE GLAD TO HEAR FROM ANYONE USING IT.

```
;****PAUSE OPERATION ROUTINE****
;
;PAUSE OPERATION - 'PAUSE:N' DELAYS N SEC.
;IF ANY KEY IS PRESSED BEFORE THE DELAY TIME
;HAS EXPIRED, THE Y/N SWITCH IS SET TO 'N' AND
;THE VARIABLE '[' (SHIFT-K) IS SET TO THE
;NUMBER OF SECONDS USED UP AND THE DELAY IS ENDED.
;
;IF THE DELAY EXPIRES BEFORE A KEY IS PRESSED,
;THE Y/N SWITCH IS SET TO 'Y', AND '[' IS LEFT = Ø.
;
;
PAUSE:  MVI     D,Ø
        CALL    LDVAR   ;SET VARIABLE '[' TO Ø
        CALL    NMCTL
        CPI     CR
        JZ      PZ1
        ANA     A
        JNZ     DELAY
PZ1:    MVI     E,1
DELAY:  XRA     A
        STA     YNSW    ;SET FOR 'N'=NOT TIMED OUT
        MOV     D,A
        MOV     A,E
        ADD     A
        MOV     E,A
DELA1:  LXI     B,Ø
DELA2:  DCR     C
        JNZ     DELA2
        DCR     B
        JNZ     DELA2
        INR     D
        CALL    CONSCK  ;SEE IF A KEY WAS PRESSED
        JNZ     STOP
        DCR     E
        JNZ     DELA1
        ORA     H
        STA     YNSW    ;'Y'=TIMED OUT
        RET
;
;SET VARIABLE '[' FOR NUMBER OF SECONDS BEFORE
;A KEY WAS PRESSED.
;
STOP:   MOV     A,D
        RAR
        MOV     D,A
LDVAR:  MVI     B,'['
        CALL    VARMCH
        INX     H
        MOV     M,D
        RET
;
;CHECK FOR A KEY PRESSED
;
;INSERT HERE A ROUTINE WHICH TESTS TO SEE IF A KEY WAS
;PRESSED.  IT SHOULD RETURN WITH A Ø IN THE ACCUMULATOR
;IF NO KEY WAS PRESSED, OR A 1 IF A KEY WAS PRESSED.
;ALSO THE ZERO FLAG SHOULD BE TESTED.
;FOR EXAMPLE:
;
CONSCK: IN      PORT    ;CHECK STATUS PORT
        ANI     1       ;MASK STATUS BIT
        RET
;
LETTER TESTS WHETHER CHAR IS A-Z OR [
;
LETTER: MOV     A,M
        MOV     B,M
        CPI     'A'
        JM      NOTL
        CPI     'Z'+2
        JP      NOTL
        XRA     A
        RET
NOTL:   ORA     H
        RET
;
;
;INITIALIZE NUMERIC VARIABLES A-Z AND [ TO ZERO.
;
INITV:  LXI     H,NVAR
        MVI     B,'A'
NV:     MOV     A,B
        CPI     'Z'+2
        RZ
        MOV     M,A
        INX     H
        MVI     M,Ø
        INX     H
        INR     B
        JMP     NV
;
```



A BUG IN SAM76's CP/M

Dear Suzanne:

Several months ago I submitted a text written in the SAM76 language to Claude Kagen at SAM76 inc. It emulated the integer multiplication function available in the language but did floating point multiplication.

Later while attempting to impress a friend a bug in the text became evident. The text failed in certain cases when the number base of the interpreter was larger than ten.

Not realizing it might be used by SAM76 inc. I did not bother to inform them of the problem.

A while ago I received the latest version of SAM76 which is configured for CP/M. Included on the diskette were several files with example texts in them. One of the files is the floating point multiplication text that is in error.

I would be very pleased if you would publish a correction to this Text as it has been distributed for several months in this form.

Thanks,
C.R. Jamieson

92 Granville Cres.
Sherwood Park
Alberta, Canada

This is the original Text—

```
< mult >
%dt,n1,%rs,[1]//`
%dt,n2,%rs,[2]//`
&pt,n1,./&pt,n2,./`
%dt,n3,0123456789ABCDEF/`
%dt,n3,%fdc,n3,%anb///`
%dt,n4,%mu,%rs,%n1//,%rs,%n2//,!%rr,[3]////`
%dt,n5,%rs,%n4///`
%dt,n5,%hc,%fts,n5,%n3////`
%dt,n3,%ad,%hc,%fe,n1//,%hc,%fe,n2////`
%dt,n4,%ig,%n5/,%n3/,!%n4//,!%ps,-%n3/,0,%n4/////`
%fdc,n4,%su,%n5/,%n3///.%fe,n4/`
%et,n1,n2,n3,n4,n5/
```

Correct version—

```
< MULT >
%dt,1,%rs,[1]//%dt,2,%rs,[2]//`
&pt,1,./&pt,2,./`
%dt,3,0123456789ABCDEF/`
%dt,3,%fdc,3,%anb///`
%dt,4,%mu,%rs,%1//,%rs,%2//,!%rr,[3]////`
%dt,5,%rs,%4///%dt,6,%anb//%cnb,10/`
%dt,5,%hc,%fts,5,%3////`
%fdc,4,%su,%hc,%4//,%5///`
%dt,3,%ad,%hc,%fe,1//,%hc,%fe,2////`
%dt,4,%ig,%5/,%3/,!%4//,!%ps,-%3/,0,%4/////`
%fdc,4,%su,%5/,%3///.%fe,4/%cnb,%6//`
%et,1,2,3,4,5,6/
```

Examples of use

```
%MULT,5.,-6./=-30.

%MULT,.007,7./=.049

%MULT,String -.007,7./=String -.049

%MULT,
123456789.123456789,987654321.987654321/=
121932631356500531.347203169112635269

%dt,SQR,!%MULT,*,*///=

%pt,SQR,*/=

%SQR,2453.567/=6019991.023489

%cnb,16/=

%SQR,2453.567/=5278B70.A9CF71
```

CORRECTION

Dear *DDJ*:

Jim Warner of the University of California, Santa Cruz was kind enough to inform me of an error in the UCSD Pascal to CP/M transfer program which was published in *DDJ* #37. The error occurs infrequently when a DLE-space code crosses a 512 byte block boundary.

The enclosed partial listing replaces the original code from 3OF hex to 3AD hex (with 16 bytes left over).

Sincerely,
Ronald G. Parsons

9001 Laurel Grove Dr.
Austin, TX 78758

```
                ;
                ; This was rewritten by J. Warner, UC Santa Cruz.
                ; The GETBYT routine is a new single place to get
                ; characters from the Pascal read buffer.
                ;
030F =          RB      EQU     $               ;read byte
030F 3AC704             LDA     TXTFLG
0312 FE02               CPI     2               ;.TEXT?
0314 C25103             JNZ     NOT$TXT
0317 3ACB04     RBCKLF  LDA     NLF             ;need LF?
031A B7                 ORA     A
031B CA2503             JZ      RBCKTB
031E AF                 XRA     A
031F 32CB04             STA     NLF
0322 3E0A               MVI     A,LF
0324 C9                 RET
0325 3ACC04     RBCKTB  LDA     NTB             ;need tab?
0328 B7                 ORA     A
0329 CA3303             JZ      RBFB
032C 3D                 DCR     A
032D 32CC04             STA     NTB
0330 3E20               MVI     A,' '
0332 C9                 RET
0333 CD5503     RBFB    CALL    GETBYT  ;read byte from buffer
0336 B7                 ORA     A       ;bypass 0's at end of 1024 byte
0337 CA3303             JZ      RBFB                                page
033A FE0D               CPI     CR
033C C24303             JNZ     CKDLE
033F 32CB04             STA     NLF     ;Need Line Feed
0342 C9                 RET
                ;
0343 FE10       CKDLE   CPI     DLE
0345 C0                 RNZ
0346 CD5503             CALL    GETBYT  ;get the space code
0349 D620               SUI     32              ;set indentation
034B 32CC04             STA     NTB
034E C32503             JMP     RBCKTB
                ;
0351 CD5503     NOT$TXT CALL    GETBYT
0354 C9                 RET
                ;
                ; Get the next character from the Pascal file.
                ; Read disk if necessary.
                ;
0355 =          GETBYT  EQU     $
0355 2ACD04             LHLD    BUFADD
0358 110012             LXI     D,BLKBUF+512
035B CDB902             CALL    EQUAL
035E CC6703             CZ      RBLK    ;return with HL --  BLKBUF
0361 7E                 MOV     A,M
0362 23                 INX     H
0363 22CD04             SHLD    BUFADD
0366 C9                 RET
                ;
                ;
0367 =          RBLK    EQU     $               ;read LSN block
0367 2ACF04             LHLD    LSN
036A EB                 XCHG
036B 2AD104             LHLD    LSTLSN
036E CDB902             CALL    EQUAL
0371 CA9803             JZ      SETEOF          ;found last sector
0374 0E01               MVI     C,1             ;select B disk
0376 CD1BBA             CALL    SELDSK
0379 110010             LXI     D,BLKBUF        ;read four sectors
037C CD8502             CALL    RDSEC
037F 118010             LXI     D,BLKBUF+80H
0382 CD8502             CALL    RDSEC
0385 110011             LXI     D,BLKBUF+100H
0388 CD8502             CALL    RDSEC
038B 118011             LXI     D,BLKBUF+180H
038E CD8502             CALL    RDSEC
0391 210010             LXI     H,BLKBUF
0394 22CD04             SHLD    BUFADD
0397 C9                 RET
                ;
0398 21CA04     SETEOF  LXI     H,EOFFLAG
039B 3601               MVI     M,1
039D C9                 RET
                ;
```

THE SIMPLER PLEASURES

Dear *DDJ*,

You published a short article of mine in *DDJ* #34 entitled "Increase Your Poly 88 Pleasure." It described how to combine two or more BASIC programs. Since I wrote that article, I have discovered a much simpler way of combining programs. Enclosed are the newer instructions. I hope you can find the space to print this new information.

Thank you,
David Larry Johnson

4104 Montreal Ave.
Prince George, VA 23875

Combining Poly 88 A00 BASIC Programs

The key to combining A00 BASIC programs is in manipulating two "pointers." At locations 205D–205E is a pointer which points to the beginning of free memory space after the end of BASIC. The pointer is in normal 8080 reversed byte format. A second pointer at 48AA–48AB points at the next available byte after the end of the current program in memory. After a SCRatch these two pointers point at locations adjacent to each other. With this knowledge it is very simple to combine programs by following these steps:

1. Using front panel mode examine 205D–205E and write down the contents. This pointer value will be needed later. (May be 49B0.)
2. LOAD the first program.
3. LIST the program to determine the line number of the last line in the program. Write this down also.
4. Using front panel mode change the contents of 205D–205E to the contents of 48AA–48AB *minus one*. What you are doing here is moving the beginning-of-memory pointer up next to the next-available-byte pointer. Now cold start BASIC with a front panel jump to location 2000. Now your first program is still in memory, but BASIC's pointers are positioned after it so as far as BASIC is concerned, it does not exist.
5. Now LOAD the next program. RENumber it so its *first* line number is greater than the last line number of the first program (the number you wrote down in step 3).
6. Using front panel mode change the contents of 205D–205E back to their initial values that you wrote down in step 1.
7. And finally, *warm start* BASIC by using a front panel jump to location 2003.

Now if you type LIST you will see that both programs are there. You can now SAVE the combined programs or RUN them or do anything else with them that you could do with either one by itself. To RUN the second program you must type RUN XXXX where XXXX is the first line number of the second program. You can add a short routine at the beginning of the first program which asks which program to run and then jumps to the correct one.

There are some potential problem areas associated with joining programs this way:

1. Unless you type RUN to execute each program each time, then variables commmon to both programs may not get initialized properly.
2. If you have DATA statements in both programs, a RESTORE will always restore to the first DATA statement in the first program. Use RESTORE referenced to a specific line number to avoid this problem.
3. If you have defined functions with the same name in both programs, you will have to change all the names of the function in one or the other of the two programs.
4. If there is not STOP at the end of the first program, you may "fall through" into the second program when you do not want to.

This technique for joining programs is quicker to do than to read these instructions. It works very well too. It is not necessary to stop at joining only two programs. You can continue adding programs until you run out of memory or out of programs. I have combined as many as ten or twelve short demonstration programs into one SAVE and LOAD unit on tape.

ANOTHER PHONE DIALER

Dear Editor:

I used the algorithm from Cash Olsen's "Fast Repertory Telephone Dialer Program" (DDJ #38) to develop an automatic phone dialer for my PET.

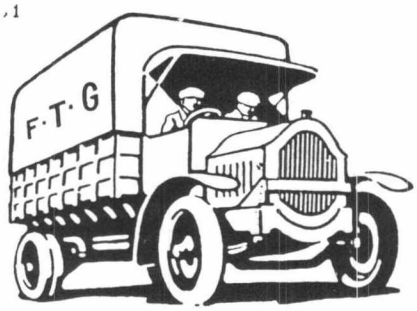
I used the Microtronics M65 HAM Interface. The relay is placed in series with the green wire to the phone and works fine. I used a phone plug jack which closes the circuit for the green wire for normal phone operation when the computer is unplugged.

The program is documented with REM statements and is self explanatory in conjunction with Olsens article.

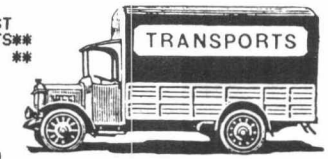
Thanks
Gary Lindenschmitt

8285 Royall Oaks Dr.
Roseville CA 95678

```
READY.

 1 REM LINDY/EDIN *DIALER*  DR DOBBS SEP79
 2 REM DRIVE RELAY FROM USER PORT PA0
 3 POKE59459,255
 10 PRINT"■■■■■"
 11 PRINT:PRINT:PRINT"WOULD YOU LIKE ME TO CALL SOMEONE?":GOTO 21
 20 PRINT:PRINT:PRINT"DO YOU WANT ME TO PHONE SOMEONE ELSE ?"
 21 GET G$:IF G$="" THEN 21
 22 IF G$="Y" THEN PRINT"■■■■■WHO SHOULD I CALL";:GOTO 2000
 23 PRINT"■■■■■■■THANK YOU.■":STOP
 30 D=1:IFZ=1THENGOSUB110
 35 IFA=1THEN400
 40 IFA>=999THEN60
 50 F=A:E=100:GOSUB300
 60 IFP>999THEN20
 65 PRINT"   ";
 70 F=P:E=100:GOSUB300
 75 PRINT"   ";
 80 IFL>9999THEN20
 85 F=L:E=1000:GOSUB300
 90 GOTO 2050
 100 IFD=0THEND=10
 110 FORN=1TOD
 120 REM OPEN RELAY 33MS
 121 POKE59459,255:POKE59471,1
 125 T=1:C=TI
 126 B=TI
 128 IFB>=C+(T*2 )THEN132
 130 GOTO126
 132 REM PRINT(B-C)/60
 134 REM CLOSE RELAY 16MS
 135 T=1:C=TI:POKE59471,0
 137 B=TI
 138 IFB>=C+(T*1)THEN140
 139 GOTO137
 140 REM PRINT(B-C)/60
 141 NEXTN
 142 T=1:C=TI
 143 B=TI
 144 IFB>=C+(T*12)THEN146
 145 GOTO143
 146 REM PRINT(B-C)/60
 150 RETURN
 300 D=INT(F/E):F=F-(D*E)
 305 PRINTD;
 310 GOSUB100
 320 IFE=1THENRETURN
 330 E=E/10:GOTO300
```



```
400 D=1:PRINT" 1 ";
410 GOSUB110
420 GOTO60
500 END
990 REM**INSERT DATA STMTS HERE**
995 REM**"NAME",AREA CODE,PREFIX,LAST
996 REM                 FOUR DIGITS**
997 REM**FOR LOCAL CALLS USE 1000    **
1000 DATA"WORK",1,916,643,4601
1010 DATA"HOME",1,916,791,4298
1020 DATA"MOM",0,1000,725,5966
1030 DATA"TIME",0,1000,783,1600
1040 DATA"JIM ELROD",0,1,988,4848
1050 DATA"BOB ZINKY",1,1000,421,1350
1070 DATA"JOE SUDDUTH",1,209,745,4331
1080 DATA"JACK EDIN",0,1000,791,7618
1090 DATA"DALLAS PARCHER",1,1000,486,0868
1999 DATA"END",0,0,0,0
2000 RESTORE:INPUTN$
2001 PRINT"■■■■
2002 PRINT"DIALING ";"■"N$"■";" NOW"
2003 PRINT:PRINT
2010 READM$,Z,A,P,L
2015 IF M$<>"END"THEN 2020
2017 PRINT"I'M SORRY BUT I DO NOT KNOW THAT NUMBER"
2018 PRINT"■■IS THERE SOMEONE ELSE YOU WOULD LIKE ME TO CALL?":GOTO 21
2020 IFN$<>M$THEN2010
2025 PRINT"■AREA CODE  PREFIX   NUMBER"
2030 PRINT:PRINTA,P,L:PRINT
2040 RESTORE
2045 GOTO 30
2050 PRINT:PRINT"■■■■■WAIT FOR RING":PRINT
2051 FORI=1TO10000:NEXT
2052 PRINT:PRINT"■■DO YOU WANT ME TO DIAL THE SAME NUMBER  AGAIN?";
2055 GET C$:IF C$="" THEN 2055
2060 IF C$<>"Y" THEN 20
2061 REM HANG UP FOR 3 SEC & DIAL AGAIN
2062 PRINT"OK":X=1
2065 POKE59471,X
2070 T=3:C=TI
2072 B=TI
2074 IFB>=C+(T*60)THEN2080
2076 GOTO2072
2078 PRINT(B-C)/60
2080 IFX=0THEN2001
2082 X=0:GOTO2065
2095 GOTO2001
READY.
```



ABOUT THE PERSONAL COMPUTING SOCIETY

To *DDJ* Readers:

I am frequently asked *why,* in regard to why the Personal Computing Society is trying so hard to become a vital service force for the personal computer user. Hobbyist groups dot the country, courteously exchanging newsletters. Professional groups, such as those under ACM, *span* the country, addressing issues of importance, such as educational and handicapped applications, standards development, data security and the like. Is not, then, PCS redundant?

There are two major differences between the Personal Computing Society and those referenced above:

(1) For those already involved with computers, there is a substantial cost efficiency. A nominal dues fee gives you access to one central source of information, and one centralized agency coordinating your opinion with those who share it across the country.

Professional organizations charge expensive fees, since these moneys, often reimbursed by members' employers, must support massive professional staffs and housing facilities.

PCS requests a fee of $10 annually for individuals, but clubs can bring their entire membership into our fold for a flat $25. Projects currently underway include a study of legal and taxation considerations for hobbyist organizations, a network capability stringing together the bulletin board facilities already utilized by individuals and clubs across the country, and a newsletter. These are projects of interest to hobbyists; there are others for educators, consumers, varying special interest groups.

(2) The PCS addresses the problems of the lay public, as well as the computer expert. No hobbyist or professional organization reaches the potential first-time user to educate him so that he can purchase wisely. Those of you who already use computers are gravely affected by the ignorance of the public. Unless the lay person is aided with purchase guidelines, this vast market remains untapped as a source of influence on manufacturers. The vendors then remain free to produce without guidelines of quality assurance, and without listening to the needs of the user.

Following purchase, the novice computer user becomes one of your number. He will require the same supportive information exchange hobbyists and professionals enjoy, but will not know this exchange exists, let alone where to find it. Allowed to enter the information network, the novice becomes an expert, adding his talent to the pool upon which computer users draw for assistance. Thus, the computer expert *needs* the support of this uninitiated public, and should reach out to it.

All that we do is accomplished by volunteers. We wish to keep personal computing within the personal control of the individual user, so that he can determine the quality of his equipment, while increasing his knowledge in its use. We aim to accomplish this goal at a minimal cost, by circumventing bureaucratic overhead and appealing to members to help each other.

The most important contribution you can make is your ideas. When you join, we become *your* group. It is up to you to tell us what you want accomplished. It is up to us, the collective, to make it happen.

For further information, write Abby Gelles, Executive Director, Personal Computing Society, Inc., Box 147, Village Station, New York, New York 10014. If you wish to join at this time, enclose a $10 (individual) or $25 (organizations) check payable to Personal Computing Society, Inc. Thank you.

Abby Gelles
The Executive

Personal Computer Society
185 W. Houston St.
NY NY 10014

Dear *Dr. Dobb's,*

Amberse M. Banks article, page 43 of *DDJ* #39, was a big help to me for Ver 4 Northstar DOS.

As I also use the XEK Assembler from the Byte Shop of Westminster that requires Ver 2 DOS, I needed the two column approach for Ver 2 as well.

Attached is a copy of the changes needed for Version 2.

Sincerely,
Don Dodge

5291 Kenilworth Dr.
Huntington Beach CA, 92649

```
A 0

0000                0010 ;VERSION 2 NORTHSTAR PATCHES TO GIVE TWO COLUMN OUTPUT
0000                0020 ;BY DELETING CR LF CALL AND SUBSTITUTING A SPACING ROUTINE
0000                0030 ;TO EVEN OUT THE COLUMNS.  ROUTINE FOR VER 4 AND VER 5
0000                0040 ;PUBLISHED IN DR. DOBBS, NUMBER 39, OCTOBER 1979,
0000                0050 ;VOLUME 4, ISSUE 9. AUTHOR AMBERSE M. BANKS.
0000                0060         ORG   29E0H
29E0 0E 0C          0070         MVI   C,12          ;ADD 12 SPACES FOR COLUMNS
29E2 CD 02 27       0080 SPACE   CALL  2702H         ;CALL SPACE ROUTINE
29E5 0D             0090         DCR   C
29E6 C2 E2 29       0100         JNZ   SPACE
29E9 C9             0110         RET
29EA                0120 ;
29EA                0130 ;ALSO CHANGE CALL AT 2707 FROM CD 07 27 TO CD E0 29
29EA                0140 ;BY DON DODGE
29EA                0150 ;    5291 KENILWORTH DR.
29EA                0160 ;    HUNTINGTON BEACH CA, 92649

SYMBOL TABLE

SPACE 29E2
```

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 43 | March 1980 | Volume 5, Issue 3

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

Mathematical Typography 5
Donald E. Knuth

TI9900: Fully Reentrant Formatter Operating Under TIPMX, Part 2 26
Norman V. Walters

SYMBOL: Generating Symbol Table Alpha Listing on the H8 21
William W. Moss

A Hex Keyboard with Applications for the 1802 ELF 40
Martin Bronstein

Can You Afford to Retire? 39
David E. Raeside

Mathematical Typography

BY DONALD E.KNUTH
Computer Science Department
Stanford University
Stanford, California 94305
©1979 by the American Mathematical Society

ABSTRACT.

Mathematics books and journals do not look as beautiful as they used to. It is not that their mathematical content is unsatisfactory, rather that the old and well-developed traditions of typesetting have become too expensive. Fortunately, it now appears that mathematics itself can be used to solve this problem.

A first step in the solution is to devise a method for unambiguously specifying mathematical manuscripts in such a way that they can easily be manipulated by machines. Such languages, when properly designed, can be learned quickly by authors and their typists, yet manuscripts in this form will lead directly to high quality plates for the printer with little or no human intervention.

A second step in the solution makes use of classical mathematics to design the shapes of the letters and symbols themselves. It is possible to give a rigorous definition of the exact shape of the letter "a", for example, in such a way that infinitely many styles (bold, extended, sans-serif, italic, etc.) are obtained from a single definition by changing only a few parameters. When the same is done for the other letters and symbols, we obtain a mathematical definition of type fonts, a definition that can be used on all machines both now and in the future. The main significance of this approach is that new symbols can readily be added in such a way that they are automatically consistent with the old ones.

Of course it is necessary that the mathematically-defined letters be beautiful according to traditional notions of aesthetics. Given a sequence of points in the plane, what is the most pleasing curve that connects them? This question leads to interesting mathematics, and one solution based on a novel family of spline curves has produced excellent fonts of type in the author's preliminary experiments. We may conclude that a mathematical approach to the design of alphabets does not eliminate the artists who have been doing the job for so many years; on the contrary, it gives them an exciting new medium to work with.

Reprinted from Donald E. Knuth, "Mathematical Typography," Bulletin (New Series) of the AMS, volume 1, number 2, pp. 337-372, by permission of the American Mathematical Society. This article is the Josiah Willard Gibbs Lecture given by Mr. Knuth in February, 1978.

INTRODUCTION.

I will be speaking today about work in progress, instead of completed research; this was not my original intention when I chose the subject of my lecture, but the fact is I couldn't get my computer programs working in time. Fortunately, it is just as well that I don't have a finished product to describe to you today, because research in mathematics is generally much more interesting while you're doing it than after it's all done. I will try therefore to convey in this lecture why I am so excited about the project I am currently working on.

My talk will be in two parts, based on two different meanings of its title. First I will speak about mathematical typography in the sense of typography as the servant of mathematics: the goal here is to communicate mathematics effectively by making it possible to publish mathematical papers and books of high quality, without excessive cost. Then I will speak about mathematical typography in the sense of mathematics as the servant of typography: in this case we will see that mathematical ideas can make advances in the art of printing.

PRELIMINARY EXAMPLES.

To set the stage for this discussion I would like to show you some examples by which you can "educate your eyes" to see mathematics as

a printer might see it. These examples are taken from the *Transactions of the American Mathematical Society*, which began publication in 1900; by now over 230 volumes have been published. I took these volumes from the library shelves and divided them into equivalence classes based on what I could perceive to be different styles of printing: two volumes were placed into the same class if and only if they appeared to be printed in the same style. It turns out that twelve different styles can be distinguished, and it will be helpful for us to look at them briefly.

The first example (Fig. 1a) comes from page 2 of *Transactions* volume 1; I have shown only a small part of the page in order to encourage you to look at the individual letters and their positions rather than to read the mathematics. This typeface has an old-fashioned appearance, primarily because the upper case letters and the taller lower case ones like 'h' and 'k' are nearly twice as tall as the other lower case letters, and this is rarely seen nowadays. Notice the style of the italic letter 'x', the two strokes having a common segment in the middle. The subscripts and superscripts are set in rather small type.

This style was used in volumes 1 to 12 of the *Transactions*, and also in the first 21 pages of volume 13. Then page 22 of volume 13 introduced a more modern typeface (Fig. 1b). In this example the subscript is still in a very small font, and unfortunately the Greek α here is almost indistinguishable from an italic 'a'. Notice also that the printer has inserted more space before and after parentheses than we are now accustomed to. During the next few years the spacing within formulas evolved gradually but the typefaces remained essentially the same up through volume 24: with one exception.

The exception was volume 23 in 1922 (Fig. 1c), which in my opinion has the most pleasing appearance of all the *Transactions* volumes. This modern typeface is less condensed, making it more pleasant to read. The italic letters have changed in style too, not quite so happily – note the 'x', for example, which is not as nice as before – but by and large one has a favorable impression when paging through this volume. Such quality was not without its cost, however; according to a contemporary report in the AMS *Bulletin* [45, p. 100], the *Transactions* came out 18 months late at the time! Perhaps this is why the Society decided to seek yet another printer.

In order to appreciate the next change, let's look quickly at two excerpts from the *Bulletin* relating to the very first Gibbs lecture (Fig. 2). The preliminary announcement in 1923 appeared in the modern typeface used during that year, but the letter shapes in the report of the first lecture in 1924 were very cramped and stilted. The upper case letters in the title are about the same, but the lower case letters in the text are completely different.

This same style appeared in volume 25 of the *Transactions* (Fig. 1d), which incidentally was set in Germany in order to reduce the cost of printing. Note that the boldface letters and the italic letters in this example are actually

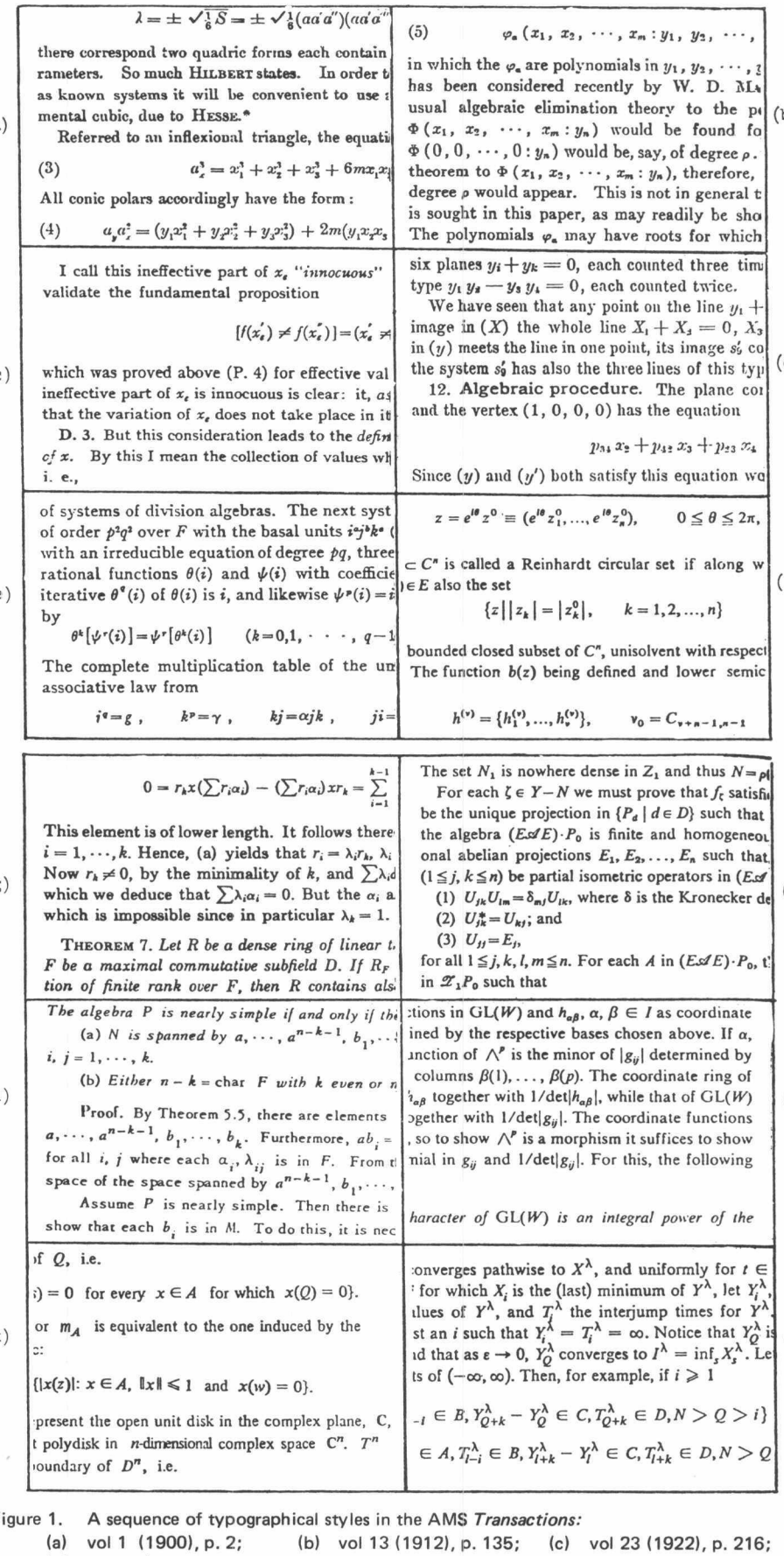


Figure 1. A sequence of typographical styles in the AMS *Transactions:*
(a) vol 1 (1900), p. 2; (b) vol 13 (1912), p. 135; (c) vol 23 (1922), p. 216;
(d) vol 25 (1923), p. 10; (e) vol 28 (1926), p. 207; (f) vol 105 (1962), p. 340;
(g) vol 114 (1965), p. 216; (h) vol 125 (1966), p. 38; (i) vol 169 (1972), p. 232;
(j) vol 179 (1973), p. 314; (k) vol 199 (1974), p. 370; (l) vol 225 (1977), p. 372.

quite beautiful – and we're back to the good old style of 'x' again – so the mathematical formulas looked great while the accompanying text was crowded. Fortunately only three volumes were published in this style.

A new era for the *Transactions* began in 1926, when its printing was taken over by the Collegiate Press in Menasha, Wisconsin. Volumes 28 through 104 were all done in the same style, covering 36 years from 1926 to 1961, inclusive, and this style (Fig. 1e) was used also in the *American Mathematical Monthly*. In general the modern typefaces were quite satisfactory, but there was also a curious anomaly: Italic letters used in subscripts and superscripts of mathematical formulas were in a different style from those used on the main line! For example, notice the two k's in the first displayed formula of Fig. 1e: the larger one has a loop, so it is topologically different from the smaller one. Similarly you can see that the p in k^p is quite different from the p in p^2. There are no x's in this example, but if you look at other pages you will find that the style of x that I like best appears only in subscripts and superscripts. I can't understand why this discrepancy was allowed to persist for so many years.

Another period of typographic turmoil for the *Transactions* began with volume 105 in (1962). This volume, which was typeset in Israel, introduced a switch to the Times Roman typeface (Fig. 1f); an easy way to recognize the difference quickly is to look at the shading on the letter o, since it now is somewhat slanted; in the previously used fonts this letter always was more symmetrical, as if it were drawn with a pen held horizontally, but in Times Roman it clearly has an oblique stress as if it were drawn by a right-handed penman. Note that the three k's are topologically the same in the displayed equation here; but for some reason the two subscripts k's are of different sizes. Many of the Times Italic letters have a somewhat different style than the readers of the *Transactions* had been accustomed to, and I personally think that this font tends to make formulas look more crowded. Actually the changeover to Times Roman and Times Italic wasn't complete; the italic letter g still had its familiar shape, perhaps because the new shape looked too strange to mathematicians.

Volumes 105 through 124 were all done in this style, except for a brief interruption: In volumes 114, 115, and 116 the shading on the o's was symmetrical and the k's had loops (Fig. 1g). Another style was used for volumes 125-168 (Fig. 1h): again Times Roman was the rule, even in the g's, except for subscripts and superscripts which were in the style I prefer; for example, compare the j's and k's. (These latter volumes were typeset in Great Britain.)

A greatly increased volume of publication, together with the rising salaries of skilled personel, was making it prohibitively expensive to use traditional methods of typesetting, and the Society eventually had to resort to a fancy form of typewriter composition that could simply be photographed for printing. This unfortunate circumstance made volumes 169-198 of the *Transactions* look like Fig. 1i, except for volumes 179, 185, 189, 192, 194, and 198 which were done in a far better (yet not wholly satisfactory) style that can be distinguished from Fig. 1f by the italic g's. Fig. 1j was composed on a computer using a system devoloped by Richard McQuillen; this was one of the fruits of an AMS research project supported by the National Science Foundation [2, 3, 4, 5, 6].

Computer typesetting of mathematics was still somewhat premature at the time, however, and another kind of "cold copy" made its appearance in volumes 199 through 224 – an "IBM Compositor" was used, except for volumes 208 and 211 which reverted to the Varityper style of Fig. 1i. The new alphabet was rather cramped in appearance, and some words were even more crowded than the others (see Fig. 1k). At this point I regretfully stopped submitting papers to the American Math. Society, since the finished product was just too painful for me to look at. Similar fluctuations of typographical quality have appeared recently in all technical fields, especially in physics where the situation has gotten even worse. (The history of publication at the American Society of Civil Engineers has been discussed in an interesting and informative article by Paul A. Parisi [44].)

Fortunately things are now improving. Beginning with volume 225, which was published last year, the *Transactions* now looks like Fig. 1l; like Fig. 1j, it is computer composed, and the Times Roman typeface is now somewhat larger. I still don't care for this particular style of italic letters, and there are some bugs needing to be ironed out such as the overlap between lines shown in this example; but it is clear that the situation is getting better, and perhaps some day we will once again be able to approach the quality of volumes 23 and 24.

COMPUTER-ASSISTED COMPOSITION.

Perhaps the main reason that the situation is improving is the fact that computers are able to manipulate text and convert it into a form suitable for printing. Experimental systems of this kind have been in use since the early 1960's (*cf.* the book by Barnett [10]), and now they are beginning to come of age. Within another ten years, I expect that most office typewriters will be replaced by television screens attached to a keyboard and to a small computer. It will be easy to make changes to a manuscript, to replace all occurrences of one phrase by another and so on, and to transmit the manuscript either to the television screen, or to a printing device, or to another computer. Such

THE JOSIAH WILLARD GIBBS LECTURESHIP

The Council of the Society has sanctioned the establishment of an honorary lectureship to be known as the Josiah Willard Gibbs Lectureship. The lectures are to be of a popular nature on topics in mathematics or its applications, and are to be given by invitation under the auspices of the Society. They will be held annually or at such intervals as the Council may direct. It is expected that the first lecture will be delivered in New York City during the winter of 1923–24, and a committee has been authorized to inaugurate the lectures by choosing the first speaker and making the necessary arrangements.

R. G. D. RICHARDSON,
Secretary.

THE FIRST JOSIAH WILLARD GIBBS LECTURE

The first Josiah Willard Gibbs Lecture was delivered under the auspices of this Society on February 29, 1924, by Professor M. I. Pupin, of Columbia University, in the auditorium of the Engineering Societies' Building, New York City. A large and distinguished audience was present, including, besides members of the Society, many physicists, chemists, and engineers who had been invited to attend.

In introducing the speaker, President Veblen spoke as follows:

"In instituting the Willard Gibbs Lectures, the American Mathematical Society has recognized the dual character of mathematics. On the one hand, mathematics is one of the essential emanations of the human spirit,—a thing to be valued in and for itself, like art or poetry. Gibbs made

Figure 2. A time of transition. (Excerpts from the AMS Bulletin 29 (1923), p. 385; 30 (1924), p. 289.)

Formula	Type C	Type B	Type T
$\frac{1}{2}$	`$f1$s2$t`	`1 over 2`	`1 \over 2`
θ^2	`*gq"2`	`theta sup 2`	`\theta↑2`
$\sqrt{f(x_i)}$	`$rf(x'i)$t`	`sqrt{f(x sub i )}`	`\sqrt{f(x↓i)}`

Figure 3. Three ways to describe a formula.

systems are already in use by most newspapers, and new experimental systems for business offices actually will display the text in a variety of fonts [26]. It won't be long before these machines change the traditional methods of manuscript preparation in universities and technical laboratories.

Mathematical typesetting adds an extra level of complication, of course. Printers refer to mathematics as "penalty copy", and one of America's foremost typographers T. L. De Vinne wrote that "[even] under the most favorable conditions algebra will be troublesome." [17, p. 171] The problem used to be that the two-dimensional formulas required complicated positioning of individual metal pieces of type; but now this problem reduces to a much simpler one, namely that two-dimensional formulas need to be represented as a one-dimensional sequence of instructions for transmission to the computer.

One-dimensional languages for mathematical formulas are now familiar in programming languages such as FORTRAN, but a somewhat different approach is needed when all of the complexities of typesetting are considered. In order to show you the flavor of language for mathematical typesetting, I will briefly describe the three reasonably successful systems known to me. The first, which I will call Type C, is typical of the commercially available systems now used to typeset mathematical journals such as the *Transactions* (*cf.* [12]). The second, which I will call Type B, was developed at Bell Telephone Laboratories and has been used to prepare several books and articles including the article that introduced the system [27]. The third, which I will call Type T, is the one I am presently developing as part of the system I call TEX [29]. (This has no connection with a similarly-named system recently announced by Honeywell Information Systems. In my language, the T, E, and X are Greek letters and TEX is pronounced "tech", following the Greek words for art and technology).

Fig. 3 shows how three simple formulas would be expressed in these three languages. The Type C language uses \$f...\$s...\$t for fractions, *g for "the next character is Greek," q for the Greek letter theta, " for superscripts, \$r...\$t for square roots and ' for subscripts. The Type B language is more mnemonic, using "over", "theta", "sup", "sqrt", and "sub" together with braces for grouping when necessary. The Type T language is similar but it does not make use of "reserved words"; a special character \ is used before any nonstandard text. This means that spaces can be ignored, while they need to be inserted in just the right places in the Type B language; for example, the space after the "i" is important in the example shown, otherwise $f(x_i)$ would become $f(x_i)$ according to the Type B rules. Another reason for the \ delimiter in Type T is that it becomes unnecessary to match each text item against a stored dictionary, and it is possible to use "sup" to mean supremum instead of superscript. The special symbols \ { } ↑ ↓ in Type T can be changed to any other characters if desired; although these five symbols don't appear on conventional typewriters, they are common on computer terminal keyboards.

Incidentally, computer typesetting brings us some good news: It is now quite easy to represent square roots in the traditional manner with radical signs or vinculums, so we won't have to write $x^{1/2}$ when we don't want to. (Added in proof: I was pleased to find that this announcement was greeted with an enthusiastic round of applause when I delivered the lecture).

None of these languages makes it possible to *read* complex formulas as easily as in the two-dimensional form, but experience shows that it is not difficult for untrained personnel to learn how to type them. According to [12]. "Within a few hours (a few days at most) a typist with no math or typesetting background can be taught to input even the most complex equations." And the Type B authors [27] report that "the learning time is short. A few minutes gives the general flavor, and typing a page or two of a paper generally uncovers most of the misconceptions about how it works." Thus it will be feasible for both typists and mathematicians to prepare papers in such a language, without investing a great deal of effort in learning the system. The only real difficulties arise when preparing tables that involve tricky alignments.

Once such systems become widespread, authors will be able to prepare their papers and see exactly how they will look when printed. Everyone who writes mathematical papers knows that his intentions are often misunderstood by the printer, and corrections to the galley proofs have a nontrivial probability of introducing further errors. Thus, in the words of three early users of the Bell Lab's system, "The moral seems clear. If you let others do your typesetting, then there will be errors beyond your control; if you do your own, then you have only yourself to blame." [1] Personally, I can't adequately describe how wonderful it feels when I now make a change to the manuscript of my book, as it is stored in the Stanford computer, since I *know* that the change is immediately in effect; it never will go through any middlemen who might misunderstand my intention.

Perhaps some day a typesetting language will become standardized to the point where papers can be submitted to the American Mathematical Society from computer to computer via telephone lines. Galley proofs will not be necessary, but referees and/or copy editors could send suggested changes to the author, and he could insert these into the manuscript, again via telephone.

Of course I am hoping that if any language becomes standard it will be my TEX language. Well... perhaps I am biased, and I know that TEX provides only small refinements over what is available in other systems. Yet several dozen small refinements add up to something that is important to me, and I think such refinement might prove important to other people as well. Therefore I'd like to spend the next few minutes explaining more about TEX.

THE TEX INPUT LANGUAGE.

TEX must deal with "ordinary" text as well as mathematics, and it is designed as a unified system in which the mathematical features blend in with the word-processing routines instead of being "tacked on" to a conventional typesetting language. The main idea of TEX is to construct what I call *boxes*. A character of type by itself is a box, as is a solid black rectangle, and we use such "atoms" to construct more complex boxes analogous to "molecules," by forming horizontal or vertical lists of boxes. The final pages of text are boxes made out of lists of boxes made out of lists of boxes, and so on down to the individual characters and black rectangles which are not decomposed further; for example, a typical page of a book is a box formed from vertical lists of boxes representing lines of type, and these lines of type are boxes formed from a horizontal list of boxes representing individual letters. A mathematical formula breaks down into boxes in a natural way; for example, the numerator and denominator of a fraction are boxes, and so is the bar line between them (since it is a thin but solid black rectangle). The elements of a rectangular matrix are boxes, and so on.

The individual boxes of a horizontal list or a vertical list are separated by a special kind of elastic mortar which I call "glue." The glue between two boxes has three component parts (x, y, z) expressed in units of length:

- the *space* component, x, is the ideal or normal space desired between these boxes;
- the *stretch* component, y, is the amount of extra space which is tolerable;
- the *shrink* component, z, is the amount of space which may be removed if necessary.

Suppose the list contains $n+1$ boxes $B_0, B_1, \ldots, B_n$ separated by n globs of glue having specifications $(x_1, x_1, z_1), \ldots, (x_n, y_n, z_n)$. When this list is made into a box, we *set the glue* according to the desired final size of the box. If the final size is to be larger than we would obtain with the normal spacing $x_1 + \ldots + x_n$, we increase the space proportional to the y's so that the actual space between boxes is

$$x_1 + ty_1, \ldots, x_n + ty_n$$

for some appropriate $t > 0$. On the other hand if the desired final size must be smaller, we decrease the space to

$$x_1 - tz_1, \ldots, x_n - tz_n ,$$

in proportion to the individual shrinkages z_i. In the later case t is not allowed to become greater than 1; the glue will never be smaller than $x - z$, although it might occasionally become greater than $x + y$. Once the glue has been set, the box is rigid and never changes its size again.

Consider, for example, a normal line of text, which is a list of individual character boxes.

The glue between letters of a word will have x = y = z = 0, say, meaning that this word which is a list of individual character boxes. The glue between letters of a word will have x = y = z = 0, say, meaning that this word always has the letters butting against each other; but the glue between words might have x equal to the width of the letter 'e,' and y = x, and z = ½x, meaning that the space between words might expand or shrink. The spaces after punctuation marks like periods and commas might be allowed to stretch at a faster rate but to shrink more slowly.

An important special case of this glue concept occurs when we have "infinite" stretchability. Suppose the x and z components are zero, but the y component is extremely large, say y is one mile long. If such an element of glue is placed at the left of a list of boxes, the effect will be to put essentially all of the expansion at the left, therefore the boxes will be right-justified so that the right edge will be flush with the margin. Similarly if we place such infinitely stretchable glue at both ends of the list, the effect will be to center the line. These common typographic operations therefore turn out to be simple special cases of the general idea of variable glue, and the computer can do its job more elegantly since it is dealing with fewer primitives. Incidentally you will notice from this example that glue is allowed to appear at the ends of a list, not just between boxes; actually it is also possible to have glue next to glue, and boxes next to boxes, so that a list of boxes really is a list of boxes and glue mixed in any fashion whatever. I didn't mention this before, because for some reason it seems easier to explain the idea first in the case when boxes alternate with glue.

The same principles apply to vertical lists. For example, the glue which appears above and below a displayed equation will tend to be stretchable and shrinkable, but the glue between lines of text will be calculated so that adjacent base lines will be uniformly spaced when possible. You can imagine how the concept of glue allows you to do special tricks like backspacing (by letting x be negative), in a natural manner.

LINE DIVISION.

One of the more interesting things a system like TEX has to do is to divide up a paragraph into individual lines so that each line is about the right length. The traditional way to do this, which is still used on today's computer typesetting systems, is to make the best possible line division you can whenever you come to the right margin, but once this line has been output you never reconsider it again—you start the next line with no memory of what has come before. Actually it often happens that one could do better by moving a short word down from one line to the next, but the problem is that you don't know what the rest of the paragraph will be like when you have only looked at one line's worth.

The TEX system will introduce a new approach to the problem of line division, in which the end of a paragraph *does* influence the way the first lines are broken; this will result in more even spacing and fewer hyphenated words. Here is how it works: First we convert the line division problem to a precisely-defined mathematical problem by using TEX's glue to introduce the concept of "badness." When a horizontal list of boxes has a certain natural width w (based on the width of its boxes and the space components of its glue), a certain stretchability y (the sum of the stretch components) and a certain shrinkability z (the sum of the shrinkages), the *badness* of setting the glue to make a box of width W is defined to be $1 + 100t^3$ in our previous notation; more precisely, it is

$$\begin{cases} 1 & \text{, if } W = w, \\ 1 + 100\left(\dfrac{W-w}{y}\right)^3 & \text{, if } W > w, \\ 1 + 100\left(\dfrac{w-W}{z}\right)^3 & \text{, if } w - z \leqslant W < w, \\ \text{infinite} & \text{, if } W < w - z. \end{cases}$$

Thus if the desired width W is near the natural width w, or if there is a lot of stretchability and shrinkability, the badness rating is very small; but if W is much greater than w and there isn't much ability to stretch, we have a lot of badness. Furthermore we add *penalty points* to the badness rating if the line ends at a comparatively undesirable place; for example, when a word needs to be hyphenated, the badness goes up by 25, and an even worse penalty is paid if we have to break up mathematical formulas.

The line division problem may now be stated as follows. "Given the text of a paragraph and the set of all allowable places to break it between lines, find breakpoints which minimize the sum of the squares of the badnesses of the resulting lines." This definition is quite arbitrary, of course, but it seems to work. Preliminary experiments show that the same choice of breakpoints is almost always found when simply minimizing the sum of the individual badnesses rather than the sum of their squares, but it seems wise to minimize the sum of squares as a precautionary measure since this will also tend to minimize the maximum badness.

Just stating the line division problem in mathematical terms doesn't solve it, of course; we need to have a good way to find the desired breakpoints. If there are n permissible places to break (including all spaces between words and all possible hyphenations), there are 2^n possible ways to divide up the paragraph, and we would never have time to look at them all. Fortunately there is a technique that can be used to reduce the number of computational steps to order n^2 instead of 2^n; this is a special case of what Richard Bellman calls "dynamic programming." Let f(j) be the minimum sum of badness squares for all ways to divide the initial text of the paragraph up to breakpoint j, including a break at j, and let b(i, j) be the badness of a line which runs from breakpoint i to breakpoint j. Let breakpoint 0 denote the beginning of the paragraph; and let breakpoint n + 1 be the end of the paragraph, with infinitely expandable glue inserted just before this final breakpoint. Then

$$f(0) = 0;$$
$$f(j) = \min_{0 \leqslant i < j} \left(f(i) + b(i,j)^2\right),$$
$$\text{for } 1 \leqslant j \leqslant n+1.$$

The computation of f(1), . . . , f(n+1) can be done in order n^2 steps, and f(n+1) will be the minimum possible sum of badnesses squared. By remembering the values of i at which the minima occurred for each j, we can find breakpoints which give the best line divisions, as desired.

In practice we need not test extremely unlikely breakpoints; for example, there is rarely any reason to hyphenate the very first word of a paragraph. Thus it turns out that this dynamic programming method can be further improved to an algorithm whose running time is almost always of order n instead of n^2, and comparatively few hyphenations will need to be tried. Incidentally, the problem of hyphenation itself leads to some interesting mathematical questions, but I don't have time to discuss them today. (Cf. [41] and the references in that paper.)

The idea of badness ratings applies in the vertical dimension as well as in the horizontal; in this case we want to avoid breaking columns or pages in a bad manner. For example, penalty points are given for splitting a paragraph between pages after a hyphenation, or for dividing it in such a way that only one of its lines—a so-called "widow" line—appears on a page. The placement of illustrations, tables, and footnotes is also facilitated by formulating appropriate rules of placement in terms of badness.

There is more to TEX, including for example some facilities for handling the rather intricate layouts often needed to typeset tables without having to calculate column widths; but I think I have described the most important principles of its organization. During the next few months I plan to write the computer programs for TEX in such a way that each algorithm is clearly explained and so that the system can be implemented on many different computers without great difficulty; then I intend to publish the programs in a book so that everyone who wants to can use them.

ENTR'ACTE.

I said at the beginning that this talk would be in two parts, discussing both the ways that typography can help mathematics and that mathematics can help typography. So far we have seen a little of both, but the mathematics has been comparatively trivial. In the remainder of my lecture I would like to discuss what I believe is a much more significant application of mathematics to typography, namely to the

specification of the letter shapes themselves. A more accurate way to describe the two parts of my lecture would be to say that the first part was about TEX, a system which takes manuscripts and converts them into specifications about where to put each character on each page; and the second part will be about another system I'm working on called METAFONT, which generates the characters themselves, for use in the inkier parts of the printing business.

Before I get into the second part of my lecture I need to discuss recent developments in printing technology. The most reliable way to print mathematics books of high quality during the past several decades has been to use the monotype process (actually the Monotype Corporation now manufactures digital photosetting equipment as well as the traditional 'monotype' machines) which casts characters in hot lead, together with hand operations for complex built-up formulas. When I watched this process being applied to my own books several years ago, I was surprised to learn that the lead type was used to print only *one* copy; this master copy was then photographed, and the real printing took place from the photographic plates. This somewhat awkward sequence of steps was justified because it was the best way known to give good results. During the 1960's, however, hot lead type was replaced for many purposes by devices like the Photon machine used to prepare the printed programs for this lecture; in this case the process is entirely photographical, since the letter shapes are stored as small negatives on a rotating disk, and the plates needed for printing are obtained by exposing the film after transforming the characters into the proper size and position with mirrors and lenses (cf. [10]). Such machines are limited by slow speed and the difficulties of adding new characters.

"THIRD-GENERATION" TYPESETTING EQUIPMENT

More recent machines, such as the one used to prepare the current volumes of the *Transactions,* have replaced these "analog" processes by a "digital" one. The new idea is to divide the page or the photographic negative into millions of tiny rectangles, like a piece of graph paper or like a television screen but with a much higher resolution of about 1000 lines per inch. For each of the tiny "pixels" in such a raster pattern – there are about a million square pixels in every square inch – the typesetting machine decides whether it is to be black or white, and the black ones are exposed on the photographic plate by using a very precisely controlled electron beam or laser beam. Since these machines have few moving parts and require little or no mechanical motion, they can operate at very high speeds even though they are exposing only a tiny bit of the film at any time.

Stating this another way, the new printing equipment essentially treats each page of a book as a huge matrix of 0's and 1's, with ink to be placed in the positions that are 1 while the 0 positions are to be left blank. It's like the flashcards at a football stadium, although on a much grander scale. The total job of a system like TEX now becomes one of converting an author's manuscript into a gigantic bit matrix.

The first question we must ask of course is, "What happens to the quality?" Clearly a television picture is no match for a photograph, and the digital typesetting machines would be quite unsatisfactory if their output looked inferior to the results obtained with metal type. In matters like this, I have to confess being somewhat of a stickler and a perfectionist; for example, I refuse to eat margarine instead of butter, and I have never heard an electronic organ that sounds even remotely as beautiful as a pipe organ. Therefore I was quite skeptical about digital typography, until I saw an actual sample of what was done on a high quality machine and held it under a magnifying glass: It was impossible to tell that the letters were generated with a discrete raster! The reason for this is not that our eyes can't distinguish more than 1000 points per inch; in appropriate circumstances they can. The reason is that particles of ink can't distinguish such fine details – you can't print the edge of an ink line that zigzags 1000 times on the diagonal of a square inch, the ink will round off the edges. In fact the critical number seems to be more like 500 than 1000. Thus the physical properties of ink cause it to appear as if there were no raster at all.

It now seems clear that discrete raster-based printing devices will soon make the other machines obsolete for nearly all publishing activity. Thus in future days the fact that Gutenberg and others invented movable type will not be especially relevant, it will merely be a curious historical fact which influenced history for only about 500 years. The ultimately relevant thing will be mathematics: the mathematics of matrices of 0's and 1's!

SEMI-PHILOSOPHICAL REMARKS.

I have to tell the next part of the story from my personal point of view. As a combinatorial mathematician, I really identify with matrices of 0's and 1's, so when I learned last spring about such printing machines it was impossible for me to continue what I was doing; I just had to take time off to explore the possibilities of the new equipment. My motivation was also increased by the degradation of quality I had been observing in technical journals; and furthermore the publishers of my books on computer programming had tried valiantly but unsuccessfully to produce the second edition of volume 2 in the style of the first edition without using the rapidly-disappearing hot lead process. It appeared that my books would soon have to look as bad as the journals! When I saw that these problems could all be solved by appropriate computer programming, I couldn't resist trying to find a solution by myself.

One of the most important factors in my motivation was the knowledge that the problem would be solved once and for all, if I could find a purely mathematical way to define the letter shapes and convert them to discrete raster patterns. Even though new printing methods are bound to be devised in the future, possibly even before I finish volume seven of the books I'm writing, any new machines are almost certain to be based on a high precision raster; and although the precision of the raster may change, the letter shapes can stay the same forever, once they are defined in a machine-independent form. My goal was therefore to give a precise description of the shapes of all the symbols I would need.

I looked at the way fonts of the type are being digitized at several places in different parts of the world; it is basically done by taking existing fonts and copying them using sophisticated camera equipment and computer programs, together with manual editing. But this seemed instinctively wrong to me, partly because the sophisticated equipment wasn't readily available in our laboratory at Stanford, and partly because the copying of copyrighted fonts is of questionable legality, but mostly because I felt that the whole idea of making a copy was not penetrating to the heart of the problem. It reminded me of the anecdote I had once heard about slide rules in Japan. According to this story, the first slide rule ever brought to the Orient had a black speck of dirt on it; so for many years all Japanese slide rules had a useless black spot in this same position! The story is probably apocryphal, but the point is that we should copy the substance rather than the form. I felt that the right question to ask would not be "How should this font of type be copied?" but rather: "If the great type designers of the past were alive today, how would they design fonts for the new equipment?" I didn't expect to be capable of finding the exact answer to this question, of course, but I did feel that it would lead me in the right direction, so I began to read about the history of type design.

Well, this is a most fascinating subject, but I can't talk much about it in a limited time. Two of the first things I read were autobiographical notes by two well-known 20th century type designers, Hermann Zapf [51] and Frederic W. Goudy [20], and I was especially interested by some of Zapf's remarks:

> With the beginning of the 'sixties ... I was stimulated by this new field [photocomposing] ... The type-designer – or better, let us start calling him the alphabet designer – will have to see his task and his responsibility more than before in the coordination of the tradition in the development of letterforms with the practical purpose and the needs of the advanced equipment of today ... The new photocomposing systems using cathode-ray tubes (CRT) or digital storage for the alphabet bring with them some absolutely new technical problems, many more than did the past ... [51, p.71].

I have the impression that Goudy would not have been so sympathetic to the new-fangled equipment, yet his book also gave helpful ideas.

MATHEMATICAL TYPE DESIGN.

Fortunately the Stanford Library has a wonderful collection of books about printing, and I had the chance to read many rather rare source materials. I learned to my surprise that the idea of defining letters mathematically is by no means new, it goes back to the fifteenth century and it became rather highly developed in the early part of the sixteenth. This was the time when there were Renaissance men who combined mathematics with the real world, and in particular there was an interest in constructing capital letters with ruler and compass. The first person to do this was apparently Felice Feliciano, about 1460, whose handwritten manuscript in the Vatican Library was published 500 years later [19]. Feliciano was an excellent calligrapher who wanted to put the principles of calligraphy on a sound mathematical foundation. Several other fifteenth-century authors made similar experiments ([8] gives a critical summary of these early developments), but the most notable work of this kind appeared in the early sixteenth century.

The Italian mathematician Luca Pacioli, who had previously written the most influential book on algebra at the time (one of the first algebra books ever published), included an appendix on alphabets in his *De Devina Proportione,* a book about geometry and the "golden section" which appeared in 1509. Another notable Italian work on the subject was published by Francesco Torniello in 1517 [48], [33]; Fig. 4 illustrates the letter B as constructed by Pacioli, Torniello, and by Giovanbattista Palatino [43]. Palatino was one of the best calligraphers of the century, and he did this work about 1550. Similar work appeared in Germany and France; the German book was probably the most famous and influential, it was Albrecht Durer's *Underweysung der Messung* [18], a manual of instruction in geometry for Renaissance painters. The French book was also rather popular, it was *Champ Fleury* by Geofroy Tory [49], the first royal printer of France and the man who introduced accented letters into French typography. Fig. 5 shows Tory's two suggestions for the letter B. Of all these books I much prefer Torniello's, since he was the only one who stated the constructions clearly and unambiguously.

Apparently nobody carried this work further to lower case letters, numerals, or italic letters and other symbols, until more than 100 years later when Joseph Moxon made a detailed study of some beautiful letters designed in Holland [38]. The ultimate in refinement of this mathematical approach took place shortly afterwards when Louis XIV of France commissioned the creation of a Royal Alphabet. A commission of artists and typographers worked on Louis' project for more than ten years beginning about 1690, and they made elaborate constructions such as those shown in Fig. 6 [24].

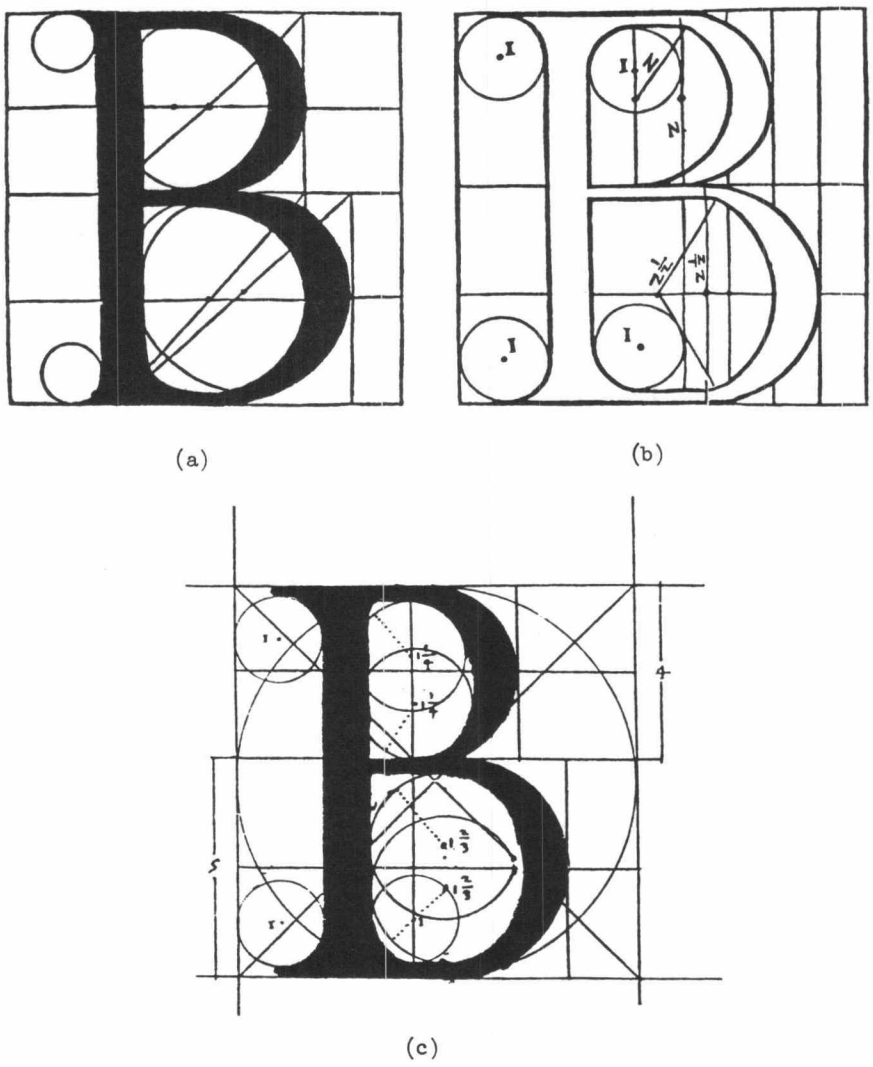


Figure 4. Sixteenth century ruler-and-compass constructions for the letter B by (a) Pacioli [42], (b) Torniello [48], and (c) Palatino [43].

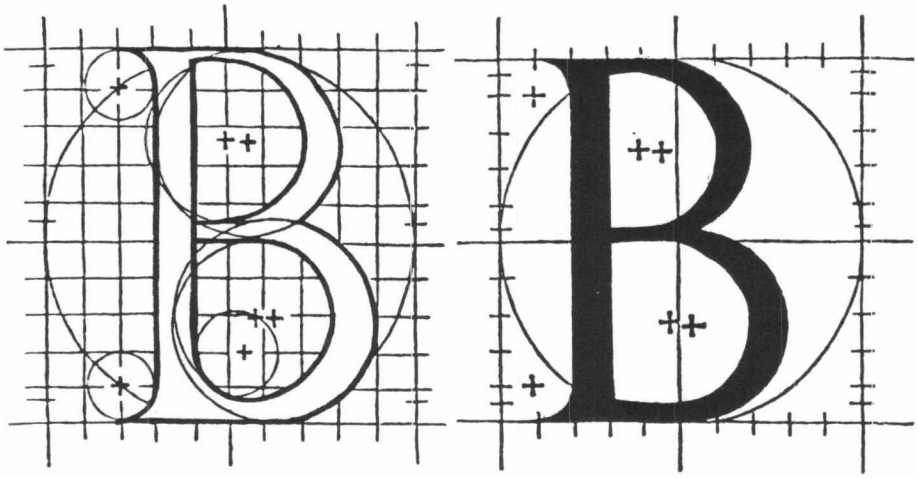
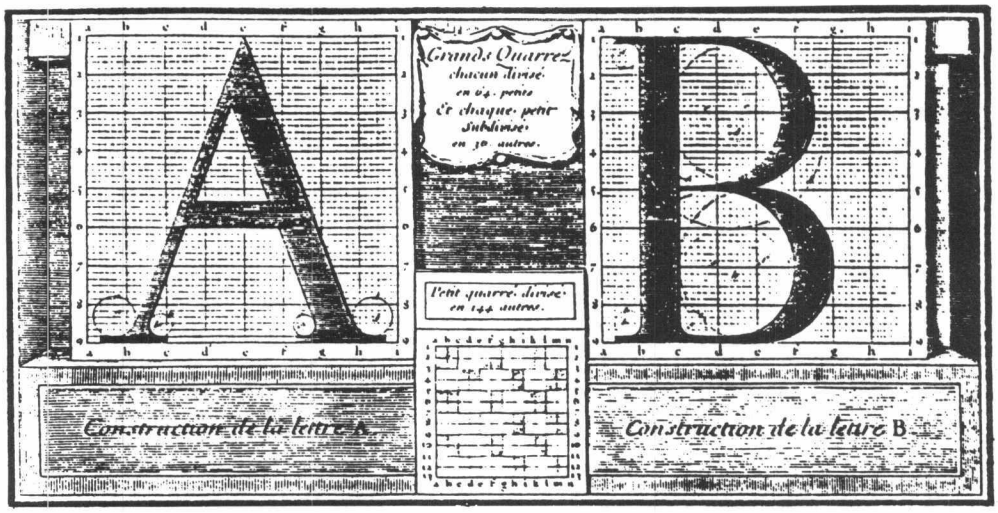


Figure 5. Two more B's, by Tory [49].

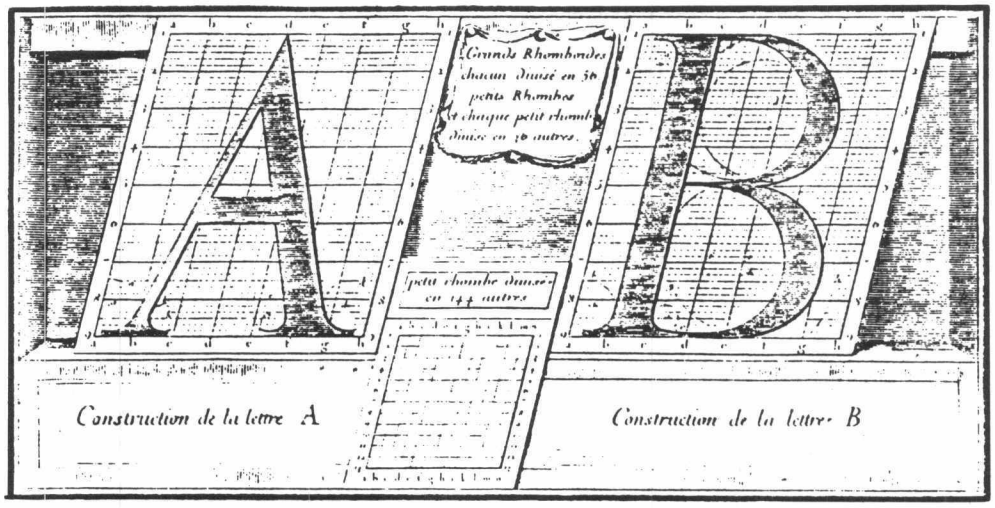
Thus it is clear that the mathematical definition of letter forms has a long history. However, I must also report near-universal agreement among today's scholars of typography that those efforts were a failure. At worst, the ruler-and-compass letters have been called "ugly" and at best they are said to be "deprived of calligraphic grace" [8]. The French designs were not really followed faithfully by Phillipe Grandjean who actually cut Louis XIV's type, nor by anybody else to date, and F. W. Goudy's reaction to this was: "God be praised!" [20, p. 139]. Such strictly geometric letter forms were in fact criticized already in the sixteenth century by Giovan Cresci, a noted scribe at the Vatican Library and the Sistine Chapel. Here is what Cresci wrote in 1560:

> I have come to the conclusion that if Euclid, the prince of geometry, returned to this world of ours, he would never find that the curves of the letters could be constructed by means of circles made with compasses. [16]

Well, Cresci was right. But fortunately there have been a few advances in mathematics during the last 400 years, and we now have some other tricks up our sleeves besides straight lines and circles. In fact, it is now possible to prescribe formulas that match the nuances of the best type designers; and perhaps a talented designer working with appropriate mathematical tools will be able to produce something even better than we now have.



(a)



(b)

Figure 6. Roman and italic letters designed for Louis XIV of France [24].

DEFINING GOOD CURVES.

Let's consider the following mathematical problem: Given n points $z_1, z_2, \ldots, z_n$ in the plane, what is the most pleasing closed curve that goes through them in the specified order $z_1, z_2, \ldots, z_n$ and then returns to z_1? To avoid degenerate situations we may assume that n is at least 4. This problem is essentially like the dot-to-dot puzzles that we give to young children.

Of course it is not a well-posed mathematical problem, since I didn't say what it means for a curve to be "most pleasing". Let's first postulate some axioms that the most pleasing curve should satisfy.

Property 1 (Invariance). If the given points are rotated, translated, or expanded, the most pleasing curve will be rotated, translated, or expanded in the same way. [In symbols: $MPC(az_1 + b, \ldots, az_n + b) = aMPC(z_1, \ldots, z_n) + b$.]

Property 2 (Symmetry). Cyclic permutation of the given points does not change the solution. [$MPC(z_1, z_2, \ldots, z_n) = MPC(z_2, \ldots, z_n, z_1)$.]

Property 3 (Extensionality). Adding a new point that is already on the most pleasing curve does not change the solution. [If z is between z_k and z_{k+1} on $MPC(z_1, \ldots, z_n)$, then $MPC(z_1, \ldots, z_k, z, z_{k+1}, \ldots, z_n) = MPC(z_1, \ldots, z_k, z_{k+1}, \ldots, z_n)$.]

These properties are rather easy to justify on intuitive grounds. For example, the extensionality property says that additional information won't lead to a poorer solution.

The next property is not so immediately apparent, but I believe it is important for the application I have in mind.

Property 4 (Locality). Each segment of the most pleasing curve between two of the given points depends only on those points and the ones immediately preceding and following. [$MPC(z_1, z_2, \ldots, z_n)$ is composed of $MPC(z_n, z_1, z_2, z_3)$ from z_1 to z_2, then $MPC(z_1, z_2, z_3, z_4)$ from z_2 to $z_3, \ldots$, then $MPC(z_{n-1}, z_n, z_1, z_2)$ from z_n to z_1.] According to the locality property, changes to one part of a pattern won't affect the other parts. This simplifies our search for the most pleasing curve, because we need only solve the problem in the case of four given points; and experience shows that it is also a great simplification when letters are being designed, since individual portions of strokes can be dealt with separately. Incidentally, property 4 implies property 2 (cyclic symmetry).

One way to satisfy all four of these properties is simply to let the most pleasing curve consist of straight line segments. But this doesn't seem adequately pleasing, so we postulate:

Property 5 (Smoothness). There are no sharp corners in the most pleasing curve. [$MPC(z_1, \ldots, z_n)$ is differentiable, under some parameterization.] In other words, there is a unique tangent at every point of the curve.

The extensionality, locality, and smoothness properties taken together imply, in fact, that *the direction of the tangent at z_2 depends only on z_1, z_2, and z_3.* For this tangent appears in two curves, the one from z_{k-1} to z_k and the one from z_k to z_{k+1}, hence we know that it depends only on $(z_{k-2}, z_{k-1}, z_k, z_{k+1})$ and

and that it depends only on $(z_{k-1}, z_k, z_{k+1}, z_{k+2})$. By the extensionality property, we can assume that n is at least 5, so z_{k-2} is different from z_{k+2} and the tangent must be independent of them both. We have actually used only a very weak form of extensionality in this argument.

If we apply the full strength of the extensionality postulate, we obtain a much stronger consequence, which is quite unfortunate: *There is no good way to satisfy Properties 1-5!* For example, suppose we add one more axiom, which is almost necessary in any reasonable definition of pleasing curves:

Property 6 (Roundness). If z_1, z_2, z_3, z_4 are consecutive points of a circle, the most pleasing curve through them is that circle. This property together with our previous observation about the tangent depending only on three points completely determines the tangent at each of our given points; namely, the tangent at z_k is the tangent to the circle which passes through z_{k-1}, z_k, and z_{k+1}. (Let's ignore for the moment the possibility that these three points lie on a straight line.) Now the extensionality property says that if z is any point between z_1 and z_2 on the most pleasing curve for $z_1, \ldots, z_n$, we know the tangent direction at z, as long as z is not the line from z_1 to z_2. But there is a unique curve starting at any z off this line and having the specified tangents at each of its points, namely the arc of the circle from z to z_2 passing through z_1: No matter where you start, off the straight line, there is only one curve having the correct tangents. It follows that the tangent at z_2 depends only on z_1, z_2 and the tangent at z_1, and this is impossible.

The above argument proves that there is no way to satisfy properties 3, 4, 5, and 6. A similar argument would show the impossibility for any reasonable replacement for property 6, since the tangents determined for all z between z_1 and z_2 will define a vector field in which there are unique curves through essentially all of the points z, yet a two-parameter family of curves is required between z_1 and z_2 in order to allow suficient flexibility in the derivatives there.

So we have to give up one of these properties. The locality property is the most suspicious one, but I mentioned before that I didn't want to give it up; therefore the extensionality property has to go. This means that if we take the most pleasing curve through $z_1, \ldots, z_n$ and if we specify a further point z actually on this curve between z_{k-1} and z_k, where the tangent at z is not the same as the tangent to the circle from z_{k-1} to z to z_k, then the "most pleasing" curve through these n+1 points will be different. A possible virtue is that we are encouraged not to specify too many points; a possible drawback is that we may not be able to get the curves we want.

A PRACTICAL APPROXIMATION.

Returning to the question of type design, our goal is to specify a few points z_k and to have a mathematical formula that defines a pleasant curve through these points; such curves will be used to define the shape of the character we are designing. Ideally it should also be easy to compute the curves. I decided to use cubic equations

$$z(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \alpha_3 t^3$$

where α_0, α_1, α_2, α_3 are complex numbers and t is a real parameter. The curves I am dealing with are *cubic spines*, namely piecewise cubic equations, since a different cubic will be used in each interval between two of the given points; however, the way I am determining the coefficients of these two cubics is different from any of the methods known to me, in my limited experience with the vast literature about spines. Perhaps my way to choose the coefficients is more awkward than the usual ones; but I have obtained good results with it, so I'm not ashamed to reveal the curious way I proceeded.

In the first place, I decided that the cubic equation between z_1 and z_2 should be determined completely by z_1 and z_2 and the directions of the tangents at z_1 and z_2. We have already seen that these tangents are essentially predetermined if properties 4, 5, and 6 are to be valid, and I have also found frequent occasion in type design when it was desirable to specify that a certain tangent was to be made horizontal or vertical. Thus, my method of computing a nice curve through a given sequence of points is first to compute the tangent directions at each point, then to compute the cubics in each interval based solely on the endpoints of that interval and on the desired tangents there. By rotation and translation and scaling, according to property 1, we can assume that the problem is to go in the complex plane from 0 to 1, with given directions at the endpoints. The most general cubic equation which does this is

$$z(t) = 3t^2 - 2t^3 + re^{i\theta}t(1-t)^2 + se^{-i\phi}t^2(1-t),$$

and it remains to determine positive numbers r and s as appropriate functions of θ and ϕ.

In the second place, I realized that it was impossible to satisfy property 6 with cubic splines, because you can't draw a circle as a cubic function of t. But I wanted to be able to get curves that were as near to being circles as possible, whenever four consecutive data points lay on a circle; the curves should preferably be indistinguishable from circles as far as the human eye is concerned. Therefore when $\theta = \phi$ I decided to choose r = s in such a way that z(1/2) was precisely on the relevant circle, hoping that the curve between 0 and 1/2 and between 1/2 and 1 wouldn't veer too far away. Well, this turned out to work extremely well: A little calculation, done with the help of a computer, (thanks are due to the developers of the computer algebra system called MACSYMA at MIT, and to the ARPA network which makes this system available for research work) showed that the maximum deviation from a true circle occurs at the points $t = (3 \pm \sqrt{3})/6$, and the relative error is negligibly small. For example, if we take four points equally spaced at distance 1 from some center, the spline curve defined by these points in the above manner stays between distance 1 and distance $71/54 - 2\sqrt{2/9} < 1.00055$ from the center, an error of less than one part in a thousand: if there are 8 points, the maximum error is less than four parts per million; and if there are n points, the maximum error goes to zero as $1/n^6$.

(Changing the notation slightly, let

$$z(t) = 1 + (e^{i\theta} - 1)(3t^2 - 2t^3) + 4it(1-t)(1-t-e^{i\theta}t)\left(\sin\frac{\theta}{2}\right)/\left(1+\cos\frac{\theta}{2}\right)$$

and $f(t) = |z(t)|^2$. Then

$$f'(t) = 8\sin^2\frac{\theta}{2}\left(\frac{\cos\frac{\theta}{2}-1}{\cos\frac{\theta}{2}+1}\right)^2 (t-1)t(2t-1)(6t^2-6t+1)$$

and

$$\max_{0\le t\le 1}|z(t)| = \left|z\left(\frac{3-\sqrt{3}}{6}\right)\right| = 1 + \frac{\theta^6}{55296} + \frac{\theta^{10}}{106168320} + \cdots,$$

while $\min_{0 \le t \le 1} |z(t)| = z(0) = z(1/2) = z(1) = 1$. The "two-point circle" has max $|z(t)| = \sqrt{28/27} = 1.01835$, the three-point circle has max $|z(t)| = \sqrt{325/324} = 1.001542$, and the eight-point circle has max $|z(t)| = 1.0000042455$.)

Another case when a natural way to choose r and s suggests itself is when $0 + \phi = 90^\circ$; then the curve z(t) should be nearly the same as an ellipse having the endpoints on its axes. So far therefore I knew that I wanted

$$r = \frac{4}{1+\cos\theta}, \quad s = \frac{4}{1+\cos\phi} \qquad \text{when } \theta = \phi;$$

$$r = \frac{4\cos\theta}{(1+\cos 45^\circ)(\cos 45^\circ)}, \quad s = \frac{4\cos\phi}{(1+\cos 45^\circ)(\cos 45^\circ)} \qquad \text{when } \theta + \phi = 90^\circ$$

So I tried the formulas

So I tried the formulas

$$r = \frac{4\cos 0}{\left(1+\cos\frac{\theta+\phi}{2}\right)\left(\cos\frac{\theta+\phi}{2}\right)}, \quad s = \frac{4\cos\phi}{\left(1+\cos\frac{\theta+\phi}{2}\right)\left(\cos\frac{\theta+\phi}{2}\right)}$$

which fit both cases. However, this didn't give satisfactory results, especially when $0 + \phi$ approached 180°. My second attempt was

$$r = \frac{4\sin\phi}{\left(1+\cos\frac{\theta+\phi}{2}\right)\left(\sin\frac{\theta+\phi}{2}\right)}, \quad s = \frac{4\sin 0}{\left(1+\cos\frac{\theta+\phi}{2}\right)\left(\sin\frac{\theta+\phi}{2}\right)}$$

and this has worked very well. Fig. 7 shows the spline curves that result from the above approach when $\phi = 60^\circ$ and when θ varies from 0° to 120° in 5° steps.

It can be proved that if θ and ϕ are nonnegative and less than 180°, the cubic curve z(t) I have defined will never cross the striaght lines at angles θ and ϕ that meet the endpoints 0 and 1 respectively. This is a valuable property in type design, since it can be used to guarantee that the curve won't get out of bounds. However, I found that it also led to unsatisfactory curves when one of θ or ϕ was very small and the other was not, since this meant that the curve z(t) would be very close to a straight line yet it would enter that line from outside at a rather sharp angle. In fact, the angle θ is not infrequently zero, and this forces a straight line and a sharp corner at the right endpoint. Therefore I

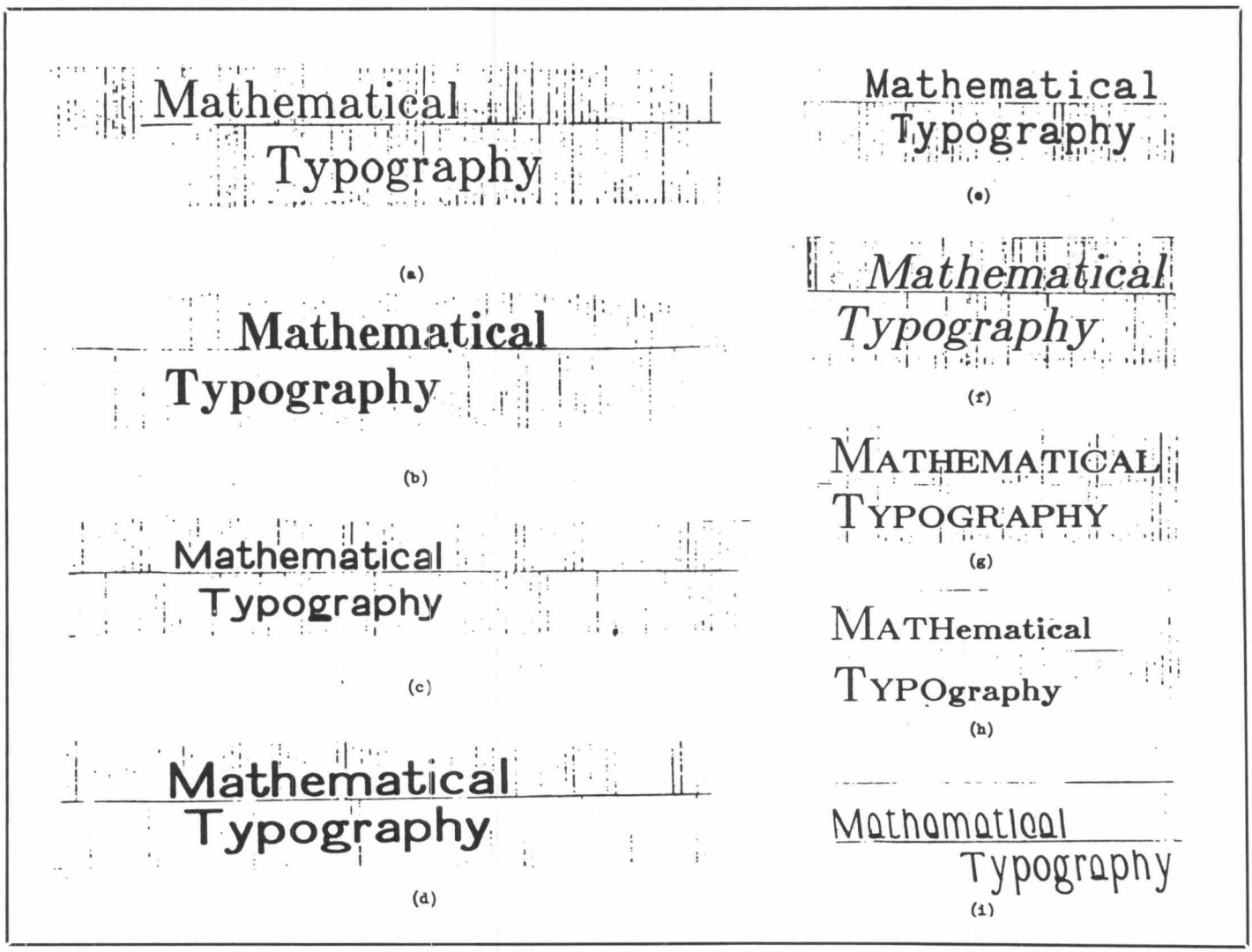


Figure 13. Different styles of type: (a) conventional "modern" font; (b) corresponding boldface; (c) sans serif font; (d) boldface sans serif; (e) typewriter imitation with fixed width letters; (f) slanted letters; (g) caps and small caps; (h) "small lower case"; (i) variations.

design each size of type individually. I'm not claiming that Fig. 14 shows the best way for the proportions to vary, it will take further experimentation before I have a good idea of what is desirable. The point I wish to make is that the alteration of type sizes for subscripts and so on is not as simple as it might seem at first, but a system like METAFONT will be able to vary the parameters quite readily, and visual experiments on different parameter settings can be carried out quickly. It used to take months for a type designer to make his drawings and have them converted to metal molds before he could see any proofs. One of the results was that there simply wasn't time to give proper attention to all the mathematical symbols and Greek letters, etc., as well as to the more common symbols, so a printer of mathematics had to make to with a hodgepodge of available characters in different sizes. (For example, he was often obliged to use different styles of letters in subscript positions, as we have seen.) Under the approach I am recomending, we automatically get consistency of all the symbols whenever the parameters change.

FROM CONTINUOUS TO DISCRETE.

The METAFONT system must not only define the characters in the continuum on the plane, it must also express them in terms of a discrete raster. Such squaring off of letters on graph paper has a long history, going back far before the invention of computers or television; for example, we all can remember seeing cross-stitch embroidery samplers from the nineteenth century. The same idea on a finer scale has been used in tapestries for many centuries: In our home library, my wife found the example of Fig. 15 which was woven in the northern part of Norway about 1500; this shows the name of St. Thomas in a style imitating contemporary

Figure 14. Adjusting the letters to coarser rasters.

changed the formulas by making sure that both r and s are always 1/2 or greater unless special exceptions are made; furthermore I never let r or s exceed 4. Fig. 8 shows the spline curves obtained under the same conditions as Fig. 7, but with s set to 1/2 if the above formula calls for any smaller value.

Using these techniques we obtain a system for drawing reasonably nice curves, if not the most pleasing ones, and it is especially good at circles. If the method gives the wrong tangent direction at some point, you can control this by specifying two points very close together having the desired slope. I have also included another way to modify the standard tangent directions, intended to make the system as good at drawing ellipses as it is at drawing circles: Before computing the splines I first shrink the entire figure in the vertical direction by muliplying all the y coordinates by a given aspect ratio (normally 1); then the splines are calculated, and the resulting shrunken curves are stretched out again by dividing the y coordinates by the aspect ratio.

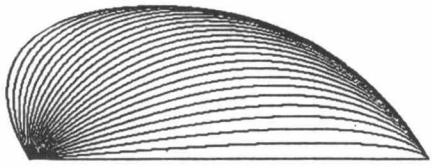


Figure 7. Spline curves with $0 = 0°$ (5) $120°$ and $\phi = 60°$

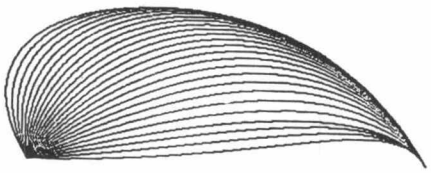


Figure 8. Same as Figure 7 but adjusted so that $r' = \min(1/2, r)$, $s' = \min(1/2, s)$.

APPLICATION TO TYPE DESIGN.

Now let's take a closer look at what can be drawn with a mathematical system like this. I suppose the natural thing to show you would be the letters A to Z; but since this is a mathematical talk, let's consider the digits 0 to 9 instead. (See Fig. 9.) Incidentally, the way I have arranged these numerals illustrates a fundamental distinction between a mathematician and a printer: the mathematician puts 0 next to the 1, but the printer always puts it next to the 9.

Most of these digits are drawn by using another idea taken from the history of typography, namely to imitate the calligrapher who uses pen and ink. Consider first the numeral '3', for example: The computer program which drew this symbol in Fig. 9 can be paraphrased

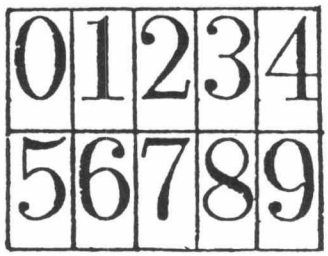


Figure 9. Digits 0 to 9 drawn by the METAFONT programs. (Further refinements to these characters will be made before the font has its final form.)

as follows. "First draw a dot whose left boundary is 1/6 of the way from the left edge to the right edge of the type and whose bottom boundary is 3/4 of the way from the top to the bottom of the desired final shape. Then take a hairline pen and, starting at the left of the dot, draw the upward arc of an ellipse; after reaching the top, the pen begins to grow in width, and it proceeds downward in another ellipse in such a way that the maximum width occurs on the axis of the ellipse, with the right edge of the pen 8/9 of the way from the left edge to the right edge of the type. Then the pen width begins to decrease to its original size again as the pen traverses another ellipse taking it down to a position 48% of the way from the top to the bottom of the desired final shape. . . ."

Notice that instead of describing the boundary of the character, as the renaissance geometers did, my METAFONT system describes the curve traveled by the *center* of the *pen*, and the shape of this pen is allowed to vary as the pen moves. The main advantage of this approach is that the same definition readily yields a family of infinitely many related fonts of type, each font being internally consistent. The change in pen size is governed by cubic splines in a manner analogous to the motion of the pen's center. In order to define the 20 or so different type fonts used in various places in my books, I need for the most part to use only three kinds of pens, namely (i) a circular pen, used for example to draw dots and at the base of the numeral '7'; (ii) a horizontal pen, whose shape is an ellipse, the width being variable but the height being constantly equal to the height of a hairline pen – such a pen is used most of the time, and in particular to draw all of the numeral '3' except for the dots; (iii) a vertical pen, analogous to the horizontal one, used for example to draw the strokes at the bottom of the '2' and at the top of the '5' and the '7'. For the fonts I am using, it was not necessary to use an oblique pen (i.e., an ellipse that is tilted on its side) except to make the tilde accent for Spanish n's; but to produce fonts of type analogous to Times Roman, an oblique pen would of course be used. If this system were to be extended to Chinese and Japanese characters, I believe it would be best to add another degree of freedom to the pen's motion, allowing an elliptical pen shape to rotate as well as to change its width.

The digit '4' shows another aspect of the METAFONT system. Although this character is fairly simple, consisting entirely of straight lines, notice that the thick line has to be cut off at an angle at the top. In order to do this, there are *erasers* as well as pens. First the computer draws a thick line all the way from top to bottom, like the upper case letter 'I', then it takes an eraser which erases everything to its left and comes down the diagonal stroke, then it takes a hairline pen and finishes the diagonal stroke. Such an eraser is used also at the top of the '1' and the bottom of the '2', etc.

Sometimes a simple spline seems to be inadequate to describe the proper growth of pen width, so in a few cases I had to resort to describing the left and right edges of the pen as separate curves, to be filled in afterwards. This occurs for example in the main stroke of the numeral '2', whose edges are defined by two splines having a specified tangent at the bottom and having vertical slope at the right of the curve.

With these techniques I found that it was possible to define a decent-looking complete font, containing a total of 128 characters, in about two months, although it will still be necessary of course to do fine tuning when more trail pages are typeset. (See Fig. 10.) The most difficult symbol by far, at least for me, was the letter S (and the numeral 8, which uses the same procedure); in fact I spent three days and nights without sleep, trying to make the S look right, before I got it. At one point I even felt it would be easier to rewrite all my books without using any S's! After the first day of discouraging trials, I showed what I had to my wife, and she said, "Why don't you make it S-shaped?"

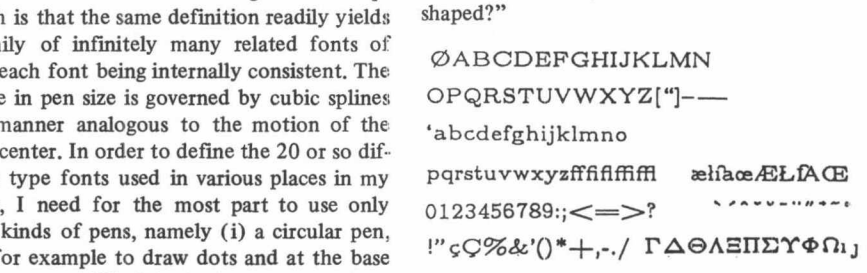


Figure 10. A font of 128 characters defined by METAFONT with standard pen settings. (The accent characters will be appropriately raised and centered over other letters when used by TEX.)

Fig. 11 shows how this problem was solved by Pacioli, Torniello, Palatino, and the French academicians; but the letter doesn't look like a modern S. Furthermore I think the engraver of the French S cheated a little in rounding off some lines near the middle – perhaps he used a French curve. With my wife's assistance, I finally came up with a satisfactory solution, somewhat like those used in the sixteenth century but generalized to ellipses. Each boundary of each arc of my S curve is composed of an ellipse and a straight line, determined by (i) the locations of the beginning and

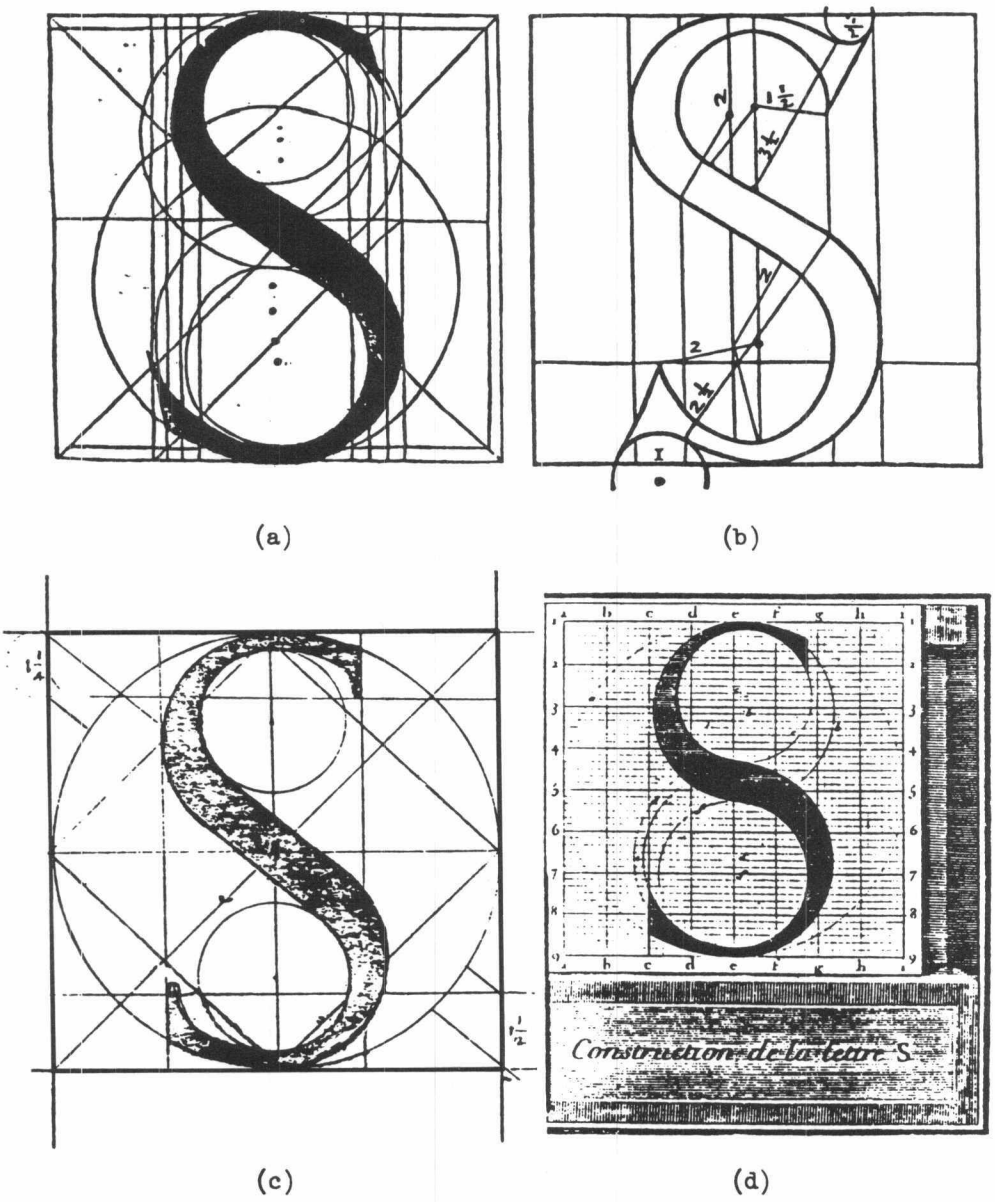


Figure 11. The letter S as defined by (a) Pacioli [42]; (b) Torniello [48]; (c) Palatino [43]; (d) French commission under Jaugeon [24].

ending points, (ii) the slope of the straight line, and (iii) the desired left extremity of the curve. It took me three hours to derive the necessary formulas, and I think Newton and Leibnitz would have enjoyed working on this problem. Fig. 12 shows various trial S's drawn by this scheme with different slopes; I hope you prefer the middle one, since it is the one I am actually using.

SSSSSSS

Figure 12 Different S's obtained by varying the slope in the middle. (This shows 1/2, 2/3, 3/4, 1, 4/3, 3/2, and 2 times the "correct" slope.)

FAMILIES OF FONTS.

To extend the METAFONT system, one essentially writes a computer program for the description of each character, in a special language intended for describing pen and eraser strokes. My colleague R. W. Gosper has observed that this is the opposite of *Sesame Street:* Instead of "This program was brought to you by the letter S" we have "This letter S was brought to you by a program." There are about 20 parameters to the program, telling how big a hairline pen is, how wide it whould be when drawing straight or curved stem lines, and specifying the sizes and proportions of various parts of the letters (the x-height, the heights of ascenders and descenders, the set width, the length of serifs, and so forth). By changing these parameters, we obtain infinitely many different styles of type, yet all of them are related and they seem to blend harmoniously with each other.

For example, Fig. 13 shows some of the possibilities. In Fig. 13a we have a conventional "modern" font in the tradition of Bodoni and Bell and "Scotch Roman". Then Fig. 13b shows a corresponding boldface, in which the hairlines are slightly larger and the stem lines are substantially wider. By making the hairlines and stem lines both the same size, and setting the serif length to zero, we obtain a sans serif font as shown in Fig. 13c. All of these examples are produced with the same programs defining the letter shapes; only the parameters are being varied. Actually the particular font shown in Fig. 13c will have a different style of g, because the descenders are especially short in this font, but I have shown this "g" in order to illustrate the parametric variations. Fig. 13d shows a boldface sans-serif style in which the pen has an oval shape wider than it is tall; I find this style especially pleasing, particularly because it came out by accident – I designed the programs only so that two or three different fonts would look right, all the others are free bonuses, and I had no idea that this one would be so nice.

With a suitable setting of the parameters, we can even imitate a typewriter with its fixed width letters, as shown in Fig. 13e. There is also a provision to slant the letters as in Fig. 13f; here the pen position is varied, but the actual shape of the pen is not being slanted, so circles remain circles.

Another setting of the parameters leads to caps and small caps as shown in Fig. 13g; small caps are drawn with the pens and heights ordinarily used for lower case letters, but controlled by the programs for upper case letters. Fig. 13h shows something printers have never seen before: this is what happens when you draw lower case letters in the small caps style, and we might call it "small lower case". It actually turns out to be one of the most pleasing fonts of all, except that the dots are too large.

Finally, Fig. 13i illustrates the variations you can get by giving weirder settings to the parameters.

When I was an assistant professor at Caltech, the math department secretaries used to send occasional "crank" visitors to my office, and I recall one time when a man came to ask if anybody had calculated the value of π "out to the end" yet. I tried to explain to him that π had been proved irrational, but this didn't seem to sink in, so finally I showed him a table of π to 100,000 decimals and told him that the expansion hadn't ended yet. I wish I could have had my typographical system ready at that time, so that I could have shown him Fig. 14!

Fig. 14 illustrates another principle of type design, namely that different sizes of type in the same style are not simply obtained from each other by optical transformations. The heights and widths and pen stroke sizes change at different rates, and a good typographer will

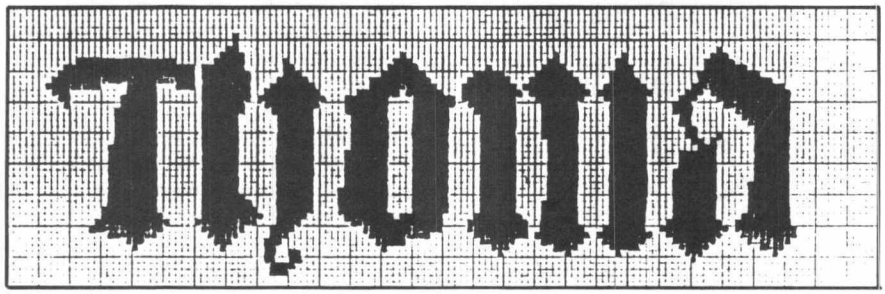


Figure 15. Lettering equivalent to this raster pattern appears in a Norwegian tapestry from Gildeskaal old church, woven about 1500 [22, p. 116].

calligraphy, and I'm sure that examples which antedate the printing press can be found elsewhere.

Fig. 16 shows how METAFONT produces the same letters from the same parameters but with different degrees of resolution in the raster. This digitalization process itself is considerably more difficult than it may seem at first, and some nontrivial mathematical concepts were needed before I could produce satisfactory results. In the first place, it is not sufficient merely to draw or to imagine drawing the character with infinite precision and then to "round" it by blacking in all the squares on graph paper that are sufficiently dark in the true image. One of the reasons this fails is that the three stem lines of the m, for instance, might be located in different relative positions with respect to the grid, so that the first stroke might round to three units wide (say) and the second might round to four. This would be quite unsatisfactory, as the eye quickly picks up such a variation in thickness, but it is avoided by METAFONT since the pen itself is first digitized and then the same digitized pen is used for all three strokes. Another problem is that those three strokes should be equally spaced; it would look bad if there were seven units between the first two and eight units between the last two, so the program for 'm' needs to round its points in such a way that this doesn't happen.

The process of digitizing the pen is not trivial either. Suppose, for example, we want a circular pen that is 2 raster units wide; the appropriate pen is clearly 2×2 square, which is the closest to a circle that we can come at this low degree of resolution. Now notice that we can't *center* a 2×2 square on any particular square, since none of the four squares is at its center; the same problem arises whenever we have to deal with a pen having even dimensions. One way to resolve this would be to insist on working only with odd numbers, but this would be far too limiting; so METAFONT uses a special rounding rule for the position of the pen's center. In general, suppose the pen is an ellipse of integer width w and integer height h; then if the pen is to be positioned at the real coordinates (x, y), its actual position on the discrete grid is taken to be

$$(\lfloor x - \delta(w) \rfloor, \lfloor y - \delta(h) \rfloor)$$

where $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x, and $\delta(\text{even}) = \frac{1}{2}$, $\delta(\text{odd}) = 0$. The pen itself, if positioned at the origin, would consist of all integers (x, y) which satisfy

$$\left(\frac{2(x-\delta(w))}{2}\right)^2 + \left(\frac{2(y-\delta(h))}{h}\right)^2 \le 1 + \max\left(\frac{2\delta(w)}{w}, \frac{2\delta(h)}{h}\right)^2$$

This formula—which incidentally is not the first one I tried—ensures that the discrete pen will indeed be w units wide and h units high, when w and h are positive integers. Fig. 17 shows the pens obtained for small w and h.

Still another problem appears when we want curved lines to look right. Fig. 18(a) shows a semicircle of radius 10 units, drawn with a pen of height 1 and width 3, when the right boundary of the pen falls exactly at an integer point; the pen sticks out terribly in one place. On the other hand if this right boundary falls just shy of an integer point, we get the curve in Fig. 18(b) which looks too flat. The ideal occurs in Fig. 18(c), when the right boundary occurs exactly midway between integers. Therefore the METAFONT programs adjust the location of curves to the raster before actually drawing the curves, forcing the favorable situation of Fig. 18(c); the actual shape of each letter changes slightly in order to adapt that letter to the desired raster size in a pleasant way.

There is yet another problem, which arises when the pen is growing in such a way that the edges of the curve it traces would be monotonic if the pen were drawn to infinite precision, yet the independent rounding of pen location and pen width causes this monotonicity to disappear. The problem arises only rarely, but when it does happen the eye immediately notices it. Consider, for example, the completely linear situation in Fig. 19, where each decrease by one unit in y is accompanied by an increase of .3 units in x and an increase of .2 units in the pen width w; the intended pen height is constant and very small, but in the discrete case the pen height is taken to be 1. The lightly shaded portion of Fig. 19 shows the true shape intended, but the darker squares show that the digitized form yields a non-monotonic left boundary. METAFONT compensates for this sort of problem by keeping track of the desired boundaries when the pen width is varying, plotting points twice (e.g. plotting both (x, y) and (x−1, y)) when necessary to keep the boundary correct. In other words, the idea of rounding the pen location and the width independently is sometimes effectively abandoned.

The final digitization problem that I needed to resolve was to make the left half of an "O" look like the mirror image of its right half, to make a left parenthesis look like the mirror image of a right parenthesis, and so on. This was done by having the METAFONT programs in such cases choose a center point that was either exactly at an integer or an integer plus ½, and to introduce dual rounding which could be proved to produce exactly the correct symmetry properties.

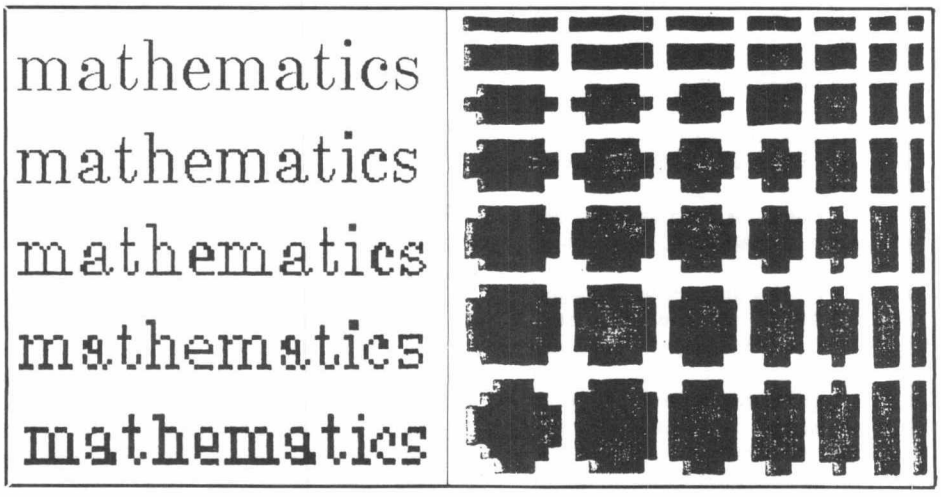


Figure 16. Adjusting the letters to coarser rasters.

Figure 17. Discrete "elliptical" pens of small integer width and height.

ALTERNATIVE APPROACHES

As I have said, I believe the METAFONT system is successful as a way to define letters and other symbols, but probably even better procedures can be devised with further research. Some of the limitations of my cubic splines are indicated in Fig. 20. Part (a) of that illustration shows a five-pointed star and the word "mathematics" in an approximation to

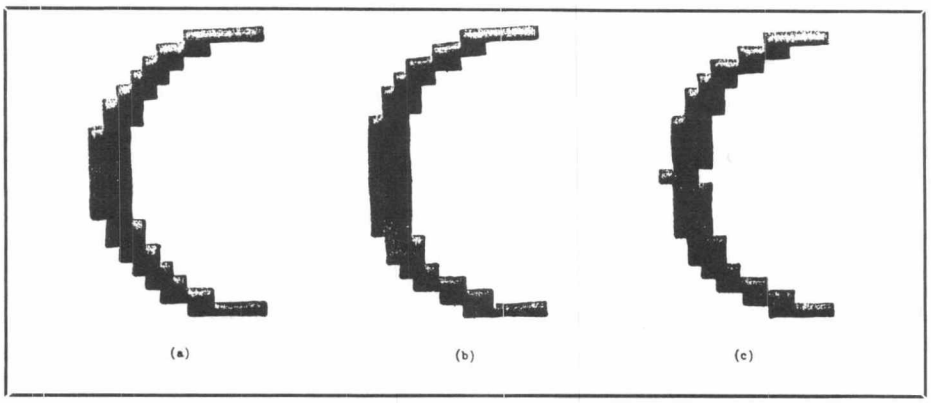


Figure 18. Difficulties of rounding an arc properly. (Three circles of radius 10 drawn with a 1 x 3 pen.)

my own handwriting, done with straight line segments so that you can see exactly what the data points are that I fed to my spline routine. Part (b) shows the way my handwriting might look when I get older; it was obtained by simply setting r = s = 2 in all the spline segments, therefore making clear what tangent angles are prescribed by the system. Part (c) is somewhat more disciplined, it was obtained by putting r = s = ½ everywhere. Fig. 20(d) is like Fig. 20(c) but drawn with a combined pen-and-eraser. Such a combination can lead to interesting effects, and the star here is my belated contribution to America's bicentennial.

When the general formulas for cubic splines are used as I explained above, we get Fig. 20(e) in which the star has become a very good approximation to a circle (as I said it would). In this illustration the pen is thicker and has a slightly oblique stress. Although my handwriting is inherently un-beautiful, there are still some kinks in Fig. 20(e) that could probably be ironed out if a different approach were taken.

The most interesting alternative from a mathematical standpoint seems to be to find a curve of given length that minimizes *the integral of the square of the curvature* with respect to arc length. This integral is proportional to the strain energy in a mechanical spline (in other words, a thin slat or beam) of the given length, going through the given points, so it seems to be an appropriate quantity to minimize. E. H. Lee and G. E. Forsythe [31] have reviewed early work on this variational problem, and shown that it is equivalent to having the spline at equilibrium with forces applied only at the given points of support. The Norwegian mathematician Evan Mehlum [36] has shown that if we specify a fixed arc length between consecutive points, the optimum curve will have linearly changing curvature of the form ax + by + c at point (x, y), and he has suggested choosing the constants by taking $b/a = (y_2 - y_1)/(x_2 - x_1)$ between (x_1, y_1) and (x_2, y_2), and requiring that slope and curvature be continuous across endpoints. Such an approach seems to require considerably more computation than the cubic splines recommended here, but it may lead to better curves, e.g. satisfying the extensionality property.

Another interesting approach to curve-drawing, which may be expecially useful for stimulating handwriting, is a "filtering" method suggested to me recently by Michael S. Paterson of the University of Warwick (unpublished). To get a smooth curve passing through points z_k, assuming that these points are about equally spaced on the desired curve, one simply writes

$$z(t) = \sum_k (-1)^k z_k f(t-k) / \sum_k (-1)^k f(t-k)$$

where f(t) is an odd function of order t^{-1} as $t \to 0$, decreasing rapidly away from zero, e.g. $f(t) = \text{csch } t = 2/(e^t - e^{-t})$.

Pen width and location	Rounded width and location
(3.5, 0.5,10.5)	(3, 0,10)
(3.7, 0.8, 9.5)	(3, 0, 9)
(3.9, 1.1, 8.5)	(3, 1, 8)
(4.1, 1.4, 7.5)	(4, 0, 7)
(4.3, 1.7, 6.5)	(4, 1, 6)
(4.5, 2.0, 5.5)	(4, 1, 5)
(4.7, 2.3, 4.5)	(4, 1, 4)
(4.9, 2.6, 3.5)	(4, 2, 3)
(5.1, 2.9, 2.5)	(5, 2, 2)
(5.3, 3.2, 1.5)	(5, 3, 1)
(5.5, 3.5, 0.5)	(5, 3, 0)

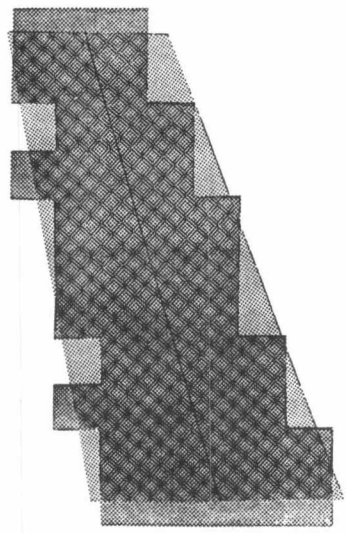


Figure 19. Failure of monotonicity due to independent rounding. (Rounding takes (w,x,y) into $(\lfloor w \rfloor, \lfloor x - \delta(\lfloor w \rfloor), \lfloor y \rfloor)$.)

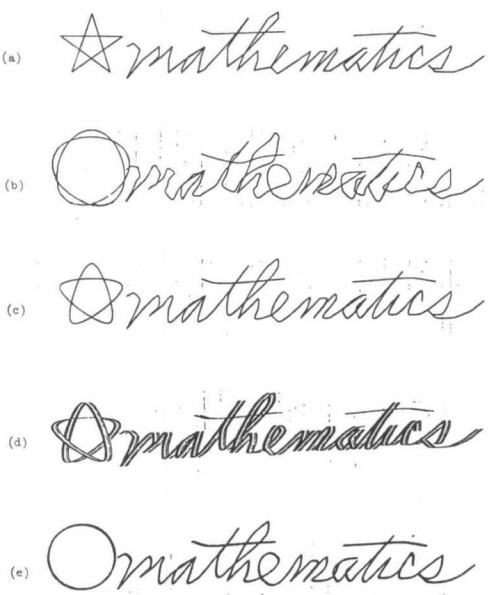


Figure 20. Examples of the cubic splines applied to sloppy handwriting.

I have not had time yet to experiment with Paterson's method or to attempt to harness it for the drawing of letters. It is easy to see that the derivative $z'(z_k) = f(1)(z_{k+1} - z_{k-1}) - f(2)(z_{k+2} - z_{k-2}) + \ldots$ lies approximately in the direction of $z_{k+1} - z_{k-1}$.

RANDOMIZATION.

In conclusion, I'd like to report on a little experiment that I did with random numbers. One might complain that the letters I have designed are too perfect, too much like a computer, so they lack "character." In order to counteract this, I can build a certain amount of randomness into the choices of where to put the pen when drawing each letter, and Fig. 21 shows what happens. The coordinates of key pen positions were chosen independently with a normal distribution and with increasing standard deviation, so that the third example has twice as much standard deviation as the second, the fourth has three times as much, and so on. Note that the two m's on each line (except the first) are different, and so are the a's and the t's, since each letter is randomly drawn.

After the deviation gets sufficiently large the results become somewhat ludicrous; and I don't want people to say that I ended this lecture by making a travesty of mathematics. So let us conclude by looking at Fig. 22, which shows what is obtained in various fonts when the degree of randomness is somewhat controlled. I think it can be said that the letters in this final example have a warmth and charm which makes it hard to believe that they were really generated by a computer following strict mathematical rules. Perhaps the reason the printing of mathematics looked so nice in the good old days was that the fonts of type were imperfect and inconsistent.

SUMMARY.

I'd like to summarize now by pointing out the moral of this long story. My experiences during the last few months vividly illustrate the fact that there are plenty of good mathematical problems still waiting to be solved, almost everywhere you look – especially in areas of life where mathematics has rarely been applied before. Mathematicians can provide solutions to these problems, receiving a double payoff – namely the pleasure of working out the mathematics, together with the appreciation of the people who can use the solutions. So let's go forth and apply mathematics in new ways.

ACKNOWLEDGMENTS.

I would like to thank my wife Jill for the many important suggestions she made to me during critical stages of this research; also Leo Guibas and Lyle Ramshaw for the help they provided in making illustrations at Xerox Palo Alto Research Laboratories; also Lester Earnest, Michael Fischer, Frank Liang, Tom Lyche, Albert Meyer, Michael Paterson, Michael Plass, Bob Sproull, Jean E. Taylor, and Hans Wolf, for helpful ideas and stimulating discussions and correspondence about this topic; also Gordon L. Walker, for verifying my conjectures about the printing history of the *Transactions* and for providing me with additional background information; also Prof. Dirk Siefkes for his help in acquiring Figs. 4(c) and 11(c), and the Berlin Kunstbibliothek for permission to publish them; and to Andre Jammes for permission to publish Figs. 6 and 11(d).

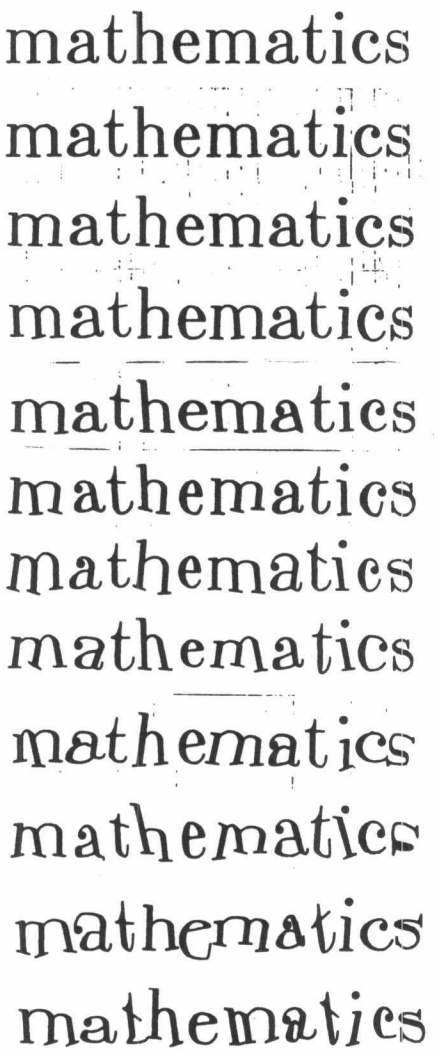


Figure21. Increasingly random pen positions; $\sigma = 0.1, \ldots,$.

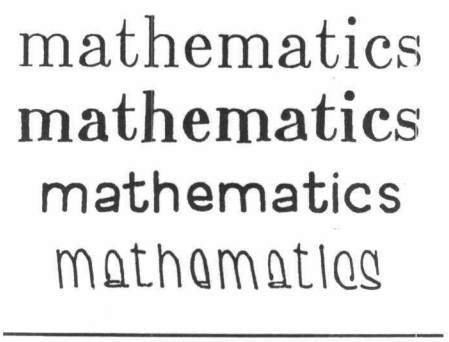


Figure 22. A bit of randomness introduced into various styles of type.

BIBLIOGRAPHY.

(I have included here several articles not referenced in the main text, namely a discussion of publishing at the American Institute of Physics [37]; some experiments in typesetting physics journals with the Bell Labs system [32], [7]; computer aids for technical magazine layout and editing together with a brief proposal for a standard typesetting language [11]; reports about early computer programs for character generation and mathematics composition [23], [30], [34], [35], [40], [46]; a description of the mathematics a traditional printer needs to know [9]; three standard references on the typesetting of mathematics [14], [47], [50]; some fonts of type and special characters designed by the American Math. Society [39]; a recent approach to mathematical definition of traditional type faces based on conic sections [15]; a proposal for a new way to control the spacing between letters based on somewhat mathematical principles [28]; and two purely mathematical papers inspired by typography [13], [21].)

1. A. V. Aho, S. C. Johnson, and J. D. Ullman, "Typesetting by ACM considered harmful," *Communications of the ACM* 18 (1975), 740.
2. American Mathematical Society, "Development of the Photon for efficient mathematical composition," Final report May 10, 1965), National Science Foundation grant G-21913; NTIS number PB168627.
3. American Mathematical Society, "Development of computer aids for tape-control of photocomposing machines," Report No. 2 (July 1967), "Extensions of the system of preparing a computer-processed tape to include the setting of multiple line equations," National Science Foundation grant GN-533; NTIS number PB175939.
4. American Mathematical Society, "Development of computer aids for tape-control of photocomposing machines," Final report, Section B (August 1968), "A system for computer-processed tape composition to include the setting of multiple line equations," National Science Foundation grant GN-533; NTIS number PB179418.
5. American Mathematical Society, "Development of computer aids for tape-control of photocomposing machines," Final report, Section C (January 1969), "Implementation, hardware, and other systems," National Science Foundation grant GN-533; NTIS number PB182088.
6. American Mathematical Society, "To complete the study of computer aids for tape-control of composing machines by developing an operating system," Final report, no. AMATHS-CAIDS-71-0 (April 1971), National Science Foundation grant GN-690; NTIS number PB200892.
7. American Physical Society, "APS tests computer system for publishing opera-

tions," *Physics Today* 30, 12 (December 1977), 75.

8. Donald M. Anderson, "Cresci and his capital alphabets," *Visible Language* 4 (1971), 331-352.
9. J. Woodard Auble, *Arithmetic for Printers,* second edition (Peoria, Ill.: Bennett, 1954).
10. Michael P. Barnett, *Computer Typesetting: Experiments and Prospects* (Cambridge, Mass.: M.I.T. Press, 1965).
11. Robert W. Bemer and A. Richard Shriver, "Integrating computer text processing with photocomposition," *IEEE Trans. on Prof. Commun.* PC-16 (1973), 92-96. This article is reprinted with another typeface and page layout in Robert W. Bemer, "The role of a computer in the publication of a primary journal," *Proc. IFIPS Nat. Comput. Conf.* 42, Part II (1973), M16-M20.
12. Peter J. Boehm, "Software and hardware considerations for a technical typesetting system," *IEEE Trans. on Prof. Commun.* PC-19 (1976), 15-19.
13. J. A. Bondy, "The 'graph theory' of the Greek alphabet," *Graph Theory and Applications,* Y. Alavi et al., eds. (Berlin: Springer, 1972), 43-54.
14. Theodore William Chaundy, Percy Reginald Barrett, and Charles Batey, *The Printing of Mathematics* (Oxford: Oxford University Press, 1954).
15. P. J. M. Coueignoux, "Generation of Roman Printed Fonts," Ph.D. thesis, Dept. of Electrical Engineering, Mass. Inst. of Technology (June, 1975).
16. Giovanni Francesco Cresci Milanese, *Essemplare de Piv Sorti Lettere* (Rome, 1560). Also edited and translated by Arthur Sidney Osley (London, 1968).
17. T. L. DeVinne, *The Practice of Typography: Modern Methods of Book Composition* (New York: Oswald, 1914).
18. Albrecht Dürer, *Underweysung der Messung mit dem Zirckel und Richtscheyt* (Nuremberg, 1525). An English translation of the section on alphabets has been published as Albrecht Dürer, *Of the Just Shaping of Letters,* R. T. Nichol, trans. (Dover, 1965).
19. Felice Feliciano Veronese, *Alphabetum Romanum,* Giovanni Mardersteig, ed. (Verona: Editiones Officinae Bodoni, 1960).
20. Frederic W. Goudy, *Typologia: Studies in Type Design and Type Making with Comments on the Invention of Typography, the First Types, Legibility and Fine Printing* (Berkeley, Calif.: Univ. of California Press, 1940).
21. F. Harary, "Typographs," *Visible Language* 7 (1973), 199-208.
22. Roar Hauglid, Randi Asker, Helen Engelstad, and Gunvor Traetteberg, *Native Art of Norway* (Oslo: Dreyer, 1965).
23. A. V. Hershey, "Caligraphy for computers," NWL Report No. 2101 (Dahlgren, Va.: U.S. Naval Weapons Laboratory, August 1967); NTIS number AD662398.
24. André Jammes, *La Réforme de la Typographie Royale sous Louis XIV* (Paris: Paul Jammes, 1961).
25. Paul E. Justus, "There is more to typesetting than setting type," *IEEE Trans. on Prof. Commun.* PC-15 (1972), 13-16.
26. Alan C. Kay, "Microelectronics and the personal computer," *Scientific American* 237, 3 (September 1977), 230-244.
27. Brian W. Kernighan and Lorinda L. Cherry, "A system for typesetting mathematics," *Communications of the ACM* 18 (1975), 151-157.
28. David Kindersley, *Optical Letter Spacing for New Printing Systems* (London: Wynkyn de Worde Society, 1976).
29. Donald E. Knuth, "Tau Epsilon Chi, a system for technical text," manuscript in preparation.
30. Dorothy K. Korbuly, "A new approach to coding displayed mathematics for photocomposition," *IEEE Trans. on Prof. Commun.* PC-18 (1975), 283-287.
31. E. H. Lee and G. E. Forsythe, "Variational study of nonlinear splines," *SIAM Review* 15 (1973), 120-133.
32. M. E. Lesk and B. W. Kernighan, "Computer typesetting of technical journals on UNIX," Computing Science Tech. Report 44, (Murray Hill, N.J.: Bell Laboratories, June, 1976).
33. Giovanni Mardersteig, *The Alphabet of Francesco Torniello (1517) da Novara* (Verona: Officina Bodoni, 1971).
34. M. V. Mathews and Joan E. Miller, "Computer editing, typesetting, and image generation," *Proc. AFIPS Fall Joint Computer Conf.* 27 (1965), 389-398.
35. M. V. Mathews, Carol Lochbaum, and Judith A. Moss, "Three fonts of computer drawn letters," *Communications of the ACM* 10 (1967), 627-630.
36. Even Mehlum, "Nonlinear Splines," in *Computer Aided Geometric Design,* Robert E. Barnhill and Richard F. Riesenfeld, eds. (New York: Academic Press, 1974), 173-207.
37. A. W. Kenneth Metzner, "Multiple use and other benefits of computerized publishing," *IEEE Trans. on Prof. Commun.* PC-18 (1975), 274-278.
38. Joseph Moxon, *Regulae Trium Ordinum Literarum Typographicarum, or the Rules of the Three Orders of Print Letters: viz. the {Roman, Italick, English} Capitals and Small; Shewing how they are compounded of Geometrick Figures, and mostly made by Rule and Compass* (London, Joseph Moxon, 1676).
39. Phoebe J. Murdock, "New alphabets and symbols for typesetting mathematics," *Scholarly Publishing* 8 (1976), 44-53. Reprinted in *Notices Amer. Math. Soc.* 24 (1977), 63-67.
40. Nicholas Negroponte, "Raster scan approaches to computer graphics," *Computers and Graphics* 2 (1977), 179-193.
41. Wolfgang A. Ocker, "A program to hyphenate English words," *IEEE Trans. on Prof. Commun.* PC-18 (1975), 78-84.
42. Luca Pacioli, *Divina Proportione, Opera a tutti glingegni perspicaci e curiosi necessaria Ove ciascun studioso di Philosophia, Prospectiva, Pictura, Sculptura: Architecturo: Musice: altre Mathematice: suavissima: sottile: e admirabile et doctrina consequira: e delectarassi: con varie questione de secretissima scientia* (Venice, 1509).
43. Giovanbattista Palatino Cittadino Romano, *Libro Primo del le Lettere Maiuscole Antiche Romane* (unpublished), Berlin Kunstbibliothek MS. OS5280. Some of the individual pages are dated 1543, 1546, 1549, 1574, or 1575. See James Wardrop, "Civis Romanus Sum: Giovanbattista Palatino and his circle," *Signature,* n.s. 14 (1952), 3-39.
44. Paul A. Parisi, "Composition innovations of the American Society of Civil Engineers," *IEES Trans. on Prof. Commun.* PC-18 (1975), 244-273.
45. R. G. D. Richardson, "The twenty-ninth annual meeting of the Society," *Amer. Math. Soc. Bulletin* 29 (1923), 97-116. (See also vol. 28 (1922) pp. 234-235, 378 for comments on the special *Transactions* volume, and pp. 2-3 of vol. 28 for discussion of deficits due to increased cost of printing.)
46. Glenn E. Roudabush, Charles R. T. Bacon, R. Bruce Briggs, James A. Fierst, Dale W. Isner, and Hiroshi A. Noguni, "The left hand of scholarship: Computer experiments with recorded text as a communication media," *Proc. AFIPS Fall Joint Computer Conf.* 27 (1965), 399-411.
47. Ellen E. Swanson, *Mathematics Into Type* (Providence, R.I.: Amer. Math. Soc., 1971).
48. Francesco Torniello, *Opera del Modo de Fare le Littere Mauiscole Antique* (Milan, 1517).
49. Geofroy Tory, *Champ Fleury* (Paris, 1529). Also translated into English and annotated by George B. Ives (New York: Grolier Club, 1927).
50. Karel Wick, *Rules for typesetting mathematics,* Tr. by V. Boublik and M. Hejlová (The Hague: Mouton, 1965).
51. Hermann Zapf, *About Alphabets: Some Marginal Notes on Type Design* (Cambridge, Mass.: M.I.T. Press, 1970).

SYMBOL: Generating Symbol Table Alpha Listing on the H8

BY WILLIAM W. MOSS
1507 Riverview Lane
Bradenton, FL 33505

Enclosed is a listing of the program 'SYMBOL' which generates alphabetically sorted listings of the symbol table from runs of the Heath H8 assembler. To utilize the program one simply has to run it immediately after a run of the assembler (i.e., before another program writes over the assembler symbol table in memory). The table is then printed out in column order on the console. The symbol values are listed in offset octal format. Hexadecimal format can also be used if desired by substituting the routine 'HEXOUT' for 'OCTOUT' and changing the code in the 'PRINT' routine as shown in the accompanying listing.

Of general interest to Intel 8080 or Zilog Z80 users is the inclusion of a routine (SMSORT) which performs the alphabetical sorting of variable length character strings utilizing the efficient Shell-Metzner algorithm. The speed of this routine is further increased because the strings in the data table are not physically swapped during the sort: 2-byte pointers are exchanged instead. Because the string data table is unchanged and only the number of characters specified in the length byte are actually compared for sorting purposes, accessory data can follow each element of the string table without affecting the sorting. In this program the symbol's value follows the string itself. There is no software limit to the number of strings which can be sorted, as all variable in the sort routine are 16-bit values, and thus the real limitation is the addressing space of the hardware (65 kilobytes). The limit for the number of bytes which can be compared from each string is 255 since one byte is allowed for this value. The initialization requirements are given in the listing of the routine. At the end of the sort the table containing the pointer addresses to the string data has been rearranged such that the first address is the pointer to the first alphabetically ranked string, etc., down to the alphabetically lowest string.

The program is written with standard Intel 8080 assembly language mnemonics except for the use of four Heath DOS calls (SCALL) which are defined as follows:

(1) .SETTOP sets upper memory limit in HDOS.
(2) .SCOUT outputs the character in register 'A' to console.
(3) .ERROR prints out system error messages.
(4) .EXIT returns to system monitor.

As an example of the program's output the symbol table for the program itself is listed.

```
>ASM

HDOS ASsemBler Issue #104.02.00.

*,-SYMBOL


00353 Statements Assembled
11926 Bytes Free
 No Errors Detected

>SYMBOL

.ERROR  000,057   ERROR   044,064   READLBL 042,233   TABADDR 043,216
.EXIT   000,000   LBLTAB  044,110   ROWCNT  043,251   USEC    043,060
.SCOUT  000,002   NEWLBL  043,317   S.OMAX  040,324   USERFWA 042,200
.SETTOP 000,052   NEWLINE 043,262   SETI    043,014   VARI    044,074
ALLDONE 044,070   NEXTLBL 042,310   SETJ    043,162   VARJ    044,076
ASMTAB  075,136   NOMATCH 043,101   SETL    043,022   VARK    044,100
COLUMNS 000,004   OCTOUT  044,040   SETM    042,350   VARL    044,102
COMP    043,062   POINT   042,276   SMSORT  042,340   VARM    044,104
CRLF    000,012   POINTAB 044,072   SPACEB  044,027   VARN    044,106
DEMHL   043,207   PRINT   043,236   STROUT  043,332
DIGOUT  044,044   READCHR 042,237   SYMVAL  043,226


SYMBOL TABLE SORTER AND PRINTER FOR H8 ASSEMBLER        HEATH ASM #104.02.00.
  WILLIAM W. MOSS - 1 JUNE, 1979

                        * PROGRAM TO PRINT OUT AN ALPHABETICALLY SORTED SYMBOL TABLE FOLLOWING USE
                        * OF THE HEATH H8 ASSEMBLER (VERSION 104.02.00).
                        *
                        * TO USE: ASSEMBLE AS USUAL WITH THE HEATH 'ASM' PROGRAM,
                        *       IMMEDIATELY AFTER THE ASSEMBLY IS COMPLETED RUN THIS PROGRAM.
                        *       THE SYMBOL TABLE IS THEN PRINTED ON THE CONSOLE.

                        * DEFINITIONS:
                        *
000,004                 COLUMNS EQU     4       NUMBER OF COLUMNS DESIRED IN TABLE LISTING
075,135                 ASMTAB  EQU     75136A  FWA SYMBOL TABLE FOR ASM 104.02.00
000,012                 CRLF    EQU     10      CARRIAGE RETURN / LINE FEED CHARACTER IN HDOS
040,324                 S.OMAX  EQU     40324A  SYSTEM OVERLAY MAXIMUM SIZE
042,200                 USERFWA EQU     42200A  START OF USER MEMORY IN HDOS

                        * HEATH DOS SYSTEM CALL DEFINITIONS:
```

```
                        *
000.052                 .SETTOP EQU     52Q
000.002                 .SCOUT  EQU     2
000.057                 .ERROR  EQU     57Q
000.000                 .EXIT   EQU     0

042.200                         ORG     USERFWA

                        * ROUTINE TO SET MAXIMAL USER HIGH MEMORY LIMIT IN HDOS
042.200  041 377 377            LXI     H,-1    REQUEST ALL OF MEMORY (I.E. TOO MUCH)
042.203  377 052                SCALL   .SETTOP
042.205  353                    XCHG
042.206  052 324 040            LHLD    S.OMAX  HL = OVERLAY SIZE
042.211  001 012 000            LXI     B,10
042.214  011                    DAD     B       HL = OVERLAY PLUS MARGIN FOR ERROR
042.215  315 207 043            CALL    DEMHL   HL = MAX MEM - OVERLAY - 10
042.220  377 052                SCALL   .SETTOP REQUEST MEMORY LIMIT
042.222  332 064 044            JC      ERROR   JUST IN CASE SOMETHING AMISS

                        * ROUTINE TO MOVE AND RESTRUCTURE THE SYMBOL TABLE
                        * FINAL STRUCTURE:
                        *       BEGIN AT 'LBLTAB'.
                        *       DATA UNIT FOR EACH SYMBOL:
                        *               (1) LENGTH OF SYMBOL CHARACTER STRING ('N') - 1 BYTE
                        *               (2) STRING DATA - 'N' BYTES
                        *               (3) SYMBOL'S VALUE - 2 BYTES - STORED IN LOW/HIGH FORM
                        *
042.225  041 136 075            LXI     H,ASMTAB
042.230  021 110 044            LXI     D,LBLTAB
042.233  325            READLBL PUSH    D       SAVE ADDR FOR STRING BYTE COUNT
042.234  023                    INX     D
042.235  016 000                MVI     C,0
042.237  176            READCHR MOV     A,M
042.240  107                    MOV     B,A     SAVE CHARACTER
042.241  247                    ANA     A
042.242  312 276 042            JZ      POINT   IF END OF TABLE REACHED
042.245  346 177                ANI     177Q    CLEAR HIGH BIT
042.247  022                    STAX    D
042.250  023                    INX     D
042.251  043                    INX     H
042.252  014                    INR     C
042.253  270                    CMP     B       SEE IF HIGH BIT WAS SET
042.254  312 237 042            JZ      READCHR
042.257  043                    INX     H       SKIP CODE BYTE
042.260  176                    MOV     A,M
042.261  022                    STAX    D       SAVE LOW BYTE
042.262  023                    INX     D
042.263  043                    INX     H
042.264  176                    MOV     A,M
042.265  022                    STAX    D       SAVE HIGH BYTE
042.266  023                    INX     D
042.267  043                    INX     H
042.270  343                    XTHL            HL = ADDRESS FOR BYTE COUNT
042.271  161                    MOV     M,C     STORE BYTE COUNT
042.272  341                    POP     H
042.273  303 233 042            JMP     READLBL

                        * ROUTINE TO CONSTRUCT A POINTER TABLE FOR THE STRING DATA TABLE.
                        * STRUCTURE AS FOLLOWS:
                        *       BEGIN AT ('POINTAB').
                        *       EACH TWO BYTE DATA ITEM IS THE ADDRESS OF THE START OF DATA
                        *       FOR EACH SYMBOL IN THE STRING DATA TABLE.
                        *
042.276  341            POINT   POP     H       HL = (LWA OF LBLTAB) +1
042.277  160                    MOV     M,B     MARK END OF LBLTAB WITH 0  (BC = 0 ON ENTRY)
042.300  043                    INX     H
042.301  042 072 044            SHLD    POINTAB
042.304  353                    XCHG
042.305  041 110 044            LXI     H,LBLTAB
042.310  176            NEXTLBL MOV     A,M
042.311  247                    ANA     A       END OF LBLTAB?
042.312  312 340 042            JZ      SMSORT
042.315  175                    MOV     A,L
042.316  022                    STAX    D
042.317  023                    INX     D
042.320  174                    MOV     A,H
042.321  022                    STAX    D
042.322  023                    INX     D
042.323  003                    INX     B       COUNT LABELS IN TABLE
042.324  176                    MOV     A,M
042.325  306 003                ADI     3       A = LENGTH OF LABEL + 3 DATA BYTES
042.327  205                    ADD     L
042.330  157                    MOV     L,A
042.331  174                    MOV     A,H
042.332  316 000                ACI     0
042.334  147                    MOV     H,A     HL = HL + A
042.335  303 310 042            JMP     NEXTLBL

                        * SHELL-METZNER STRING SORTING ROUTINE
                        *       ENTRY:  BC = NUMBER OF STRINGS TO BE SORTED
                        *               ('POINTAB') = START OF POINTER TABLE
                        *                  POINTER TABLE MADE UP OF A SERIES OF 16 BIT ADDRESSES
                        *                  POINTING TO THE POSITION IN THE DATA TABLE FOR EACH
                        *                  OF THE ITEMS TO BE SORTED.
                        *               DATA TABLE STRUCTURE:
                        *                  FIRST BYTE = LENGTH OF STRING
                        *                  FOLLOWING BYTES = STRING CHARACTERS
                        *                  BYTES BEYOND SPECIFIED LENGTH ARE IGNORED
                        *       EXIT:   POINTER TABLE REARRANGED TO POINT TO STRINGS IN ALPHABETICAL ORDER.
                        *               STRING DATA TABLE IS UNCHANGED.
                        *               VARN = NUMBER OF STRINGS SORTED.
                        *
042.340  140            SMSORT  MOV     H,B
042.341  151                    MOV     L,C
042.342  042 106 044            SHLD    VARN
042.345  042 104 044            SHLD    VARM
042.350  052 104 044 SETM       LHLD    VARM
042.353  174                    MOV     A,H
042.354  247                    ANA     A       CLEAR CARRY
```

```
042,355  037                    RAR
042,356  147                    MOV     H,A
042,357  175                    MOV     A,L
042,360  037                    RAR
042,361  157                    MOV     L,A
042,362  042 104 044           SHLD    VARM    VARM = VARM / 2
042,365  264                    ORA     H
042,366  312 236 043           JZ      PRINT   IF END OF SORT
042,371  041 001 000           LXI     H,1
042,374  042 076 044           SHLD    VARJ    VARJ = 1
042,377  052 106 044           LHLD    VARN
043,002  353                    XCHG
043,003  052 104 044           LHLD    VARM
043,006  315 207 043           CALL    DEMHL
043,011  042 100 044           SHLD    VARK    VARK = VARN - VARM
043,014  052 076 044 SETI      LHLD    VARJ
043,017  042 074 044           SHLD    VARI    VARI = VARJ
043,022  052 074 044 SETL      LHLD    VARI
043,025  353                    XCHG
043,026  052 104 044           LHLD    VARM
043,031  031                    DAD     D
043,032  042 102 044           SHLD    VARL    VARL = VARI + VARM
043,035  315 226 043           CALL    SYMVAL  HL = START OF DATA POINTED BY VARL
043,040  345                    PUSH    H
043,041  052 074 044           LHLD    VARI
043,044  315 226 043           CALL    SYMVAL
043,047  321                    POP     D
043,050  032                    LDAX    D       A = CHAR COUNT FOR STRING [VARL]
043,051  116                    MOV     C,M     C = CHAR COUNT FOR STRING [VARI]
043,052  271                    CMP     C
043,053  365                    PUSH    PSW     'C' SET IF LENSTRING [VARI] > LENSTRING [VARL]
043,054  322 060 043           JNC     USEC
043,057  117                    MOV     C,A
043,060  043           USEC    INX     H       C = LENGTH OF SHORTER STRING
043,061  023             .     INX     D       HL & DE POINT TO START OF LABEL STRINGS
043,062  032           COMP    LDAX    D
043,063  276                    CMP     M       COMPARE STRINGS
043,064  302 101 043           JNZ     NOMATCH IF NO MATCH
043,067  023                    INX     D
043,070  043                    INX     H
043,071  015                    DCR     C
043,072  302 062 043           JNZ     COMP
043,075  361                    POP     PSW     CHARACTERS THE SAME; SEE IF ONE STRING LONGER?
043,076  303 102 043           JMP     NOMATCH+1
043,101  341           NOMATCH POP     H       CLEAR STACK
043,102  322 162 043           JNC     SETJ    IF NO REARRANGEMENT REQUIRED
                       * SWITCH THE POINTER ADDRESS AT (VARI) WITH THAT AT (VARL)
043,105  052 074 044           LHLD    VARI
043,110  315 216 043           CALL    TABADDR
043,113  345                    PUSH    H       STACK = POINTER DATA FOR (VARI)
043,114  052 102 044           LHLD    VARL
043,117  315 216 043           CALL    TABADDR HL = POINTER DATA FOR (VARL)
043,122  321                    POP     D
043,123  032                    LDAX    D
043,124  107                    MOV     B,A
043,125  176                    MOV     A,M
043,126  022                    STAX    D
043,127  160                    MOV     M,B
043,130  023                    INX     D
043,131  043                    INX     H
043,132  032                    LDAX    D
043,133  107                    MOV     B,A
043,134  176                    MOV     A,M
043,135  022                    STAX    D
043,136  160                    MOV     M,B
                       * SWITCH COMPLETED
043,137  052 074 044           LHLD    VARI
043,142  353                    XCHG
043,143  052 104 044           LHLD    VARM
043,146  315 207 043           CALL    DEMHL
043,151  042 074 044           SHLD    VARI    VARI = VARI - VARM
043,154  053                    DCX     H
043,155  174                    MOV     A,H
043,156  247                    ANA     A
043,157  362 022 043           JP      SETL    IF VARI >= 1
043,162  052 076 044 SETJ      LHLD    VARJ
043,165  043                    INX     H
043,166  042 076 044           SHLD    VARJ    VARJ = VARJ + 1
043,171  353                    XCHG
043,172  052 100 044           LHLD    VARK
043,175  353                    XCHG
043,176  315 207 043           CALL    DEMHL
043,201  332 350 042           JC      SETM    IF VARJ > VARK
043,204  303 014 043           JMP     SETI

                       * DEMHL -- HL = DE - HL
                       *       ENTRY:  HL & DE = 16 BIT INTEGERS (UNSIGNED)
                       *       EXIT:   HL = DE - HL
                       *               DE = UNCHANGED
                       *               'C' SET IF HL > DE
                       *       USES:   A,F,H,L
043,207  173           DEMHL   MOV     A,E
043,210  225                    SUB     L
043,211  157                    MOV     L,A
043,212  172                    MOV     A,D
043,213  234                    SBB     H
043,214  147                    MOV     H,A
043,215  311                    RET

                       * TABADDR -- FIND LOCATION OF DATA POINTER IN TABLE
                       *       ENTRY: HL = INTEGER VALUE ( 1 -> N )
                       *       EXIT:  HL = ADDR OF TWO BYTE DATA POINTER ( FOR INPUT VALUE )
                       *       USES:  D,E,H,L,F
043,216  053           TABADDR DCX     H
043,217  051                    DAD     H       HL = [(ORIGINAL HL) - 1] * 2
043,220  353                    XCHG
043,221  052 072 044           LHLD    POINTAB
043,224  031                    DAD     D
```

```
043.225  311                    RET

                         * SYMVAL -- FIND DATA RELATED TO INPUT INTEGER VALUE IN ARRAY
                         *       ENTRY: HL = INTEGER VALUE ( 1 -> N ) POINTING TO DATA ARRAY
                         *       EXIT:  HL = FWA STRING DATA FOR THAT VALUE
                         *       USES:  A,D,E,H,L,F
043.226  315 216 043 SYMVAL  CALL    TABADDR
043.231  176                 MOV     A,M
043.232  043                 INX     H
043.233  146                 MOV     H,M
043.234  157                 MOV     L,A
043.235  311                 RET

                         * SORTING IS COMPLETED.  NOW PRINT OUT TABLE.
                         *
043.236  042 074 044 PRINT   SHLD    VARI    VARI = 0  (HL = 0 ON EXIT FROM SMSORT)
043.241  353                 XCHG            DE = 0
043.242  001 374 377         LXI     B,-COLUMNS  COMPLEMENT OF NUMBER OF COLUMNS
043.245  052 106 044         LHLD    VARN
043.250  053                 DCX     H
043.251  011           ROWCNT  DAD     B
043.252  023                 INX     D
043.253  332 251 043         JC      ROWCNT
043.256  353                 XCHG
043.257  042 104 044         SHLD    VARM    VARM = NUMBER OF ROWS IN TABLE
043.262  076 012       NEWLINE MVI     A,CRLF
043.264  377 002               SCALL   .SCOUT  CR/LF TO CONSOLE
043.266  076 004               MVI     A,COLUMNS  NUMBER OF COLUMNS TO PRINT
043.270  062 100 044         STA     VARK    VARK = NUMBER COLUMNS LEFT TO BE PRINTED
043.273  052 104 044         LHLD    VARM
043.276  353                 XCHG
043.277  052 074 044         LHLD    VARI
043.302  043                 INX     H
043.303  042 074 044         SHLD    VARI    VARI = LINE COUNT
043.306  042 076 044         SHLD    VARJ    VARJ = ARRAY INDEX VALUE
043.311  315 207 043         CALL    DEMHL   SUBTRACT VARI FROM VARM
043.314  332 070 044         JC      ALLDONE
043.317  052 076 044 NEWLBL  LHLD    VARJ
043.322  315 226 043         CALL    SYMVAL
043.325  116                 MOV     C,M     C = BYTE COUNT OF STRING
043.326  076 010             MVI     A,8     TAB 8 SPACES FOR LABEL VALUES
043.330  221                 SUB     C
043.331  107                 MOV     B,A     B = SPACES NEEDED FOR TAB
043.332  043           STROUT  INX     H
043.333  176                 MOV     A,M
043.334  377 002             SCALL   .SCOUT
043.336  015                 DCR     C
043.337  302 332 043         JNZ     STROUT
043.342  315 027 044         CALL    SPACEB
043.345  043                 INX     H
043.346  043                 INX     H
043.347  315 040 044         CALL    OCTOUT  PRINT HIGH BYTE
043.352  076 056             MVI     A,','
043.354  377 002             SCALL   .SCOUT
043.356  053                 DCX     H
043.357  315 040 044         CALL    OCTOUT  PRINT LOW BYTE
043.362  041 100 044         LXI     H,VARK
043.365  065                 DCR     M
043.366  312 262 043         JZ      NEWLINE
043.371  052 076 044         LHLD    VARJ
043.374  353                 XCHG
043.375  052 104 044         LHLD    VARM
044.000  031                 DAD     D
044.001  042 076 044         SHLD    VARJ
044.004  353                 XCHG
044.005  052 106 044         LHLD    VARN
044.010  353                 XCHG
044.011  315 207 043         CALL    DEMHL   OUTSIDE ARRAY BOUNDS?
044.014  332 262 043         JC      NEWLINE YES!
044.017  006 004             MVI     B,4
044.021  315 027 044         CALL    SPACEB  4 SPACES BETWEEN COLUMNS
044.024  303 317 043         JMP     NEWLBL

                         * SPACEB -- PRINT SPACES ON CONSOLE
                         *       ENTRY: B = NUMBER OF SPACES TO BE PRINTED
                         *       USES:  A,B
044.027  076 040       SPACEB  MVI     A,' '
044.031  377 002               SCALL   .SCOUT
044.033  005                   DCR     B
044.034  302 031 044           JNZ     SPACEB+2
044.037  311                   RET

                         * OCTOUT -- PRINT OCTAL VALUE OF BYTE AT M
                         *       ENTRY:  HL = ADDRESS OF BYTE TO BE OUTPUT
                         *       EXIT:   BYTE OUTPUT IN OCTAL TO CONSOLE
                         *       USES:   A,F,C
044.040  176           OCTOUT  MOV     A,M
044.041  247                   ANA     A       CLEAR CARRY
044.042  016 003               MVI     C,3
044.044  027           DIGOUT  RAL
044.045  027                   RAL
044.046  027                   RAL
044.047  365                   PUSH    PSW
044.050  346 007               ANI     7
044.052  306 060               ADI     60Q
044.054  377 002               SCALL   .SCOUT
044.056  361                   POP     PSW
044.057  015                   DCR     C
044.060  302 044 044           JNZ     DIGOUT
044.063  311                   RET

044.064  046 012       ERROR   MVI     H,CRLF
044.066  377 057               SCALL   .ERROR  PRINT OUT SYSTEM ERROR MESSAGE
044.070  377 000       ALLDONE SCALL   .EXIT   RETURN TO HDOS CONTROL

                         * DATA SECTION:
                         *
044.072                  POINTAB DS      2
```

(continued on page 38)

TI 9900 FULLY REENTRANT FORMATTER OPERATING UNDER TIPMX

PART 2

BY NORMAN V. WALTERS
Software Engineer
R&D Department
Fairfield Industries, Inc.
Houston, TX 77042

This is the concluding segment in a two-part series.

Listing

```
FRMTR          TXMIRA
TIPMX FORMATTING ROUTINE

                * TIPMX FORMATTING ROUTINE
                *      LAST REVISION 7/19/79  REVISION NO. 39
                *
                *
                *
                *      THIS PROGRAM WAS WRITTEN FOR FAIRFIELD INDUSTRIES INC
                *                                   10052 HARWIN DRIVE
                *                                   HOUSTON, TEXAS 77036
                *      BY JACK BURKETT AND NORMAN WALTERS
                *
                *
                *
                *      CALLING SEQUENCE        LI R11,FRMTR
                *                              BLWP R10
                *                              DATA OPLST,FMTLST
                *                              JMP ERR
                *
                *    WHERE    OPLST  - ADDRESS OF AN OPERAND LIST
                *             FMTLST - ADDRESS OF FORMAT LIST
                *                      ONE-TO-ONE TO OPLST
                *
                *
                *
                *                              IF FORMATTER IS EXITED BECAUSE
                *                              OF AN ERROR, THE ERROR AREA
                *                              WILL BE INDICATED BY THE VALUE
                *                              STORED IN R0. THE ADDRESS
                *                              IN R1, THE ADDRESS OF THE
                *                              OPERAND LIST IS STORED IN R2.
                *
                *                              R0 VALUE     ERROR AREA
                *                              1            INVALID OPCODE
                *                              2            REPEAT NESTING
                *                                           ERROR
                *                              3            INPUT CHARACTER
                *                                           SYNTAX ERROR
                *                              4            REPEAT VALUE MORE
                *                                           THAT MAX. ALLOWED
                *                              5            OUTPUT CHARACTER
                *                                           SUBROUTINE
                *                              6            INPUT CHARACTER
                *                                           SUBROUTINE
                *                              7            CURSOR POSITION
                *                                           OUTPUT ROUTINE
                *                              8            CURSOR POSITION
                *                                           INPUT ROUTINE
                *                              9            RESERVED (NOT USED)
                *                              10           ESCAPE SYMBOL
                *                                           DETECTED
                *
                *
                *
                *
                        IDT  'FRMTR'
                        DEF  FRMTR
                        DEF  TRANAD
                        REF  B$RAM
                        REF  CIO$$
                        REF  ENT$M
                        REF  PANIC
                        REF  RIO$
        002A  ARG1    EQU  42                POINT PAST OVERHEAD
        000C  BASEV   EQU  12                BASE OF VARIABLE AREA
        0300  BMASK   EQU  >300              RADIX MASK
        003F  COLM    EQU  >3F               COLUMN MASK
        000F  IWMASK  EQU  >F                WIDTH MASK
        000B  LINK    EQU  11                LINK REGISTER
```

```
0069        000E  MAX     EQU  14              2X MAX FORMAT NEST REPEAT LOOP
0070        0040  NCOLM   EQU  >40             NEGATIVE COLUMN MASK
0071        0010  NROWM   EQU  >10             NEGATIVE ROW MASK
0072        0028  RESULT  EQU  40              INPUT DEVICE STORAGE
0073        007F  RMASK   EQU  >7F             REPEAT MASK
0074        0780  ROWM    EQU  >780            ROW MASK
0075        000F  ROWM2   EQU  >F              NEGATIVE ROW UNNEEDED BITS
0076              *                            MASK
0077        003F  SMASK   EQU  >3F             SHIFT MASK
0078        0400  TMASK   EQU  >400            STRING I/O BIT MASK
0079        000A  TOP     EQU  10              TOP OF STACK
0081              *
0082              *
0083              *
0084              *       STACK DEFINITIONS
0085              *
0086              *
0087              *
0088 0000                   DORG 0
0089 0000         BAS     BSS  2               BASE STORAGE
0090 0002         BASE    BSS  2               BASE STORAGE
0091 0004         BASOU   BSS  20              ASCII FORM OF OUTPUT NUMBER
0092 0018         BIN     BSS  2               INPUT NUMBER
0093 001A         BTYPE   BSS  2               BACKSPACE TYPE
0094 001C         BUF     BSS  6               BUFFER
0095 0022         BUFA    BSS  2               BUFFER ADDRESS
0096 0024         CBP     BSS  2               CURRENT POSITION FOR STRING IN
0097 0026         DECCT   BSS  2               BACKSPACE COUNTER
0098 0028         DIGFND  BSS  2               DIGIT FOUND FLAG
0099 002A         ECR     BSS  2               MAX LINE ACTUAL
0100 002C         FIRST   BSS  2               LOCATION OF FIRST DIGIT
0101 002E         FLDAD   BSS  2               FIELD ADDRESS
0102 0030         FMSTK   BSS  16              NEXT FORMAT ADDRESS
0103 0040         LBUF    BSS  66              LINE BUFFER
0104 0082         LEN     BSS  2               STRING LENGTH
0105 0084         LOOPC   BSS  2               LOOP COUNT
0106 0086         MAXL    BSS  2               MAX LINE LENGTH
0107 0088         RDCURS  BSS  2               READ CURSOR INFORMATION
0108 008A         RDINP   BSS  2               READ INPUT INFORMATION
0109 008C         RESTK   BSS  16              LOOP COUNTER
0110 009C         RETURN  BSS  2               R11 TEMPORARY SAVE
0111 009E         RTYPE   BSS  2               READ TYPE
0112 00A0         SAVI    BSS  2               LINK SAVE
0113 00A2         SAVR0   BSS  2               REGISTER SAVE AREAS
0114 00A4         SAVR1   BSS  2               "
0115 00A6         SAVR2   BSS  2               "
0116 00A8         SAVR3   BSS  2               "
0117 00AA         SAVR4   BSS  2               "
0118 00AC         SIG     BSS  2               SIGN
0119 00AE         SIGNLG  BSS  2               SIGN LEGAL FLAG
0120 00B0         STAT    BSS  2               ERROR STATUS
0121 00B2         TEMP    BSS  2               R0 TEMPORARY SAVE
0122 00B4         TEST    BSS  2               WIDTH STORAGE
0123 00B6         UPER    BSS  2               ASCII UPPER CASE STORAGE
0124 00B8         WID     BSS  2               FIELD WIDTH
0125 00BA         WRCURS  BSS  2               WRITE CURSOR INFORMATION
0126 00BC         WROUT   BSS  2               WRITE OUT INFORMATION
0127        00BE  EFRAME  EQU  $
0129 0000                   PSEG
0130 0000 0100    C100    DATA >100            READ PERIPHERAL OPCODE
0131 0002 0200    C200    DATA >200            WRITE PERIPHERAL OPCODE
0132 0004 0500    C500    DATA >500            READ CURSOR POSITION OPCODE
0133 0006 0600    C600    DATA >600            WRITE CURSOR POSITION OPCODE
0134 0008 000A    CA      DATA >A              INITIAL VALUE FOR BASE
0135 000A 0001    C1      DATA >1              STACK AREA POINTER OFFSET
0136 000C 0004    C4      DATA >4              INITIAL VALUE FOR BUF
0137 000E 0001    D1      DATA >1              INITIAL VALUE FOR DECCT
0138              *
0139              *
0140              *
0141              *       THE FOLLOWING SEVEN MASKS ARE PLACED IN THE PSEG
0142              *       BECAUSE THEY ARE USED IN COC INSTRUCTIONS. DO NOT SET
0143              *       AS EQUATES UNLESS CODE IS MODIFIED TO REMOVE COC'S.
0144              *
0145              *
0146              *
0147 0010 0040    DMASK   DATA >40             RADIX USE BIT MASK
```

```
0148 0012 0020    INTINC  DATA >20             INTEGER OP ADDRESS INCREMENT
0149 0014 0040    NCOLM2  DATA >40             NEGATIVE COLUMN CHECK MASK
0150 0016 0800    NROWM2  DATA >800            NEGATIVE ROW CHECK MASK
0151 0018 0080    NMASK   DATA >80             SET OR EXAMINE RADIX BIT MASK
0152 001A 0800    PMASK   DATA >800            I/O DIRECTIONAL BIT MASK
0153 001C 0400    SMASK2   DATA >400           SIGNED NUMBER MASK
0154 001E 000A    TABL    DATA 10              BASE INDEX TABLE
0155 0020 0010            DATA 16              "
0156 0022 0002            DATA 2               "
0157 0024 0008            DATA 8               "
0158 0026 0800    ZMASK   DATA >800            LEADING ZERO SUPPRESSION MASK
0159 0028   0D    CHRCR   BYTE >D              RETURN
0160 0029   2E    CHRPER  BYTE '.'             PERIOD
0161 002A   0A    CHRLF   BYTE >A              LINE FEED
0162 002B   3A    CHRCOL  BYTE ':'             COLON
0163 002C   2C    CHRCOM  BYTE ','             COMMA
0164 002D   7F    CHRRUB  BYTE >7F             RUBOUT
0165 002E   0B    CHRUPA  BYTE >B              UP CURSOR (VT)
0166 002F   08    CHRBKA  BYTE >8              BACKSPACE
0167 0030   20    CHRBLN  BYTE ' '             SPACE
0168 0031   5E    CHRCAR  BYTE '^'             ESCAPE SYMBOL (UP ARROW)
0169 0032   2A    CHRSTA  BYTE '*'             STAR
0170 0033   2B    CHRPLS  BYTE '+'             PLUS SIGN
0171 0034   2D    CHRMIN  BYTE '-'             MINUS SIGN
0172 0035   2F    CHRSLA  BYTE '/'             SLASH
0173              EVEN
0175 0036 06A0    FRMTR   BL   @ENT$M          GET ARGUMENT ADDRESSES
     0038 0000
0176 003A 00E6            DATA 40+EFRAME       ADJUST FOR MEMORY LOCATION
0177 003C C309            MOV  R9,R12          CURRENT STACK FRAME IN R9
0178 003E 022C            AI   R12,40          GET PAST OVERHEAD
     0040 0028
0179 0042 CB20            MOV  @CA,@BASE(R12)  INITIALIZE BASE AT 10
     0044 0008'
     0046 0002
0180 0048 CB20            MOV  @C4,@BUF(R12)   INITIALIZE BUF
     004A 000C'
     004C 001C
0181 004E C0CC            MOV  R12,R3          DUPLICATE STACK FRAME
0182 0050 0223            AI   R3,BUF          ADD ADDRESS OF BUF
     0052 001C
0183 0054 CB03            MOV  R3,@BUFA(R12)   PUT ADDRESS OF BUF IN BUFA
     0056 0022
0184 0058 04EC            CLR  @BAS(R12)
     005A 0000
0185 005C 04EC            CLR  @BIN(R12)
     005E 0018
0186 0060 04EC            CLR  @FIRST(R12)
     0062 002C
0187 0064 04EC            CLR  @FLDAD(R12)
     0066 002E
0188 0068 04EC            CLR  @LEN(R12)
     006A 0082
0189 006C 04EC            CLR  @LOOPC(R12)
     006E 0084
0190 0070 04EC            CLR  @RETURN(R12)
     0072 009C
0191 0074 04EC            CLR  @STAT(R12)
     0076 00B0
0192 0078 C1FE            MOV  *R14+,R7        INITIALIZE OPERAND POINTER
0193 007A 06A0            BL   @TRANAD
     007C 0804'
0194 007E C247            MOV  R7,R9
0195 0080 C1FE            MOV  *R14+,R7        INITIALIZE FORMAT POINTER
0196 0082 06A0            BL   @TRANAD
     0084 0804'
0197 0086 C207            MOV  R7,R8
0198 0088 C1FE            MOV  *R14+,R7        INITIALIZE LUNS POINTER
0199 008A 06A0            BL   @TRANAD
     008C 0804'
0200 008E C087            MOV  R7,R2
0201 0090 0200            LI   R0,>FFFE        INITIALIZE REPEAT FORMAT PNTR
     0092 FFFE
0202 0094 C092            MOV  *R2,R2          GET LUNS
0203 0096 C0C2            MOV  R2,R3           MAKE COPY OF LUNS
0204 0098 0243            ANDI R3,>FF          MASK FOR OUTPUT DEVICE
     009A 00FF
```

```
0205 009C CB03          MOV  R3,@WROUT(R12)   PUT OUTPUT DEVICE IN WROUT
     009E 00BC
0206 00A0 CB03          MOV  R3,@WRCURS(R12)  PUT OUTPUT DEVICE IN WRCURS
     00A2 00BA
0207 00A4 AB20          A    @C200,@WROUT(R12) PUT OPCODE IN WROUT
     00A6 0002'
     00A8 00BC
0208 00AA AB20          A    @C600,@WRCURS(R12) PUT OPCODE IN WRCURS
     00AC 0006'
     00AE 00BA
0209 00B0 06C2          SWPB R2               PUT INPUT DEVICE IN LOWER BYTE
0210 00B2 0242          ANDI R2,>FF           MASK FOR INPUT DEVICE
     00B4 00FF
0211 00B6 CB02          MOV  R2,@RDINP(R12)   PUT INPUT DEVICE IN RDINP
     00B8 008A
0212 00BA CB02          MOV  R2,@RDCURS(R12)  PUT INPUT DEVICE IN RDCURS
     00BC 0088
0213 00BE AB20          A    @C100,@RDINP(R12) PUT OPCODE IN RDINP
     00C0 0000'
     00C2 008A
0214 00C4 AB20          A    @C500,@RDCURS(R12) PUT OPCODE IN RDCURS
     00C6 0004'
     00C8 0088
0216 00CA C1D8  GTFMT   MOV  *R8,R7           GET NEXT ENTRY IN FMTLST
0217 00CC 0247          ANDI R7,>F000         MASK FOR OP CODE
     00CE F000
0218 00D0 09A7          SRL  R7,10            SHIFT OP CODE FOR INDEXING
0219 00D2 0287          CI   R7,7*4           CHECK FOR VALID OP CODE
     00D4 001C
0220 00D6 1521          JGT  ERR1B
0221 00D8 0467  FRM3    B    @FRM3+4(R7)      BRANCH INDEXED ON OP CODE
     00DA 00DC'
0222 00DC 0460          B    @OP0             TAB OR ENDFRM
     00DE 00FC'
0223 00E0 0460          B    @OP1             PRINT INTEGER
     00E2 0134'
0224 00E4 0460          B    @OP2             SET CURSOR ABSOLUTE
     00E6 01DE'
0225 00E8 0460          B    @OP3             INPUT INTEGER
     00EA 01F6'
0226 00EC 0460          B    @OP4             SET CURSOR INTEGER
     00EE 0272'
0227 00F0 0460          B    @OP5             STRING I/O
     00F2 02BA'
0228 00F4 0460          B    @OP6             REPEAT/ENDFRM
     00F6 036A'
0229 00F8 0460          B    @OP7             CRLF
     00FA 03C4'
0231              *
0232              *
0233              *
0234              *     THIS SECTION PROCESSES TAB OR ENDFRM
0235              *
0236              *
0237              *
0238 00FC C198  OP0     MOV  *R8,R6           GET W FIELD OF OP 0
0239 00FE 0246          ANDI R6,>007F         IF IT IS 0 THEN
     0100 007F
0240 0102 1310          JEQ  ENDF             PROCESS AN  END FORMAT
0241 0104 C1D8          MOV  *R8,R7           GET CHAR AND FILL COLS
0242 0106 0247          ANDI R7,>0F80
     0108 0F80
0243 010A 0227          AI   R7,>1000         ADD 20 HEX AFTER WHEN LEFT
     010C 1000
0244              *                           SHIFT
0245 010E 0A17          SLA  R7,1             IT PLACES ASCII IN UPPER BYTE
0246 0110 0606  FILL    DEC  R6               FILL UP W COLUMNS
0247 0112 1106          JLT  FRM5             WITH CHAR IN R7
0248 0114 06A0          BL   @OCHAR
     0116 0620'
0249 0118 10FB          JMP  FILL
0250 011A 0460  ERR1B   B    @ERR1
     011C 040A'
0251 011E 05C9  FRM4    INCT R9               BUMP OPERAND LIST POINTER
0252 0120 05C8  FRM5    INCT R8               BUMP FORMAT LIST POINTER
0253 0122 10D3          JMP  GTFMT
0254 0124 0280  ENDF    CI   R0,>FFFE         TEST FOR REPEAT NESTING ERROR
     0126 FFFE
0255 0128 1603          JNE  ERR2B
0256 012A 05CE          INCT R14
0257 012C 0460          B    @EXIT
     012E 041A'
0258 0130 0460  ERR2B   B    @ERR2
     0132 0406'
0260              *
0261              *
0262              *
0263              *     THIS SECTION PROCESSES PRINT INTEGER
0264              *
0265              *
0266              *
0267 0134 C1D8  OP1     MOV  *R8,R7           GET CURRENT FORMAT
0268 0136 21E0          COC  @NMASK,R7        IF N-BIT IS
     0138 0018'
0269 013A 1607          JNE  FRM6             ONE RESET BASE ONLY
0270 013C 0247  FM5A1   ANDI R7,BMASK
     013E 0300
0271 0140 0977          SRL  R7,7             USE CODE TIMES 2 FOR INDEX
0272 0142 CB27          MOV  @TABL(R7),@BASE(R12) MOVE NEW BASE FROM TABLE
     0144 001E'
     0146 0002
0273 0148 10EB          JMP  FRM5
0274 014A 21E0  FRM6    COC  @DMASK,R7        IF D-BIT IS ONE RESET BASE
     014C 0010'
0275 014E 1606          JNE  FRM6AA           BEFORE PRINTING INTEGER
0276 0150 0247          ANDI R7,BMASK
     0152 0300
0277 0154 0977          SRL  R7,7             USE 2 TIMES CODE FOR INDEX
0278 0156 CB27          MOV  @TABL(R7),@BASE(R12) MOVE NEW BASE FROM TABLE
     0158 001E'
     015A 0002
0279      015C' FRM6AA  EQU  $
0280 015C C158          MOV  *R8,R5           GET CURRENT FORMAT
0281 015E 0245          ANDI R5,IWMASK        AND MASK FOR WIDTH
     0160 000F
0282 0162 C1D9          MOV  *R9,R7           GET BINARY ADDRESS
0283 0164 06A0          BL   @TRANAD
     0166 0804'
0284 0168 C187          MOV  R7,R6
0285 016A C1EC          MOV  @BASE(R12),R7    SET BASE AND LEAVE IN R7
     016C 0002
0286 016E 06A0          BL   @BINBAS          CONVERT
     0170 041C'
0287 0172 C1C7          MOV  R7,R7            CHECK STATUS
0288 0174 1307          JEQ  FRM8             TOO BIG FOR
0289 0176 0207          LI   R7,'#'*256       FIELD WIDTH OR OTHER ERRORS
     0178 2300
0290 017A 0605  FRM7    DEC  R5               FILL FIELD
0291 017C 11D0          JLT  FRM4             WITH #'S
0292 017E 06A0          BL   @OCHAR
     0180 0620'
0293 0182 10FB          JMP  FRM7
0294 0184 C118  FRM8    MOV  *R8,R4           IF OUTPUT IS UNSIGNED
0295 0186 2120          COC  @SMASK2,R4       THEN
     0188 001C'
0296 018A 1602          JNE  FRM8A
0297 018C 0585          INC  R5               INCREASE WIDTH FOR SIGN
0298 018E 1001          JMP  FRM9
0299 0190 0586  FRM8A   INC  R6               SKIP THE SIGN
0300 0192 04C3  FRM9    CLR  R3
0301 0194 04C2          CLR  R2               RESET FIRST NON-ZERO CHAR FLAG
0302 0196 D1D6  FRM10   MOVB *R6,R7
0303 0198 0287          CI   R7,'-'*256
     019A 2D00
0304 019C 1313          JEQ  FRM11
0305 019E 0287          CI   R7,'+'*256
     01A0 2B00
0306 01A2 1310          JEQ  FRM11
0307 01A4 2120          COC  @ZMASK,R4        IF THE ZERO SUPPRESSION
     01A6 0026'
0308 01A8 130D          JEQ  FRM11            IS SELECTED THEN
0309 01AA C082          MOV  R2,R2            DON'T OUTPUT BLANK
0310 01AC 160B          JNE  FRM11
0311 01AE 0287          CI   R7,'0'*256       OUTPUT A BLANK
```

```
      01B0 3000
0312  01B2 1607        JNE   FRM10A          INSTEAD
0313  01B4 C2C3        MOV   R3,LINK         DO NOT SUPPRESS LAST DIGIT
0314  01B6 058B        INC   LINK
0315  01B8 814B        C     LINK,R5
0316  01BA 1303        JEQ   FRM10A
0317  01BC 0207        LI    R7,'  '*256
      01BE 2000
0318  01C0 1001        JMP   FRM11
0319  01C2 0702  FRM10A SETO  R2             SET NON-ZERO CHAR FLAG
0320  01C4 06A0  FRM11  BL    @OCHAR          CONTINUE UNTIL
      01C6 0620'
0321  01C8 0583        INC   R3
0322  01CA 8143        C     R3,R5           WIDTH IN R5
0323  01CC 1302        JEQ   FRM11A          IS OUTPUT
0324  01CE 0586        INC   R6
0325  01D0 10E2        JMP   FRM10
0326  01D2 C1D8  FRM11A MOV   *R8,R7         GET CURRENT FORMAT
0327  01D4 21E0        COC   @INTINC,R7      IS FORMAT INCREMENT BIT SET?
      01D6 0012'
0328  01D8 16A2        JNE   FRM4            NO, CONTINUE TO NEXT FORMAT
0329  01DA 05D9        INCT  *R9             INCREMENT OPERAND BY 2
0330  01DC 10A0        JMP   FRM4            NOW GO TO NEXT FORMAT
0332               *
0333               *
0334               *
0335               *       THIS SECTION PROCESSES SET CURSOR ABSOLUTE
0336               *
0337               *
0338               *
0339  01DE C198  OP2    MOV   *R8,R6          GET CURRENT FORMAT
0340  01E0 0246        ANDI  R6,ROWM         MASK FOR ROW IN R6
      01E2 0780
0341  01E4 0BF6        SRC   R6,16-1         PUT ROW IN UPPER BYTE
0342  01E6 C1D8        MOV   *R8,R7          MASK FOR COL IN R7
0343  01E8 0247        ANDI  R7,COLM         COLUMN IN LOWER BYTE
      01EA 003F
0344  01EC A1C6        A     R6,R7           ROW AND COLUMN IN R7
0345  01EE 06A0        BL    @CURABS         CALL ABSOLUTE CURS SUBROUTINE
      01F0 0654'
0346  01F2 0460        B     @FRM5           GO GET NEXT ROUTINE
      01F4 0120'
0348               *
0349               *
0350               *
0351               *       THIS SECTION PROCESSES INPUT INTEGER
0352               *
0353               *
0354               *
0355  01F6 C158  OP3    MOV   *R8,R5          GET CURRENT FORMAT
0356  01F8 C1D8        MOV   *R8,R7
0357  01FA 2160        COC   @NMASK,R5       IF N-BIT IS SET
      01FC 0018'
0358  01FE 139E        JEQ   FM5A1           GO RESET THE BASE
0359  0200 0245        ANDI  R5,IWMASK       GET THE WIDTH
      0202 000F
0360  0204 1602        JNE   FRM13
0361  0206 0205        LI    R5,16           0 IMPLIES 16
      0208 0010
0362  020A C198  FRM13  MOV   *R8,R6          IF S-BIT IS ONE
0363  020C 21A0        COC   @SMASK2,R6      INCREASE WIDTH IN R5
      020E 001C'
0364  0210 1601        JNE   FRM12
0365  0212 0585        INC   R5
0366       0214' FRM12  EQU   $
0367  0214 21A0        COC   @PMASK,R6       IS PADDING BIT ON?
      0216 001A'
0368  0218 160E        JNE   FRM12A          NO- DO NOT PAD
0369  021A C105        MOV   R5,R4           YES, FILL FIELD WITH STARS
0370  021C D1E0        MOVB  @CHRSTA,R7
      021E 0032'
0371  0220 06A0  FRM12B BL    @OCHAR
      0222 0620'
0372  0224 0604        DEC   R4
0373  0226 16FC        JNE   FRM12B
0374  0228 C105        MOV   R5,R4           NOW BACK UP CURSOR TO START OF
0375  022A D1E0        MOVB  @CHRBKA,R7      FIELD
      022C 002F'
```

```
0376  022E 06A0  FRM12C BL    @OCHAR
      0230 0620'
0377  0232 0604        DEC   R4
0378  0234 16FC        JNE   FRM12C
0379       0236' FRM12A EQU   $
0380  0236 CB05        MOV   R5,@MAXL(BASEV)   SET MAX STRING LENGTH
      0238 0086
0381  023A 06A0        BL    @INNUM            READ STRING
      023C 06B8'
0382  023E 21A0        COC   @DMASK,R6         IF D-BIT IS SET
      0240 0010'
0383  0242 1606        JNE   FRM14             THEN MASK FOR
0384  0244 0246        ANDI  R6,BMASK          BASE AND
      0246 0300
0385  0248 0976        SRL   R6,7
0386  024A CB26        MOV   @TABL(R6),@BASE(R12) RESET BASE
      024C 001E'
      024E 0002
0387       0250' FRM14  EQU   $
0388  0250 C1EC        MOV   @BASE(R12),R7     SET UP R5,R6,R7
      0252 0002
0389  0254 0206        LI    R6,LBUF           FOR CALL TO BASBIN
      0256 0040
0390  0258 A18C        A     BASEV,R6
0391  025A 06A0        BL    @BASBIN           TO CONVERT BINARY
      025C 04FE'
0392  025E C1C7        MOV   R7,R7             CHECK FOR ERRORS
0393  0260 1302        JEQ   FRM15A            NONE
0394  0262 0460        B     @ERR3             ERRORS, IGNORE FIELD
      0264 0402'
0395  0266 C1D9  FRM15A MOV   *R9,R7           GET ADDRESS OF OPLIST
0396  0268 06A0        BL    @TRANAD
      026A 0804'
0397  026C C5D6        MOV   *R6,*R7           SAVE BINARY AT OPLIST POSITION
0398  026E 0460        B     @FRM4             GET NEXT FORMAT
      0270 011E'
0400               *
0401               *
0402               *
0403               *
0404               *       THIS SECTION PROCESSES SET CURSOR RELATIVE
0405               *
0406               *
0407               *
0408  0272 C198  OP4    MOV   *R8,R6          GET CURRENT FORMAT
0409  0274 21A0        COC   @NROWM2,R6      IS ROW MOVEMENT UPWARD?
      0276 0016'
0410  0278 160A        JNE   OP4A            NO
0411  027A 0246        ANDI  R6,ROWM         GET ROW
      027C 0780
0412  027E 0B76        SRC   R6,7            PUT IN LOWER BYTE
0413  0280 0506        NEG   R6              GET TWO'S COMPLEMENT
0414  0282 0246        ANDI  R6,ROWM2        GET RID OF UNNEEDED BITS
      0284 000F
0415  0286 0226        AI    R6,NROWM        2'S COMPLEMENT W/SIGN IN R6
      0288 0010
0416  028A 0B96        SRC   R6,9            PUT ROW BACK TO ORIGINAL PLACE
0417  028C 1002        JMP   OP4B
0418  028E 0246  OP4A   ANDI  R6,ROWM         GET ROW IN R6
      0290 0780
0419  0292 C1D8  OP4B   MOV   *R8,R7          GET FORMAT IN R7
0420  0294 21E0        COC   @NCOLM2,R7      IS COLUMN MOVEMENT TO LEFT?
      0296 0014'
0421  0298 1608        JNE   OP4C            NO
0422  029A 0247        ANDI  R7,COLM         GET COLUMN
      029C 003F
0423  029E 0507        NEG   R7              GET TWO'S COMPLEMENT
0424  02A0 0247        ANDI  R7,COLM         GET RID OF UNNEEDED BITS
      02A2 003F
0425  02A4 0227        AI    R7,NCOLM        2'S COMPLE. W/SIGN OF COLUMN
      02A6 0040
0426  02A8 1002        JMP   OP4D
0427  02AA 0247  OP4C   ANDI  R7,COLM         MASK FOR COLUMN IN R7
      02AC 003F
0428  02AE 0BF6  OP4D   SRC   R6,16-1         PUT ROW IN UPPER BYTE
0429  02B0 A1C6        A     R6,R7           ROW AND COLUMN IN R7
0430  02B2 06A0        BL    @CURREL         CALL RELATIVE CURSOR SUB
```

```
      02B4 0686'
0431  02B6 0460          B     @FRM5               GO GET NEXT FORMAT
      02B8 0120'
0433                *
0434                *
0435                *
0436                *       THIS SECTION PROCESSES STRING INPUT/OUTPUT
0437                *       THIS IS SETUP CODE. REAL WORK DONE LATER.
0438                *
0439                *
0440                *
0441  02BA C118     OP5   MOV   *R8,R4              GET CURRENT FORMAT MASK
0442  02BC 0244           ANDI  R4,SMASK            FOR STRING LEN IN R4
      02BE 003F
0443  02C0 C1D8           MOV   *R8,R7
0444  02C2 0247           ANDI  R7,>380             MASK FOR LENGTH MULTIPLIER
      02C4 0380
0445  02C6 0977           SRL   R7,7                RIGHT JUSTIFY MULTIPLIER
0446  02C8 0587           INC   R7                  ADD 1 FOR BASE (1 * N = N)
0447  02CA 3907           MPY   R7,R4               MULTIPLY STRING LENGTH BY MULT
0448  02CC C1D8           MOV   *R8,R7              IF D BIT IS ZERO, JUMP TO
0449  02CE 21E0           COC   @DMASK,R7           PROCESS OUTPUT
      02D0 0010'
0450  02D2 1623           JNE   FRM19
0452                *
0453                *
0454                *
0455                *       THIS SECTION HANDLES STRING INPUT
0456                *
0457                *
0458                *
0459  02D4 C1D8           MOV   *R8,R7              GET STRING LENGTH MAX FROM
0460                *                               FORMAT
0461  02D6 0247           ANDI  R7,>3F
      02D8 003F
0462  02DA CB07           MOV   R7,@MAXL(BASEV)     SET UP INPUT ROUTINE
      02DC 0086
0463  02DE 06A0           BL    @INSTRG             GET INPUT STRING
      02E0 06C2'
0464  02E2 04C3           CLR   R3                  READY R3 FOR INPUT LOOP
0465  02E4 04C4           CLR   R4                  CR FLAG
0466  02E6 C1D9           MOV   *R9,R7              ADDRESS OF OPERAND TO R6
0467  02E8 06A0           BL    @TRANAD
      02EA 0804'
0468  02EC C187           MOV   R7,R6
0469  02EE C104     FM15A1 MOV  R4,R4               IF CR FLAG IS SET
0470  02F0 1303           JEQ   FRM17               REPLACE INPUT WITH BLANK
0471  02F2 0207     FRM16 LI    R7,'  '*256         PADDING
      02F4 2000
0472  02F6 1007           JMP   FRM18
0473       02F8'    FRM17 EQU   $
0474  02F8 C1C3           MOV   R3,R7               GET CHARACTER OUT OF BUFFER
0475  02FA A1CC           A     BASEV,R7
0476  02FC D1E7           MOVB  @LBUF(R7),R7
      02FE 0040
0477  0300 9807           CB    R7,@CHRCR           IS IT A CR
      0302 0028'
0478  0304 1308           JEQ   FRM18A              YES, VALID TERMINATION
0479  0306 C046     FRM18 MOV   R6,R1               STORE CHAR IN OPLIST
0480  0308 A043           A     R3,R1
0481  030A D447           MOVB  R7,*R1
0482  030C 0583           INC   R3                  IS STRING ALL IN
0483  030E 8143           C     R3,R5               AND PADDED
0484  0310 16EE           JNE   FM15A1              GET NEXT CHARACTER
0485  0312 0460           B     @FRM4               GO GET NEXT FORMAT AND OPLIST
      0314 011E'
0486  0316 0704     FRM18A SETO R4                  SET CR DETECTED
0487  0318 10EC           JMP   FRM16               AND SUBSTITUTE BLANK
0489                *
0490                *
0491                *
0492                *       THIS SECTION HANDLES STRING OUTPUT
0493                *
0494                *
0495                *
0496  031A C098     FRM19 MOV   *R8,R2
0497  031C 20A0           COC   @PMASK,R2           IF F-BIT IS ZERO
      031E 001A'
0498  0320 1303           JEQ   FRM20               SET DATA TO FORMAT AREA
0499  0322 C188           MOV   R8,R6
0500  0324 05C6           INCT  R6
0501  0326 1004           JMP   FRM21
0502  0328 C1D9     FRM20 MOV   *R9,R7              ELSE USE ADDRESS IN OPLIST
0503  032A 06A0           BL    @TRANAD
      032C 0804'
0504  032E C187           MOV   R7,R6
0505  0330 04C3     FRM21 CLR   R3                  CLEAR R3 FOR OUTPUT LOOP
0506  0332 04C7           CLR   R7
0507  0334 C118           MOV   *R8,R4
0508  0336 0244           ANDI  R4,TMASK            GET THE T-BIT NULL INDICATOR
      0338 0400
0509  033A C046     FRM22 MOV   R6,R1               GET NEXT CHAR
0510  033C A043           A     R3,R1
0511  033E D1D1           MOVB  *R1,R7
0512  0340 C104           MOV   R4,R4               IF OUTPUT TERMINATES ON
0513  0342 1602           JNE   FRM23               NULL THEN
0514  0344 D1C7           MOVB  R7,R7               CHECK FOR NULL
0515  0346 1305           JEQ   FRM24               EXIT IF NULL
0516  0348 06A0     FRM23 BL    @OCHAR              ELSE OUTPUT CHAR
      034A 0620'
0517  034C 0583           INC   R3
0518  034E 8143           C     R3,R5               CONTINUE UNTIL ALL OUT
0519  0350 16F4           JNE   FRM22
0520  0352 C098     FRM24 MOV   *R8,R2
0521  0354 20A0           COC   @PMASK,R2
      0356 001A'
0522  0358 1306           JEQ   FRM24A
0523  035A 0583           INC   R3
0524  035C 0913           SRL   R3,1                COMPUTE WORDS PROCESSED
0525  035E 0A13           SLA   R3,1
0526  0360 A203           A     R3,R8               SKIP PAST IMBEDDED STRING
0527  0362 0460           B     @FRM5               GET NEXT FORMAT
      0364 0120'
0528  0366 0460     FRM24A B    @FRM4               GET FMT & ADVANCE OPL POINTER
      0368 011E'
0530                *
0531                *
0532                *
0533                *       THIS SECTION HANDLES REPEAT/ENDREP
0534                *       THIS IS SETUP CODE ONLY. REAL WORK DONE LATER.
0535                *
0536                *
0537                *
0538  036A C1D8     OP6   MOV   *R8,R7
0539  036C 21E0           COC   @PMASK,R7
      036E 001A'
0540  0370 1611           JNE   FRM25
0542                *
0543                *
0544                *
0545                *       THIS SECTION HANDLES REPEAT
0546                *
0547                *
0548                *
0549  0372 05C0           INCT  R0                  ELSE STORE A NEW REPEAT VALUE
0550  0374 8380           C     R0,MAX
0551  0376 1543           JGT   ERR4
0552  0378 C080           MOV   R0,R2               R2 IS TEMP STORAGE REGISTER
0553  037A A08C           A     R12,R2              ADD POINTER TO CURRENT ADDRESS
0554  037C C188           MOV   R8,R6
0555  037E 05C6           INCT  R6
0556  0380 C886           MOV   R6,@FMSTK(R2)       SAVE NEXT FORMAT ADDRESS
      0382 0030
0557  0384 0247           ANDI  R7,RMASK            GET REPEAT VALUE
      0386 007F
0558                *
0559                *
0560                *
0561                *       THIS SECTION HANDLES REPEAT
0562                *
0563                *
0564                *
0565  03A2 05C0           INCT  R0                  ELSE STORE A NEW REPEAT VALUE
0566  03A4 0280           CI    R0,MAX
      03A6 000E
```

```
0567 03A8 1560          JGT  ERR4
0568 03AA C080          MOV  R0,R2              R2 IS TEMP STORAGE REGISTER
0569 03AC A08C          A    R12,R2             ADD POINTER TO CURRENT ADDRESS
0570 03AE C188          MOV  R8,R6
0571 03B0 05C6          INCT R6
0572 03B2 C886          MOV  R6,@FMSTK(R2)      SAVE NEXT FORMAT ADDRESS
     03B4 0030
0573 03B6 0247          ANDI R7,RMASK           GET REPEAT VALUE
     03B8 007F
0574 03BA C887          MOV  R7,@RESTK(R2)      SAVE REPEAT VALUE
     03BC 008C
0575 03BE CB02          MOV  R2,@LOOPC(R12)     SAVE POINTER TO ADDRESS
     03C0 0084
0576 03C2 0460          B    @FRM5
     03C4 0150'
0577 03AA 0642          DECT R2
0578 03AC CB02          MOV  R2,@LOOPC(R12)
     03AE 0084
0579 03B0 0460  FRM25B  B    @FRM5              NO NESTS LEFT
     03B2 0120'
0580 03B4 C887  FRM26   MOV  R7,@RESTK(R2)      GET LOOP COUNTER POINTER
     03B6 008C
0581 03B8 C222          MOV  @FMSTK(R2),R8      GET FORMAT POINTER
     03BA 0030
0582 03BC CB02          MOV  R2,@LOOPC(R12)
     03BE 0084
0583 03C0 0460          B    @GTFMT
     03C2 00CA'
0585                    *
0586                    *
0587                    *
0588                    *       THIS SECTION HANDLES CRLF
0589                    *
0590                    *
0591                    *
0592       03C4' OP7    EQU  $
0593 03C4 C158          MOV  *R8,R5             GET NUMBER OF CRLF'S
0594 03C6 0245          ANDI R5,>3F             MASK TO SIX BITS
     03C8 003F
0595 03CA 1601          JNE  OP71
0596 03CC 0585          INC  R5                 TREAT 0 AS 1
0597       03CE' OP71   EQU  $
0598 03CE D1E0          MOVB @CHRCR,R7          OUTPUT A CR, LF COMBO
     03D0 0028'
0599 03D2 06A0          BL   @OCHAR
     03D4 0620'
0600 03D6 D1E0          MOVB @CHRLF,R7
     03D8 002A'
0601 03DA 06A0          BL   @OCHAR
     03DC 0620'
0602 03DE 0605          DEC  R5                 DONE?
0603 03E0 16F6          JNE  OP71               NO, DO ANOTHER
0604 03E2 0460          B    @FRM5              GO GET NEXT FORMAT
     03E4 0120'
0606                    *
0607                    *
0608                    *
0609                    *       THIS SECTION IS ERROR EXITS
0610                    *
0611                    *
0612                    *
0613 03E6 05AC  ERR10   INC  @STAT(R12)
     03E8 00B0
0614 03EA 05AC  ERR9    INC  @STAT(R12)
     03EC 00B0
0615 03EE 05AC  ERR8    INC  @STAT(R12)
     03F0 00B0
0616 03F2 05AC  ERR7    INC  @STAT(R12)
     03F4 00B0
0617 03F6 05AC  ERR6    INC  @STAT(R12)
     03F8 00B0
0618 03FA 05AC  ERR5    INC  @STAT(R12)
     03FC 00B0
0619 03FE 05AC  ERR4    INC  @STAT(R12)
     0400 00B0
0620 0402 05AC  ERR3    INC  @STAT(R12)
     0404 00B0
0621 0406 05AC  ERR2    INC  @STAT(R12)
     0408 00B0
0622 040A 05AC  ERR1    INC  @STAT(R12)
     040C 00B0
0623                    *
0624                    *
0625                    *
0626                    *       THE PROGRAM WILL OUTPUT THREE DATA ITEMS (ERROR
0627                    *       CODE, OPLIST ADDRESS, FMTLIST ADDRESS) TO THE CALLING
0628                    *       PROGRAM IN R0,R1, AND R2 TO ASSIST DEBUG.
0629                    *
0630                    *
0631                    *
0632 040E C76C          MOV  @STAT(R12),*R13 ERROR CODE
     0410 00B0
0633 0412 CB49          MOV  R9,@2(R13)         OPLIST ADDRESS
     0414 0002
0634 0416 CB48          MOV  R8,@4(R13)         FMTLST ADDRESS
     0418 0004
0635 041A 0380  EXIT    RTWP
0637                    *
0638                    *
0639                    *
0640                    *       THIS SUBROUTINE CONVERTS A NUMBER FROM
0641                    *       BINARY TO ASCII
0642                    *
0643                    *
0644                    *
0645 041C CB05  BINBAS  MOV  R5,@TEST(R12)      SAVE WIDTH FOR TEST LATER
     041E 00B4
0646 0420 C196          MOV  *R6,R6             IF BINARY NUMBER IS NEGATIVE
0647 0422 1104          JLT  BIB0
0648 0424 DB20          MOVB @CHRPLS,@BASOU(R12)  STORE PLUS
     0426 0033'
     0428 0004
0649 042A 1004          JMP  BIB1
0650 042C DB20  BIB0    MOVB @CHRMIN,@BASOU(R12)  STORE - SIGN
     042E 0034'
     0430 0004
0651 0432 0506          NEG  R6
0652 0434 CB06  BIB1    MOV  R6,@BIN(R12)       SAVE THE BINARY NUMBER
     0436 0018
0653 0438 CB07          MOV  R7,@BAS(R12)       SAVE THE BASE
     043A 0000
0654 043C CB05          MOV  R5,@WID(R12)       SAVE THE FIELD WIDTH
     043E 00B8
0655 0440 0287          CI   R7,10              IF BASE IS TEN
     0442 000A
0656 0444 1603          JNE  BIB3
0657 0446 0205  BIB2    LI   R5,5               SET MAX OF 5 DIGITS
     0448 0005
0658 044A 1011          JMP  BIB4B
0659 044C 0287  BIB3    CI   R7,16              IF BASE IS 16
     044E 0010
0660 0450 1603          JNE  BIB4
0661 0452 0205          LI   R5,4               SET MAX OF 4 DIGITS
     0454 0004
0662 0456 100B          JMP  BIB4B
0663 0458 0287  BIB4    CI   R7,8               IF BASE IS 8
     045A 0008
0664 045C 13F4          JEQ  BIB2               SET MAX OF 5 DIGITS
0665 045E 0287          CI   R7,2               IF BASE IS 2
     0460 0002
0666 0462 1303          JEQ BIB4A
0667 0464 0420          BLWP @PANIC             INTERNAL ERROR
     0466 0000
0668 0468 0500          DATA >500               NO SUCH BASE
0669 046A 0205  BIB4A   LI   R5,16              SET MAX OF 16 DIGITS
     046C 0010
0670 046E C1EC  BIB4B   MOV  @TEST(R12),R7      DOWNLOAD INPUT MAX NO. DIGITS
     0470 00B4
0671 0472 1602          JNE  BIB4C              IF INPUT MAX IS ZERO,SET SAVED
0672 0474 CB05          MOV  R5,@TEST(R12)      MAX EQUAL TO TABLE MAX
     0476 00B4
0673 0478 8B05  BIB4C   C    R5,@TEST(R12)      IS INPUT MAX > TABLE MAX?
     047A 00B4
0674 047C 1503          JGT  BIB5               NO,CONTINUE
```

```
0675 047E 1302          JEQ  BIB5
0676 0480 0707 BIB4D    SETO R7              LOAD ERROR FLAG IN R7
0677 0482 103A          JMP  BIB9E           ERROR OUT FAST
0678 0484 CB05 BIB5     MOV  R5,@LEN(R12)    SAVE LOOP LENGTH
     0486 0082
0679 0488 CB05          MOV  R5,@FIRST(R12)  WATCH FOR ALL ZEROS
     048A 002C
0680 048C C1C6 BIB5A    MOV  R6,R7           SET R6,R7 TO HOLD DIVIDEND
0681 048E 04C6          CLR  R6
0682 0490 3DAC          DIV  @BAS(R12),R6    DIVIDE BY BASE
     0492 0000
0683 0494 0287          CI   R7,10           CONVERT
     0496 000A
0684 0498 1103          JLT  BIB6            TO DIGIT
0685 049A 0227          AI   R7,>37          ASCII CHAR
     049C 0037
0686 049E 1002          JMP  BIB6A
0687 04A0 0227 BIB6     AI   R7,'0'
     04A2 0030
0688 04A4 0287 BIB6A    CI   R7,'0'          SAVE LOCATION OF FIRST NON-ZERO
     04A6 0030
0689 04A8 1302          JEQ  BIB6B
0690 04AA CB05          MOV  R5,@FIRST(R12)
     04AC 002C
0691 04AE 06C7 BIB6B    SWPB R7
0692 04B0 C105          MOV  R5,R4
0693 04B2 A10C          A    R12,R4
0694 04B4 D907          MOVB R7,@BASOU(R4)   STORE THE REMAINDER
     04B6 0004
0695 04B8 0605          DEC  R5              CONTINUE UNTIL ALL DIGITS
0696 04BA 16E8          JNE  BIB5A           ARE STORED
0697 04BC C1AC          MOV  @LEN(R12),R6    CALCULATE NEW ADDRESS
     04BE 0082
0698 04C0 81AC          C    @WID(R12),R6
     04C2 00B8
0699 04C4 15DD          JGT  BIB4D
0700 04C6 1314          JEQ  BIB8
0701 04C8 61AC          S    @WID(R12),R6
     04CA 00B8
0702 04CC 882C          C    @BAS(R12),@TABL
     04CE 0000
     04D0 001E'
0703 04D2 1604          JNE  BIB7
0704 04D4 81AC          C    @FIRST(R12),R6
     04D6 002C
0705 04D8 11D3          JLT  BIB4D
0706 04DA 13D2          JEQ  BIB4D
0707 04DC C146 BIB7     MOV  R6,R5
0708 04DE A14C          A    R12,R5
0709 04E0 D96C          MOVB @BASOU(R12),@BASOU(R5)
     04E2 0004
     04E4 0004
0710 04E6 0205          LI   R5,BASOU        SET R6 TO ADDRESS OF
     04E8 0004
0711 04EA A14C          A    R12,R5          ADD OFFSET
0712 04EC A185          A    R5,R6           START OF DIGIT STRING
0713 04EE 1003          JMP  BIB9
0714 04F0 0206 BIB8     LI   R6,BASOU
     04F2 0004
0715 04F4 A18C          A    R12,R6          ADD OFFSET
0716 04F6 04C7 BIB9     CLR  R7              SET STATUS 0-GOOD
0717 04F8 C16C BIB9E    MOV  @TEST(R12),R5   RESTORE STORED MAX VALUE
     04FA 00B4
0718 04FC 045B          B    *LINK
0720                 *
0721                 *
0722                 *
0723                 *       THIS SUBROUTINE CONVERTS A NUMBER FROM
0724                 *       ASCII TO BINARY
0725                 *
0726                 *
0727                 *
0728 04FE CB07 BASBIN MOV    R7,@BAS(R12)     SAVE BASE FOR LATER
     0500 0000
0729 0502 04EC          CLR  @SIG(R12)        CLEAR SIGN
     0504 00AC
0730 0506 04EC          CLR  @DIGFND(R12)     CLEAR DIGIT FOUND FLAG
     0508 0028
0731 050A 04EC          CLR  @SIGNLG(R12)     CLEAR SIGN LEGAL FLAG
     050C 00AE
0732 050E CB06          MOV  R6,@FLDAD(R12)   SAVE FIELD ADDRESS
     0510 002E
0733 0512 0287          CI   R7,2             BASE 2?
     0514 0002
0734 0516 130C          JEQ  BAB5             YES, SET UPPER LIMIT AT '1'
0735 0518 0287          CI   R7,8             BASE 8?
     051A 0008
0736 051C 130C          JEQ  BAB6             YES, SET UPPER LIMIT AT '7'
0737 051E 0287          CI   R7,10            BASE 10?
     0520 000A
0738 0522 130C          JEQ  BAB7             YES, SET UPPER LIMIT AT '9'
0739 0524 0287          CI   R7,16            BASE 16?
     0526 0010
0740 0528 1309          JEQ  BAB7             YES, SAME AS BASE 10
0741 052A 0420          BLWP @PANIC           INTERNAL ERROR-NO SUCH BASE
     052C 0466'
0742 052E 0501          DATA >501
0743 0530 0207 BAB5     LI   R7,'1'
     0532 0031
0744 0534 1005          JMP  BAB8
0745 0536 0207 BAB6     LI   R7,'7'
     0538 0037
0746 053A 1002          JMP  BAB8
0747 053C 0207 BAB7     LI   R7,'9'
     053E 0039
0748 0540 CB07 BAB8     MOV  R7,@UPER(R12)
     0542 00B6
0749 0544 04EC          CLR  @BIN(R12)        READY FOR CONVERSION LOOP
     0546 0018
0750 0548 C1AC BAB10A   MOV  @FLDAD(R12),R6   GET ADDRESS NEXT CHAR
     054A 002E
0751 054C D196          MOVB *R6,R6           GET CHAR
0752 054E 06C6          SWPB R6
0753 0550 0246          ANDI R6,>FF
     0552 00FF
0754                 *
0755                 *
0756                 *
0757                 *       THIS SECTION OF CODE CHECKS TO SEE IF A VALID TERMIN-
0758                 *       ATION OF THE INTEGER HAS BEEN INPUT. A VALID TERMIN-
0759                 *       ATION IS ONE OF THE FOLLOWING: COMMA(,), CARRIAGE
0760                 *       RETURN(CR), SLASH(/), COLON(:), OR PERIOD(.). ANY
0761                 *       OTHER CHARACTER NOT DEFINED AS AN INTEGER WILL GIVE
0762                 *       AN ERROR INDICATION.
0763                 *
0764                 *
0765                 *
0766 0554 0286          CI   R6,',           IS IT A COMMA (,)?
     0556 002C
0767 0558 1348          JEQ  BAB16           YES
0768 055A 0286          CI   R6,>0D          IS IT A CARRIAGE RETURN (CR)?
     055C 000D
0769 055E 1345          JEQ  BAB16           YES
0770 0560 0286          CI   R6,' '          IS IT A SPACE ( )?
     0562 0020
0771 0564 1357          JEQ  BAB18           YES
0772 0566 0286          CI   R6,'/'          IS IT A SLASH (/)?
     0568 002F
0773 056A 133F          JEQ  BAB16           YES
0774 056C 0286          CI   R6,':'          IS IT A COLON (:)?
     056E 003A
0775 0570 133C          JEQ  BAB16
0776 0572 0286          CI   R6,'.'          IS IT A PERIOD (.)?
     0574 002E
0777 0576 1339          JEQ  BAB16           YES
0778 0578 0286          CI   R6,'+'          IS IT A PLUS (+)?
     057A 002B
0779 057C 1340          JEQ  BAB9A           YES
0780 057E 0286          CI   R6,'-'          IS IT A MINUS (-)?
     0580 002D
0781 0582 133D          JEQ  BAB9A           YES
0782 0584 0286          CI   R6,'0'          CHECK FOR NUMERIC RANGE
     0586 0030
0783 0588 1149          JLT  BAB9            MUST HAVE BEEN ILLEGAL
```

```
0784 058A 1319          JEQ   BAB12M          IGNORE LEADING ZEROS
0785 058C 8B06          C     R6,@UPER(R12)
     058E 00B6
0786 0590 1118          JLT   BAB12
0787 0592 1317          JEQ   BAB12
0788 0594 882C          C     @BAS(R12),@TABL+2 IF BASE 16
     0596 0000
     0598 0020'
0789 059A 1640          JNE   BAB9
0790 059C 0286          CI    R6,'A'            CHECK UPPER AND LOWER
     059E 0041
0791 05A0 113D          JLT   BAB9              CASE RANGE
0792 05A2 0286          CI    R6,'F'
     05A4 0046
0793 05A6 1110          JLT   BAB13
0794 05A8 130F          JEQ   BAB13
0795 05AA 0226          AI    R6,-32            CHANGE LOWER TO UPPER CASE
     05AC FFE0
0796 05AE 0286          CI    R6,'A'
     05B0 0041
0797 05B2 1134          JLT   BAB9
0798 05B4 0286          CI    R6,'F'
     05B6 0046
0799 05B8 1107          JLT   BAB13
0800 05BA 1306          JEQ   BAB13
0801 05BC 102F          JMP   BAB9
0802 05BE 04C6  BAB12M  CLR   R6
0803 05C0 1007          JMP   BAB15
0804 05C2 0226  BAB12   AI    R6,-'0'
     05C4 FFD0
0805 05C6 1002          JMP   BAB15M
0806 05C8 0226  BAB13   AI    R6,-'A'+10
     05CA FFC9
0807 05CC 072C  BAB15M  SETO  @DIGFND(R12)      FOUND NON ZERO DIGIT
     05CE 0028
0808 05D0 C1C6  BAB15   MOV   R6,R7             BIN=BIN*BASE+R6
0809 05D2 C16C          MOV   @BIN(R12),R5
     05D4 0018
0810 05D6 396C          MPY   @BAS(R12),R5
     05D8 0000
0811 05DA A187          A     R7,R6
0812 05DC CB06          MOV   R6,@BIN(R12)
     05DE 0018
0813 05E0 C145          MOV   R5,R5             CHECK IF TOO BIG FOR STORAGE
0814 05E2 161C          JNE   BAB9              IT WAS, ERROR OUT
0815 05E4 05AC  BAB15A  INC   @FLDAD(R12)
     05E6 002E
0816 05E8 10AF          JMP   BAB10A
0817 05EA C1EC  BAB16   MOV   @SIG(R12),R7      IF SIGN IS NEGATIVE
     05EC 00AC
0818 05EE 1302          JEQ   BAB17
0819 05F0 052C          NEG   @BIN(R12)         NEGATE RESULTS
     05F2 0018
0820 05F4 0206  BAB17   LI    R6,BIN            STORE ADDR. OF BIN IN R6
     05F6 0018
0821 05F8 A18C          A     R12,R6            ADD POINTER TO CURRENT ADDRESS
0822 05FA 04C7          CLR   R7
0823 05FC 045B          RT                      RETURN
0824              *
0825              *
0826              *
0827              *       THIS SECTION HANDLES + AND -
0828              *
0829              *
0830              *
0831 05FE C1EC  BAB9A   MOV   @SIGNLG(R12),R7   ARE SIGNS LEGAL?
     0600 00AE
0832 0602 160C          JNE   BAB9              NO, FOUND ONE EARLIER
0833 0604 0286          CI    R6,'+'            OK, WHAT WAS IT AGAIN?
     0606 002B
0834 0608 1302          JEQ   BAB9A1
0835 060A 072C          SETO  @SIG(R12)         WAS MINUS, SAVE INFO
     060C 00AC
0836 060E 072C  BAB9A1  SETO  @SIGNLG(R12)      MAKE FURTHER SIGNS ILLEGAL
     0610 00AE
0837 0612 10E8          JMP   BAB15A
0838              *
0839              *
```

```
0840              *
0841              *       THIS SECTION HANDLES BLANKS
0842              *
0843              *
0844              *
0845 0614 C1EC  BAB18   MOV   @DIGFND(R12),R7   ANY NON ZERO DIGITS YET?
     0616 0028
0846 0618 16D2          JNE   BAB12M            YES, BLANKS ARE NOW ZEROS
0847 061A 10E4          JMP   BAB15A            NO, IGNORE LEADING BLANKS
0848 061C 0707  BAB9    SETO  R7                SET R7 FOR ERROR INDICATION
0849 061E 045B          RT
0851              *
0852              *
0853              *
0854              *       THIS SUBROUTINE IS THE INTERFACE TO FIPMX CIO$$
0855              *       OUTPUT CHARACTER OPERATIONS
0856              *
0857              *
0858              *
0859      0620' OCHAR   EQU   $                 OUTPUT CHARACTER TO CRT
0860 0620 CB0B          MOV   LINK,@RETURN(R12) SAVE LINK
     0622 009C
0861 0624 02AB          STWP  LINK              PUT WSP IN LINK
0862 0626 022B          AI    LINK,14           CALCULATE ADDRESS OF R7
     0628 000E
0863 062A CAAC          MOV   @WROUT(R12),@40(TOP) SETUP FOR CHARACTER OUTPUT
     062C 00BC
     062E 0028
0864 0630 CA8B          MOV   LINK,@42(TOP)     R7 HAS CHARACTER TO BE OUTPUT
     0632 002A
0865 0634 CB00          MOV   R0,@TEMP(R12)     SAVE R0
     0636 00B2
0866 0638 020B          LI    LINK,CIO$$        SETUP CIO$$ ADDRESS FOR CALL
     063A 0000
0867 063C 040A  OCHAR1  BLWP  TOP               OUTPUT A CHARACTER
0868 063E 1008          JMP   ERR51             ERROR EXIT
0869 0640 10FD          JMP   OCHAR1            LOOP UNTIL CHARACTER OUTPUT
0870 0642 C2EC          MOV   @RETURN(R12),LINK RESTORE VALUE OF LINK
     0644 009C
0871 0646 C02C          MOV   @TEMP(R12),R0     RESTORE VALUE OF R0
     0648 00B2
0872 064A CB00          MOV   R0,@TEMP(R12)     SAVE R0
     064C 00B2
0873 064E 045B          RT
0874 0650 0460  ERR51   B     @ERR5
     0652 03FA'
0876              *
0877              *
0878              *
0879              *       THIS SUBROUTINE IS THE INTERFACE TO THE FIPMX
0880              *       CIO$$ CURSOR SET FUNCTION
0881              *
0882              *
0883              *
0884      0654' CURABS  EQU   $                 OUTPUT ABSOLUTE CURSOR POSIT
0885 0654 CB0B          MOV   LINK,@RETURN(R12) SAVE LINK
     0656 009C
0886 0658 CB07  CUROUT  MOV   R7,@BUF+2(R12)    PUT CURSOR POSITION INTO BUF
     065A 001E
0887 065C CAAC          MOV   @WRCURS(R12),@40(TOP) SET UP FOR CURSOR OUTPUT
     065E 00BA
     0660 0028
0888 0662 CAAC          MOV   @BUFA(R12),@42(TOP) BUF+2 HAS NEW CURSOR POSITIO
     0664 0022
     0666 002A
0889 0668 04EA          CLR   @44(TOP)          CLEAR DSR WORD
     066A 002C
0890 066C CB00          MOV   R0,@TEMP(R12)
     066E 00B2
0891 0670 020B          LI    LINK,RIO$         SETUP RIO$ ADDRESS FOR CALL
     0672 0000
0892 0674 040A          BLWP  TOP               OUTPUT CURSOR POSITION
0893 0676 1005          JMP   ERR71             ERROR EXIT
0894 0678 C2EC          MOV   @RETURN(R12),LINK RESTORE VALUE OF LINK
     067A 009C
0895 067C C02C          MOV   @TEMP(R12),R0     RESTORE VALUE OF R0
     067E 00B2
```

```
0896 0680 045B          RT
0897 0682 0460  ERR71   B     @ERR7
     0684 03F2'
0899                    *
0900                    *
0901                    *
0902                    *       THIS SUBROUTINE IS THE INTERFACE TO THE FIPMX
0903                    *       CIO$$ READ CURSOR FUNCTION
0904                    *
0905                    *
0906                    *       RELATIVE CURSOR ADDRESSING CANNOT BE DONE WITH THE
0907                    *       ADM3 (CURSOR READ CANNOT BE DONE). AS SET UP
0908                    *       SUBROUTINE IS FOR THE ADM-1 ONLY.
0909                    *
0910                    *
0911                    *
0912      0686' CURREL  EQU   $                 OUTPUT RELATIVE CURSOR  POSITIO
0913 0686 CB0B          MOV   LINK,@RETURN(R12) SAVE LINK
     0688 009C
0914 068A CAAC          MOV   @RDCURS(R12),@40(TOP) SETUP TO READ PRESENT
     068C 0088
     068E 0028
0915                    *                               CURSOR POSITION
0916 0690 CAAC          MOV   @BUFA(R12),@42(R10) BUF WILL HAVE PRESENT CURSOR
     0692 0022
     0694 002A
0917                    *                               POSITION
0918 0696 04EA          CLR   @44(TOP)          CLEAR DSR WORD
     0698 002C
0919 069A CB00          MOV   R0,@TEMP(R12)
     069C 00B2
0920 069E 020B          LI    LINK,RIO$         SETUP RIO$ ADDRESS FOR CALL
     06A0 0672'
0921 06A2 040A          BLWP  TOP               INPUT PRESENT CURSOR POSITION
0922 06A4 1007          JMP   ERR81             ERROR EXIT
0923 06A6 C02C          MOV   @TEMP(R12),R0
     06A8 00B2
0924 06AA C06C          MOV   @BUF+2(R12),R1    GET PRESENT CURSOR POSITION
     06AC 001E
0925 06AE A1C1          A     R1,R7             ADD PRESENT TO NEW CURSOR POS
0926 06B0 0460          B     @CUROUT           OUTPUT NEW RELATIVE POSITION
     06B2 0658'
0927 06B4 0460  ERR81   B     @ERR8
     06B6 03EE'
0929                    *
0930                    *
0931                    *
0932                    *       THIS SUBROUTINE HANDLES CONSOLE EDITING FUNCTIONS.
0933                    *       FOR INPUT STRING AND INPUT INTEGER. MAXL IS ASSUMED
0934                    *       TO BE SET UP BY THE CALLER TO THE MAXIMUM NUMBER OF
0935                    *       CHARACTERS TO ALLOW ON INPUT. THIS SUBROUTINE HAS TWO
0936                    *       (2) ENTRY POINTS, INNUM FOR INPUT INTEGER AND INSTRG
0937                    *       FOR INPUT STRING.
0938                    *
0939                    *
0940                    *
0941      06B8' INNUM   EQU   $
0942 06B8 CB0B          MOV   LINK,@SAVI(BASEV) SAVE LINK
     06BA 00A0
0943 06BC 072C          SETO  @RTYPE(BASEV)     RTYPE=.TRUE.
     06BE 009E
0944 06C0 1004          JMP   L$0
0945      06C2' INSTRG  EQU   $
0946 06C2 CB0B          MOV   LINK,@SAVI(BASEV) SAVE LINK
     06C4 00A0
0947 06C6 04EC          CLR   @RTYPE(BASEV)     RTYPE=.FALSE.
     06C8 009E
0948                    *
0949                    *
0950                    *
0951                    *       NOW DO MAJOR SETUP
0952                    *
0953                    *
0954                    *
0955      06CA' L$0     EQU   $
0956 06CA CB00          MOV   R0,@SAVR0(BASEV)  SAVE R0-R4
     06CC 00A2
0957 06CE CB01          MOV   R1,@SAVR1(BASEV)
     06D0 00A4
0958 06D2 CB02          MOV   R2,@SAVR2(BASEV)
     06D4 00A6
0959 06D6 CB03          MOV   R3,@SAVR3(BASEV)
     06D8 00A8
0960 06DA CB04          MOV   R4,@SAVR4(BASEV)
     06DC 00AA
0961      06DE' L$2B    EQU   $
0962 06DE 04EC          CLR   @CBP(BASEV)       CBP=0
     06E0 0024
0963 06E2 04C3          CLR   R3                CURCT=0
0964 06E4 04C2          CLR   R2                BPTR=0
0966                    *
0967                    *
0968                    *
0969                    *       READ A CHARACTER AND DETERMINE ITS FUNCTIONAL CLASS
0970                    *
0971                    *
0972                    *
0973      06E6' L$2A    EQU   $
0974 06E6 CAAC          MOV   @RDINP(BASEV),@RESULT(TOP)
     06E8 008A
     06EA 0028
0975 06EC 02A1          STWP  R1
0976 06EE CA81          MOV   R1,@ARG1(TOP)
     06F0 002A
0977 06F2 020B          LI    LINK,CIO$$
     06F4 063A'
0978 06F6 040A          BLWP  TOP
0979 06F8 104E          JMP   L$7A              I/O ERROR
0980 06FA 10F5          JMP   L$2A              BUFFER EMPTY
0981 06FC 9800          CB    R0,@CHRCR         CHAR=CR?
     06FE 0028'
0982 0700 1329          JEQ   L$4B
0983 0702 9800          CB    R0,@CHRCOM        =COMMA?
     0704 002C'
0984 0706 1323          JEQ   L$4A
0985 0708 9800          CB    R0,@CHRPER        =PERIOD?
     070A 0029'
0986 070C 1320          JEQ   L$4A
0987 070E 9800          CB    R0,@CHRRUB        =RUBBOUT?
     0710 002D'
0988 0712 134D          JEQ   L$5B
0989 0714 9800          CB    R0,@CHRSLA        = SLASH?
     0716 0035'
0990 0718 131A          JEQ   L$4A
0991 071A 9800          CB    R0,@CHRCOL        = COLON?
     071C 002B'
0992 071E 1317          JEQ   L$4A
0993 0720 9800          CB    R0,@CHRUPA        =UP CURSOR?
     0722 002E'
0994 0724 132F          JEQ   L$6A
0995 0726 9800          CB    R0,@CHRBKA        =BACK ARROW?
     0728 002F'
0996 072A 1337          JEQ   L$5A
0997 072C 9800          CB    R0,@CHRBLN        =PRINTABLE?
     072E 0030'
0998 0730 1401          JHE   L$3A
0999 0732 10D9          JMP   L$2A              MUST HAVE BEEN NON PRINTING?
1001                    *
1002                    *
1003                    *
1004                    *       THIS SECTION PROCESSES PRINTABLE CHARACTERS THAT
1005                    *       ARE NOT TERMINATORS
1006                    *
1007                    *
1008                    *
1009      0734' L$3A    EQU   $
1010 0734 8B03          C     R3,@MAXL(BASEV)
     0736 0086
1011 0738 13D6          JEQ   L$2A
1012 073A C1C0          MOV   R0,R7
1013 073C 06A0          BL    @OCHAR            ECHO CHARACTER
     073E 0620'
1014 0740 C042          MOV   R2,R1
1015 0742 A04C          A     BASEV,R1
```

```
1016 0744 D840          MOVB R0,@LBUF(R1)       LBUF(BPTR)=CHAR
     0746 0040
1017 0748 0582          INC  R2                 BPTR=BPTR+1
1018 074A 0583          INC  R3                 CURCT=CURCT+1
1019 074C 10CC          JMP  L$2A
1021                    *
1022                    *
1023                    *
1024                    *       THIS SECTION HANDLES COMMAS AND PERIODS
1025                    *
1026                    *
1027                    *
1028      074E' L$4A    EQU  $
1029 074E C06C          MOV  @RTYPE(BASEV),R1   TYPE OF INPUT?
     0750 009E
1030 0752 13F0          JEQ  L$3A               FALSE- STRING
1031      0754' L$4B    EQU  $
1032 0754 C1C0          MOV  R0,R7
1033 0756 06A0          BL   @OCHAR             ECHO CHARACTER
     0758 0620'
1034 075A C042          MOV  R2,R1
1035 075C A04C          A    BASEV,R1
1036 075E D840          MOVB R0,@LBUF(R1)       LBUF(BPTR)=CHAR
     0760 0040
1037 0762 0582          INC  R2                 BPTR=BPTR+1
1038 0764 0583          INC  R3                 CURCT=CURCT+1
1039 0766 CB03          MOV  R3,@ECR(BASEV)     ECR=CURCT
     0768 002A
1041                    *
1042                    *
1043                    *
1044                    *
1045                    *       THIS SECTION RESTORES REGISTERS AND EXITS
1046                    *
1047                    *
1048                    *
1049 076A C02C          MOV  @SAVR0(BASEV),R0   RESTORE REGISTERS
     076C 00A2
1050 076E C06C          MOV  @SAVR1(BASEV),R1
     0770 00A4
1051 0772 C0AC          MOV  @SAVR2(BASEV),R2
     0774 00A6
1052 0776 C0EC          MOV  @SAVR3(BASEV),R3
     0778 00A8
1053 077A C12C          MOV  @SAVR4(BASEV),R4
     077C 00AA
1054 077E C2EC          MOV  @SAVI(BASEV),LINK
     0780 00A0
1055 0782 045B          B    *LINK
1057                    *
1058                    *
1059                    *
1060                    *       THIS SECTION PROCESSES UP ARROW (ESCAPE EXIT)
1061                    *
1062                    *
1063                    *
1064      0784' L$6A    EQU  $
1065 0784 D1E0          MOVB @CHRCAR,R7         OUTPUT A '^'
     0786 0031'
1066 0788 06A0          BL   @OCHAR
     078A 0620'
1067 078C 04C3          CLR  R3                 CURCT=0
1068 078E 04EC          CLR  @ECR(BASEV)        ECR=0
     0790 002A
1069 0792 0460          B    @ERR10             DO ESCAPE EXIT
     0794 03E6'
1070      0796' L$7A    EQU  $
1071 0796 0460          B    @ERR6
     0798 03F6'
1073                    *
1074                    *
1075                    *
1076                    *       THIS SECTION HANDLES BACKSPACE AND RUBOUT
1077                    *
1078                    *
1079                    *
1080      079A' L$5A    EQU  $                  THIS IS ENTRY FOR BACKSPACE
```

```
1081 079A C043          MOV  R3,R1              CURCT=0?
1082 079C 13A4          JEQ  L$2A               YES
1083 079E 0602          DEC  R2                 NO, BPTR=BPTR-1
1084 07A0 0603          DEC  R3                 CURCT=CURCT-1
1085 07A2 CB20          MOV  @D1,@DECCT(BASEV)  DECCT=1
     07A4 000E
     07A6 0026
1086 07A8 072C          SETO @BTYPE(BASEV)      BTYPE=.TRUE.
     07AA 001A
1087 07AC 1006          JMP  L$3
1088      07AE' L$5B    EQU  $                  THIS IS ENTRY FOR RUBOUT
1089 07AE C043          MOV  R3,R1              CURCT=0?
1090 07B0 139A          JEQ  L$2A               YES
1091 07B2 CB03          MOV  R3,@DECCT(BASEV)   NO, DECCT=CURCT
     07B4 0026
1092 07B6 04EC          CLR  @BTYPE(BASEV)      BTYPE=.FALSE.
     07B8 001A
1093      07BA' L$3     EQU  $
1094                    *
1095                    *
1096                    *
1097                    *       BACK UP DECCT CHARACTERS
1098                    *
1099                    *
1100                    *
1101 07BA 04C4          CLR  R4                 I=0
1102      07BC' L$4     EQU  $
1103 07BC 0584          INC  R4                 I=I+1
1104 07BE 8B04          C    R4,@DECCT(BASEV)
     07C0 0026
1105 07C2 1B05          JH   L$6
1106 07C4 D1E0          MOVB @CHRBKA,R7
     07C6 002F'
1107 07C8 06A0          BL   @OCHAR
     07CA 0620'
1108 07CC 10F7          JMP  L$4
1109      07CE' L$6     EQU  $
1110                    *
1111                    *
1112                    *
1113                    *NOW OUTPUT DECCT BLANKS
1114                    *
1115                    *
1116                    *
1117 07CE 04C4          CLR  R4                 I=0
1118      07D0' L$7     EQU  $
1119 07D0 0584          INC  R4                 I=I+1
1120 07D2 8B04          C    R4,@DECCT(BASEV)
     07D4 0026
1121 07D6 1B05          JH   L$9
1122 07D8 D1E0          MOVB @CHRBLN,R7
     07DA 0030'
1123 07DC 06A0          BL   @OCHAR
     07DE 0620'
1124 07E0 10F7          JMP  L$7
1125      07E2' L$9     EQU  $
1126                    *
1127                    *
1128                    *
1129                    *       NOW MOVE BACK DECCT CHARACTERS (AGAIN!)
1130                    *
1131                    *
1132                    *
1133 07E2 04C4          CLR  R4                 I=0
1134      07E4' L$10    EQU  $
1135 07E4 0584          INC  R4                 I=I+1
1136 07E6 8B04          C    R4,@DECCT(BASEV)
     07E8 0026
1137 07EA 1B05          JH   L$12
1138 07EC D1E0          MOVB @CHRBKA,R7
     07EE 002F'
1139 07F0 06A0          BL   @OCHAR
     07F2 0620'
1140 07F4 10F7          JMP  L$10
1141      07F6' L$12    EQU  $
1142 07F6 C02C          MOV  @BTYPE(BASEV),R0
     07F8 001A
```

```
1143 07FA 1302          JEQ   BL$2B             FALSE
1144 07FC 0460          B     @L$2A             TRUE
     07FE 06E6'
1145      0800' BL$2B   EQU   $
1146 0800 0460          B     @L$2B
     0802 06DE'
1148                *
1149                *
1150                *
1151                *       THIS SUBROUTINE HANDLES ADDRESS TRANSLATION
1152                *       FOR ADDRESSES
1153                *
1154                *
1155                *
1156      0804' TRANAD  EQU   $
1157 0804 0287          CI    R7,B$RAM          STACK OR ABSOLUTE?
     0806 0000
1158 0808 1B01          JH    TRAD1             ABSOLUTE
1159 080A A1CD          A     R13,R7            STACK RELATIVE
1160      080C' TRAD1   EQU   $
1161 080C 21E0          COC   @C1,R7            DIRECT OR INDIRECT?
     080E 000A'
1162 0810 1601          JNE   TRAD2             DIRECT
1163 0812 C1D7          MOV   *R7,R7            INDIRECT
1164      0814' TRAD2   EQU   $
1165 0814 045B          RT
1166 0816               PEND
1167                    END
```

```
   ARG1     002A    E B$RAM    0806    ' BAB10A   0548    ' BAB12    05C2
 ' BAB12M   05BE    ' BAB13    05C8    ' BAB15    05D0    ' BAB15A   05E4
 ' BAB15M   05CC    ' BAB16    05EA    ' BAB17    05F4    ' BAB18    0614
 ' BAB5     0530    ' BAB6     0536    ' BAB7     053C    ' BAB8     0540
 ' BAB9     061C    ' BAB9A    05FE    ' BAB9A1   060E      BAS      0000
 ' BASBIN   04FE      BASE     0002      BASEV    000C      BASOU    0004
 ' BIB0     042C    ' BIB1     0434    ' BIB2     0446    ' BIB3     044C
 ' BIB4     0458    ' BIB4A    046A    ' BIB4B    046E    ' BIB4C    0478
 ' BIB4D    0480    ' BIB5     0484    ' BIB5A    048C    ' BIB6     04A0
 ' BIB6A    04A4    ' BIB6B    04AE    ' BIB7     04DC    ' BIB8     04F0
 ' BIB9     04F6    ' BIB9E    04F8      BIN      0018    ' BINBAS   041C
 ' BL$2B    0800      BMASK    0300      BTYPE    001A      BUF      001C
   BUFA     0022    ' C1       000A    ' C100     0000    ' C200     0002
 ' C4       000C    ' C500     0004    ' C600     0006    ' CA       0008
   CBP      0024    ' CHRBKA   002F    ' CHRBLN   0030    ' CHRCAR   0031
 ' CHRCOL   002B    ' CHRCOM   002C    ' CHRCR    0028    ' CHRLF    002A
 ' CHRMIN   0034    ' CHRPER   0029    ' CHRPLS   0033    ' CHRRUB   002D
 ' CHRSLA   0035    ' CHRSTA   0032    ' CHRUPA   002E    E CIO$$    06F4
   COLM     003F    ' CURABS   0654    ' CUROUT   0658    ' CURREL   0686
 ' D1       000E      DECCT    0026      DIGFND   0028    ' DMASK    0010
   ECR      002A      EFRAME   00BE    ' ENDF     0124    E ENT$M    0038
 ' ERR1     040A    ' ERR10    03E6    ' ERR1B    011A    ' ERR2     0406
 ' ERR2B    0130    ' ERR3     0402    ' ERR4     03FE    ' ERR5     03FA
 ' ERR51    0650    ' ERR6     03F6    ' ERR7     03F2    ' ERR71    0682
 ' ERR8     03EE    ' ERR81    06B4    ' ERR9     03EA    ' EXIT     041A
 ' FILL     0110      FIRST    002C      FLDAD    002E    ' FM15A1   02EE
 ' FM5A1    013C      FMSTK    0030    ' FRM10    0196    ' FRM10A   01C2
 ' FRM11    01C4    ' FRM11A   01D2    ' FRM12    0214    ' FRM12A   0236
 ' FRM12B   0220    ' FRM12C   022E    ' FRM13    020A    ' FRM14    0250
 ' FRM15A   0266    ' FRM16    02F2    ' FRM17    02F8    ' FRM18    0306
 ' FRM18A   0316    ' FRM19    031A    ' FRM20    0328    ' FRM21    0330
 ' FRM22    033A    ' FRM23    0348    ' FRM24    0352    ' FRM24A   0366
 ' FRM25    0394    ' FRM25B   03B0    ' FRM26    03B4    ' FRM3     00D8
 ' FRM4     011E    ' FRM5     0120    ' FRM6     014A    ' FRM6AA   015C
 ' FRM7     017A    ' FRM8     0184    ' FRM8A    0190    ' FRM9     0192
 D FRMTR    0036    ' GTFMT    00CA    ' INNUM    06B8    ' INSTRG   06C2
 ' INTINC   0012      IWMASK   000F    ' L$0      06CA    ' L$10     07E4
 ' L$12     07F6    ' L$2A     06E6    ' L$2B     06DE    ' L$3      07BA
 ' L$3A     0734    ' L$4      07BC    ' L$4A     074E    ' L$4B     0754
 ' L$5A     079A    ' L$5B     07AE    ' L$6      07CE    ' L$6A     0784
 ' L$7      07D0    ' L$7A     0796    ' L$9      07E2      LBUF     0040
```

```
   LEN      0082      LINK     000B      LOOPC    0084      MAX      000E
   MAXL     0086      NCOLM    0040    ' NCOLM2   0014    ' NMASK    0018
   NROWM    0010    ' NROWM2   0016    ' OCHAR    0620    ' OCHAR1   063C
 ' OP0      00FC    ' OP1      0134    ' OP2      01DE    ' OP3      01F6
 ' OP4      0272    ' OP4A     028E    ' OP4B     0292    ' OP4C     02AA
 ' OP4D     02AE    ' OP5      02BA    ' OP6      036A    ' OP7      03C4
 ' OP71     03CE    E PANIC    052C    ' PMASK    001A      R0       0000
   R1       0001      R10      000A      R11      000B      R12      000C
   R13      000D      R14      000E      R15      000F      R2       0002
   R3       0003      R4       0004      R5       0005      R6       0006
   R7       0007      R8       0008      R9       0009      RDCURS   0088
   RDINP    008A      RESTK    008C      RESULT   0028      RETURN   009C
 E RIO$     06A0      RMASK    007F      ROWM     0780      ROWM2    000F
   RTYPE    009E      SAVI     00A0      SAVR0    00A2      SAVR1    00A4
   SAVR2    00A6      SAVR3    00A8      SAVR4    00AA      SIG      00AC
   SIGNLG   00AE      SMASK    003F    ' SMASK2   001C      STAT     00B0
 ' TABL     001E      TEMP     00B2      TEST     00B4      TMASK    0400
   TOP      000A    ' TRAD1    080C    ' TRAD2    0814    D TRANAD   0804
   UPER     00B6      WID      00B8      WRCURS   00BA      WROUT    00BC
 ' ZMASK    0026
```

0000 ERRORS

TXXREF

```
ARG1     0063    0976
B$RAM            0058  1157
BAB10A   0750    0816
BAB12    0804    0786  0787
BAB12M   0802    0784  0846
BAB13    0806    0793  0794  0799  0800
BAB15    0808    0803
BAB15A   0815    0837  0847
BAB15M   0807    0805
BAB16    0817    0767  0769  0773  0775  0777
BAB17    0820    0818
BAB18    0845    0771
BAB5     0743    0734
BAB6     0745    0736
BAB7     0747    0738  0740
BAB8     0748    0744  0746
BAB9     0848    0783  0789  0791  0797  0801  0814  0832
BAB9A    0831    0779  0781
BAB9A1   0836    0834
BAS      0089    0184  0653  0682  0702  0728  0788  0810
BASBIN   0728    0391
BASE     0090    0179  0272  0278  0285  0386  0388
BASEV    0064    0380  0390  0462  0475  0942  0943  0946  0947  0956
                 0957  0958  0959  0960  0962  0974  1010  1015  1029
                 1035  1039  1049  1050  1051  1052  1053  1054  1068
                 1085  1086  1091  1092  1104  1120  1136  1142
BASOU    0091    0648  0650  0694  0709  0709  0710  0714
BIB0     0650    0647
BIB1     0652    0649
BIB2     0657    0664
BIB3     0659    0656
BIB4     0663    0660
BIB4A    0669    0666
BIB4B    0670    0658  0662
BIB4C    0673    0671
BIB4D    0676    0699  0705  0706
BIB5     0678    0674  0675
BIB5A    0680    0696
BIB6     0687    0684
BIB6A    0688    0686
BIB6B    0691    0689
BIB7     0707    0703
BIB8     0714    0700
BIB9     0716    0713
BIB9E    0717    0677
BIN      0092    0185  0652  0749  0809  0812  0819  0820
BINBAS   0645    0286
BL$2B    1145    1143
BMASK    0065    0270  0276  0384
BTYPE    0093    1086  1092  1142
BUF      0094    0180  0182  0886  0924
```

BUFA 0095 0183 0888 0916
C1 0135 1161
C100 0130 0213
C200 0131 0207
C4 0136 0180
C500 0132 0214
C600 0133 0208
CA 0134 0179
CBP 0096 0962
CHRBKA 0166 0375 0995 1106 1138
CHRBLN 0167 0997 1122
CHRCAR 0168 1065
CHRCOL 0162 0991
CHRCOM 0163 0983
CHRCR 0159 0477 0598 0981
CHRLF 0161 0600
CHRMIN 0171 0650
CHRPER 0160 0985
CHRPLS 0170 0648
CHRRUB 0164 0987
CHRSLA 0172 0989
CHRSTA 0169 0370
CHRUPA 0165 0993
CIO$$ 0059 0866 0977
COLM 0066 0343 0422 0424 0427
CURABS 0884 0345
CUROUT 0886 0926
CURREL 0912 0430
D1 0137 1085
DECCT 0097 1085 1091 1104 1120 1136
DIGFND 0098 0730 0807 0845
DMASK 0147 0274 0382 0449
ECR 0099 1039 1068
EFRAME 0127 0176
ENDF 0254 0240
ENT$M 0060 0175
ERR1 0622 0250
ERR10 0613 1069
ERR1B 0250 0220
ERR2 0621 0258
ERR2B 0258 0255
ERR3 0620 0394
ERR4 0619 0551
ERR5 0618 0874
ERR51 0874 0868
ERR6 0617 1071
ERR7 0616 0897
ERR71 0897 0893
ERR8 0615 0927
ERR81 0927 0922
ERR9 0614
EXIT 0635 0257
FILL 0246 0249
FIRST 0100 0186 0679 0690 0704
FLDAD 0101 0187 0732 0750 0815
FM15A1 0469 0484
FM5A1 0270 0358
FMSTK 0102 0556 0581
FRM10 0302 0325
FRM10A 0319 0312 0316
FRM11 0320 0304 0306 0308 0310 0318
FRM11A 0326 0323
FRM12 0366 0364
FRM12A 0379 0368
FRM12B 0371 0373
FRM12C 0376 0378
FRM13 0362 0360
FRM14 0387 0383
FRM15A 0395 0393
FRM16 0471 0487
FRM17 0473 0470
FRM18 0479 0472
FRM18A 0486 0478
FRM19 0496 0450
FRM20 0502 0498
FRM21 0505 0501
FRM22 0509 0519

FRM23 0516 0513
FRM24 0520 0515
FRM24A 0528 0522
FRM25 0569 0540
FRM25B 0579 0575
FRM26 0580 0572
FRM3 0221 0221
FRM4 0251 0291 0328 0330 0398 0485 0528
FRM5 0252 0247 0273 0346 0431 0527 0560 0579 0604
FRM6 0274 0269
FRM6AA 0279 0275
FRM7 0290 0293
FRM8 0294 0288
FRM8A 0299 0296
FRM9 0300 0298
FRMTR 0175 0056
GTFMT 0216 0253 0583
INNUM 0941 0381
INSTRG 0945 0463
INTINC 0148 0327
IWMASK 0067 0281 0359
L$0 0955 0944
L$10 1134 1140
L$12 1141 1137
L$2A 0973 0980 0999 1011 1019 1082 1090 1144
L$2B 0961 1146
L$3 1093 1087
L$3A 1009 0998 1030
L$4 1102 1108
L$4A 1028 0984 0986 0990 0992
L$4B 1031 0982
L$5A 1080 0996
L$5B 1088 0988
L$6 1109 1105
L$6A 1064 0994
L$7 1118 1124
L$7A 1070 0979
L$9 1125 1121
LBUF 0103 0389 0476 1016 1036
LEN 0104 0188 0678 0697
LINK 0068 0313 0314 0315 0718 0860 0861 0862 0864 0866
0870 0885 0891 0894 0913 0920 0942 0946 0977
1054 1055
LOOPC 0105 0189 0559 0569 0578 0582
MAX 0069 0550
MAXL 0106 0380 0462 1010
NCOLM 0070 0425
NCOLM2 0149 0420
NMASK 0151 0268 0357
NROWM 0071 0415
NROWM2 0150 0409
OCHAR 0859 0248 0292 0320 0371 0376 0516 0599 0601 1013
1033 1066 1107 1123 1139
OCHAR1 0867 0869
OP0 0238 0222
OP1 0267 0223
OP2 0339 0224
OP3 0355 0225
OP4 0408 0226
OP4A 0418 0410
OP4B 0419 0417
OP4C 0427 0421
OP4D 0428 0426
OP5 0441 0227
OP6 0538 0228
OP7 0592 0229
OP71 0597 0595 0603
PANIC 0061 0667 0741
PMASK 0152 0367 0497 0521 0539
R0 0201 0254 0549 0550 0552 0574 0574 0576 0865
0871 0872 0890 0895 0919 0923 0956 0981 0983
0985 0987 0989 0991 0993 0995 0997 1012 1016
1032 1036 1049 1142
R1 0479 0480 0481 0509 0510 0511 0924 0925 0957
0975 0976 1014 1015 1016 1029 1034 1035 1036
1050 1081 1089
R10 0916

(continued from page 24)

```
R12     0177  0178  0179  0180  0181  0183  0184  0185  0186
        0187  0188  0189  0190  0191  0205  0206  0207  0208
        0211  0212  0213  0214  0272  0278  0285  0386  0388
        0553  0559  0569  0578  0582  0613  0614  0615  0616
        0617  0618  0619  0620  0621  0622  0632  0645  0648
        0650  0652  0653  0654  0670  0672  0673  0678  0679
        0682  0690  0693  0697  0698  0701  0702  0704  0708
        0709  0711  0715  0717  0728  0729  0730  0731  0732
        0748  0749  0750  0785  0788  0807  0809  0810  0812
        0815  0817  0819  0821  0831  0835  0836  0845  0860
        0863  0865  0870  0871  0872  0885  0886  0887  0888
        0890  0894  0895  0913  0914  0916  0919  0923  0924
R13     0632  0633  0634  1159
R14     0192  0195  0198  0256
R2      0200  0202  0202  0203  0203  0209  0210  0211  0212  0301
        0309  0309  0319  0496  0497  0520  0521  0552  0553
        0556  0558  0559  0569  0570  0573  0577  0578  0580
        0581  0582  0958  0964  1014  1017  1034  1037  1051
        1083
R3      0181  0182  0183  0203  0204  0205  0206  0300  0313
        0321  0322  0464  0474  0480  0482  0483  0505  0510
        0517  0518  0523  0524  0525  0526  0959  0963  1010
        1018  1038  1039  1052  1067  1081  1084  1089  1091
R4      0294  0295  0307  0369  0372  0374  0377  0441  0442
        0447  0465  0469  0469  0486  0507  0508  0512  0512
```

```
044.074                 VARI    DS      2
044.076                 VARJ    DS      2
044.100                 VARK    DS      2
044.102                 VARL    DS      2
044.104                 VARM    DS      2
044.106                 VARN    DS      2
044.110                 LBLTAB  EQU     *       MUST BE LAST LABEL IN LIST

044.110  000                    END     USERFWA


00353 Statements Assembled
11926 Bytes Free
 No Errors Detected



                        * TO CHANGE THE SYMBOL TABLE LISTING TO HEXADECIMAL FORMAT ONE SHOULD
                        * SUBSTITUTE THE FOLLOWING THREE LINES OF CODE FOR THE FIVE LINES IN THE
                        * 'PRINT' ROUTINE BEGINNING AND ENDING WITH 'CALL OCTOUT'.

000,000  315 007 000           CALL    HEXOUT  HIGH BYTE OUT
000,003  053                   DCX     H
000,004  315 007 000           CALL    HEXOUT  LOW BYTE OUT

                        * THEN REPLACE THE ROUTINE 'OCTOUT' WITH THE FOLLOWING ROUTINE:
                        *
                        * HEXOUT -- PRINT OUT BYTE AT M IN HEXADECIMAL FORMAT
                        *       ENTRY:  HL = ADDRESS OF BYTE TO BE OUTPUT
                        *       EXIT:   BYTE PRINTED ON CONSOLE
                        *       USES:   A,B,C,F
000,007  176            HEXOUT  MOV     A,M
000,010  107                    MOV     B,A
000,011  016 002                MVI     C,2
000,013  037                    RAR
000,014  037                    RAR
000,015  037                    RAR
000,016  037                    RAR
000,017  346 017        HEX1    ANI     1111B
000,021  306 060                ADI     '0'
000,023  376 072                CPI     '9'+1
000,025  332 032 000            JC      HEX2
000,030  306 007                ADI     'A'-'0'-10
000,032  377 002        HEX2    SCALL   .SCOUT  OUTPUT CHARACTER IN 'A'
000,034  015                    DCR     C
000,035  310                    RZ
000,036  170                    MOV     A,B
000,037  303 017 000            JMP     HEX1
```

```
                0692  0693  0694  0960  1053  1101  1103  1104  1117
                1119  1120  1133  1135  1136
R5              0280  0281  0290  0297  0315  0322  0355  0357  0359
                0361  0365  0369  0374  0380  0483  0518  0593  0594
                0596  0602  0645  0654  0657  0661  0669  0672  0673
                0678  0679  0690  0692  0695  0707  0708  0709  0710
                0711  0712  0717  0809  0810  0813  0813
R6              0238  0239  0246  0284  0299  0302  0324  0339  0340
                0341  0344  0362  0363  0367  0382  0384  0385  0386
                0389  0390  0397  0408  0409  0411  0412  0413  0414
                0415  0416  0418  0428  0429  0468  0479  0499  0500
                0504  0509  0554  0555  0556  0646  0646  0651  0652
                0680  0681  0682  0697  0698  0701  0704  0707  0712
                0714  0715  0732  0750  0751  0751  0752  0753  0766
                0768  0770  0772  0774  0776  0778  0780  0782  0785
                0790  0792  0795  0796  0798  0802  0804  0806  0808
                0811  0812  0820  0821  0833
R7              0192  0194  0195  0197  0198  0200  0216  0217  0218
                0219  0221  0241  0242  0243  0245  0267  0268  0270
                0271  0272  0274  0276  0277  0278  0282  0284  0285
                0287  0287  0289  0302  0303  0305  0311  0317  0326
                0327  0342  0343  0344  0356  0370  0375  0388  0392
                0392  0395  0397  0419  0420  0422  0423  0424  0425
                0427  0429  0443  0444  0445  0446  0447  0448  0449
                0459  0461  0462  0466  0468  0471  0474  0475  0476
                0476  0477  0481  0502  0504  0506  0511  0514  0514
                0538  0539  0557  0558  0570  0571  0580  0598  0600
                0653  0655  0659  0663  0665  0670  0676  0680  0683
                0685  0687  0688  0691  0694  0716  0728  0733  0735
                0737  0739  0743  0745  0747  0748  0808  0811  0817
                0822  0831  0845  0848  0886  0925  1012  1032  1065
                1106  1122  1138  1157  1159  1161  1163  1163
R8              0197  0216  0238  0241  0252  0267  0280  0294  0326
                0339  0342  0355  0356  0362  0408  0419  0441  0443
                0448  0459  0496  0499  0507  0520  0526  0538  0554
                0581  0593  0634
R9              0177  0194  0251  0282  0329  0395  0466  0502  0633
RDCURS  0107    0212  0214  0914
RDINP   0108    0211  0213  0974
RESTK   0109    0558  0570  0573  0580
RESULT  0072    0974
RETURN  0110    0190  0860  0870  0885  0894  0913
RIO$            0062  0891  0920
RMASK   0073    0557
ROWM    0074    0340  0411  0418
ROWM2   0075    0414
RTYPE   0111    0943  0947  1029
SAVI    0112    0942  0946  1054
SAVR0   0113    0956  1049
SAVR1   0114    0957  1050
SAVR2   0115    0958  1051
SAVR3   0116    0959  1052
SAVR4   0117    0960  1053
SIG     0118    0729  0817  0835
SIGNLG  0119    0731  0831  0836
SMASK   0077    0442
SMASK2  0153    0295  0363
STAT    0120    0191  0613  0614  0615  0616  0617  0618  0619  0620
                0621  0622  0632
TABL    0154    0272  0278  0386  0702  0788
TEMP    0121    0865  0871  0872  0890  0895  0919  0923
TEST    0122    0645  0670  0672  0673  0717
TMASK   0078    0508
TOP     0079    0863  0864  0867  0887  0888  0889  0892  0914  0918
                0921  0974  0976  0978
TRAD1   1160    1158
TRAD2   1164    1162
TRANAD  1156    0057  0193  0196  0199  0283  0396  0467  0503
UPER    0123    0748  0785
WID     0124    0654  0698  0701
WRCURS  0125    0206  0208  0887
WROUT   0126    0205  0207  0863
ZMASK   0158    0307

THERE ARE 0239 SYMBOLS
```

Can You Afford To Retire?

BY DAVID E. RAESIDE

Department of Radiological Sciences
University of Oklahoma Health Sciences Center
Oklahoma City, Oklahoma

This computer program will calculate how much initial capital you will need to financially survive a given number of retirement years at a given standard of living. Inflation and a return on invested capital are accounted for. The retirement income is assumed to be made up of two components: a fixed component (a fraction of the initial capital) which is not indexed to the inflation rate and an inflation indexed component (for example, Social Security).

The program is based upon an article by Edward W. Herold entitled "How Inflation Affects Your Retirement Dollar" which appeared in the September, 1979 issue of the IEEE magazine Spectrum. *Since the program is self-instructional, it is not necessary to know anything about it to use it.*


```
0010 PRINT
0020 PRINT
0030 PRINT "        WILL YOU BE ABLE TO AFFORD TO RETIRE?"
0040 PRINT
0050 PRINT
0060 PRINT
0070 PRINT
0080 PRINT
0090 PRINT
0100 PRINT
0110 PRINT
0120 PRINT
0130 PRINT
0140 PRINT
0150 PRINT
0160 PRINT
0170 PRINT
0180 PRINT
0190 PRINT
0200 PRINT
0210 PRINT
0220 PRINT
0230 PRINT
0240 PRINT
0250 PRINT
0260 PRINT
0270 PRINT
0280 PRINT
0290 PRINT
0300 PRINT
0310 PRINT "WOULD YOU LIKE TO SEE AN EXPLANATION OF THE PROGRAM (Y,N)?"
0320 INPUT X$
0330 IF X$="Y" THEN GOSUB 0980
0340 IF X$="Y" THEN GOTO 0420
0350 IF X$="N" THEN GOTO 0420
0360 GOTO 0310
0370 PRINT "ARE YOU READY FOR A RUN (Y,N)?"
0380 INPUT X$
0390 IF X$="Y" THEN GOTO 0420
0400 IF X$="N" THEN GOTO 1930
0410 GOTO 0370
0420 GOSUB 0840
0430 PRINT "INPUT AVERAGE ANNUAL INFLATION RATE"
0440 GOSUB 0880
0450 INPUT I
0460 GOSUB 0840
0470 PRINT "INPUT AFTER-TAX RATE OF RETURN ON INVESTMENT"
0480 GOSUB 0880
0490 INPUT J
0500 GOSUB 0840
0510 PRINT "INPUT ANNUAL FIXED (AFTER-TAX) INCOME"
0520 GOSUB 0910
0530 INPUT F
0540 GOSUB 0840
0550 PRINT "INPUT INITIAL EXCESS LIVING EXPENSE"
0560 GOSUB 0910
0570 INPUT E
0580 GOSUB 0840
0590 PRINT "INPUT THE NUMBER OF YEARS AFTER RETIREMENT"
0600 PRINT "       FOR WHICH FINANCIAL SURVIVAL IS PLANNED"
0610 GOSUB 0950
0620 INPUT N
0630 PRINT
0640 PRINT
0650 PRINT "THIS PROGRAM WILL NOW CALCULATE AND DISPLAY"
0660 PRINT "       THE AMOUNT OF INITIAL CAPITAL REQUIRED"
0670 PRINT
0680 PRINT
0690 LET B=(F*(1-EXP(-J*N)))/J
0700 IF I=J THEN GOTO 0730
0710 LET A=(E*(EXP((I-J)*N)-1))/(I-J)
0720 GOTO 0740
0730 LET A=N*E
0740 LET T=A-B
0750 PRINT "THE REQUIRED INITIAL CAPITAL IS:",T,"DOLLARS"
0760 PRINT
0770 PRINT
0780 PRINT "DO YOU WISH TO MAKE ANOTHER RUN (Y,N)?"
0790 INPUT X$
0800 IF X$="Y" THEN GOTO 0300
0810 IF X$="N" THEN GOTO 1930
0820 GOTO 0780
0830 REM SUBROUTINE NUMBER ONE
0840 PRINT
0850 PRINT "WHEN QUESTION MARK APPEARS"
0860 RETURN
0870 REM SUBROUTINE NUMBER TWO
0880 PRINT "            ENTER A DECIMAL NUMBER"
0890 RETURN
0900 REM SUBROUTINE NUMBER THREE
0910 PRINT "            ENTER A DECIMAL NUMBER WITHOUT"
0920 PRINT "                A DOLLAR SIGN, A COMMA, OR A SPACE"
0930 RETURN
0940 REM SUBROUTINE NUMBER FOUR
0950 PRINT "            ENTER AN INTEGER"
0960 RETURN
0970 REM SUBROUTINE NUMBER FIVE
0980 PRINT
0990 PRINT "ASSUME THAT A NEW RETIREE FACES AN ANNUAL INFLATION RATE"
1000 PRINT "OF 13% (0.13), AN AFTER-TAX RATE OF RETURN ON INVESTMENT"
1010 PRINT "OF 5% (0.05), SOCIAL SECURITY INCOME (INFLATION INDEXED) OF"
1020 PRINT "$8,000, AND A FIXED PENSION (AFTER TAXES) OF $16,000.  IN"
1030 PRINT "ADDITION, ASSUME THAT THE RETIREE WISHES TO PLAN FOR 20"
1040 PRINT "YEARS OF FINANCIAL SURVIVAL.  THE RETIREE'S INITIAL LIVING"
1050 PRINT "EXPENSE IS $24,000, WITH $16,000 OF THIS BEING IN EXCESS"
1060 PRINT "OF INDEXED INCOME.  THE PROGRAM SHOWS THAT THE INITIAL"
1070 PRINT "CAPITAL NEEDED TO MAINTAIN THE RETIREE'S STANDARD OF"
1080 PRINT "LIVING FOR 20 YEARS AFTER RETIREMENT IS ABOUT $588,000."
1090 PRINT
1100 PRINT
1110 PRINT "DO YOU WISH TO SEE THE EQUATIONS UPON WHICH THIS PROGRAM"
1120 PRINT "      IS BASED (Y,N)?"
1130 INPUT X$
1140 IF X$="Y" THEN GOSUB 1250
1150 IF X$="Y" THEN GOTO 1180
1160 IF X$="N" THEN GOTO 1180
1170 GOTO 1110
1180 PRINT "WOULD YOU LIKE TO SEE A SAMPLE DIALOGUE (Y,N)?"
1190 INPUT X$
1200 IF X$="Y" THEN GOSUB 1480
1210 IF X$="Y" THEN GOTO 0370
1220 IF X$="N" THEN GOTO 0370
1230 GOTO 1180
1240 REM SUB-SUBROUTINE NUMBER ONE
1250 PRINT
1260 PRINT "NOTATION:"
1270 PRINT "      N - YEAR NUMBER FOLLOWING RETIREMENT"
1280 PRINT "      S - INVESTMENT CAPITAL REMAINING AT YEAR N"
1290 PRINT "      T - INITIAL CAPITAL"
1300 PRINT "      F - ANNUAL FIXED (AFTER-TAX) INCOME"
1310 PRINT "      E - INITIAL EXCESS LIVING EXPENSE"
1320 PRINT "      I - ANNUAL AVERAGE INFLATION RATE"
1330 PRINT "      J - ANNUAL AFTER-TAX RETURN ON INVESTMENT"
1340 PRINT
1350 PRINT "THE UNDERLYING DIFFERENTIAL EQUATION STATES THAT"
1360 PRINT "THE RATE OF CHANGE OF S WITH RESPECT TO N IS EQUAL TO"
1370 PRINT "F + J*S - E*EXP(I*N)"
1380 PRINT
1390 PRINT "THE SOLUTION EQUATIONS ARE:"
1400 PRINT "S = (EXP(I*N)) * (T + (F*(1-EXP(-J*N))/J)"
1410 PRINT "      - (E*(EXP((I-J)*N)-1)/(I-J)))"
1420 PRINT "T = (E*((EXP(I-J)*N)-1)/(I-J)) - ((F*(1-EXP(-J*N)))/J),"
1430 PRINT "      FOR I NOT EQUAL TO J"
1440 PRINT "T = (N*E) - (F*((1-EXP(-J*N))/J)), FOR I EQUAL TO J"
1450 PRINT
1460 RETURN
1470 REM SUB-SUBROUTINE NUMBER TWO
1480 PRINT
1490 PRINT
1500 PRINT "THE FOLLOWING IS A SAMPLE DIALOGUE AND CALCULATION:"
1510 PRINT
1520 PRINT "WHEN QUESTION MARK APPEARS"
1530 PRINT "INPUT AVERAGE ANNUAL INFLATION RATE"
1540 PRINT "      ENTER A DECIMAL NUMBER"
1550 PRINT "? 0.13"
1560 PRINT
1570 PRINT "WHEN QUESTION MARK APPEARS"
1580 PRINT "INPUT AFTER-TAX RATE OF RETURN ON INVESTMENT"
1590 PRINT "      ENTER A DECIMAL NUMBER"
1600 PRINT "? 0.05"
1610 PRINT
1620 PRINT "WHEN QUESTION MARK APPEARS"
1630 PRINT "INPUT ANNUAL FIXED (AFTER-TAX) INCOME"
1640 PRINT "      ENTER A DECIMAL NUMBER WITHOUT"
1650 PRINT "          A DOLLAR SIGN, A COMMA, OR A SPACE"
1660 PRINT "? 16000.00"
1670 PRINT
1680 PRINT "WHEN QUESTION MARK APPEARS"
1690 PRINT "INPUT INITIAL EXCESS LIVING EXPENSE"
1700 PRINT "      ENTER A DECIMAL NUMBER WITHOUT"
1710 PRINT "          A DOLLAR SIGN, A COMMA, OR A SPACE"
1720 PRINT "? 16000.00"
1730 PRINT
1740 PRINT "WHEN QUESTION MARK APPEARS"
1750 PRINT "INPUT THE NUMBER OF YEARS AFTER RETIREMENT"
1760 PRINT "      FOR WHICH FINANCIAL SURVIVAL IS PLANNED"
1770 PRINT "          ENTER AN INTEGER"
1780 PRINT "? 20"
1790 PRINT
1800 PRINT
1810 PRINT "THIS PROGRAM WILL NOW CALCULATE AND DISPLAY"
1820 PRINT "      THE AMOUNT OF INITIAL CAPITAL REQUIRED"
1830 PRINT
1840 PRINT
1850 PRINT "THE REQUIRED INITIAL CAPITAL IS:    588328 DOLLARS"
1860 PRINT
1870 PRINT
1880 PRINT "THIS ENDS THE SAMPLE DIALOGUE AND CALCULATION"
1890 PRINT
1900 PRINT
1910 RETURN
1920 RETURN
1930 END
```

A HEX KEYBOARD WITH APPLICATIONS FOR THE 1802 ELF

BY MARTIN BRONSTEIN
9925 Zelzah #518
Northridge, CA 91325

Martin Bronstein is a 22-year-old senior at California State University at Northridge, majoring in chemistry. His interest in computers began when he took a programming class at school. He was inspired to write this article by Paul Wasserman's floating point mathpack which appeared in DDJ *#37.*

Introduction

Loading programs via a microcomputer's front panel toggle switches is a cumbersome process to say the least. The obvious alternative is to use a keyboard. The keyboard system presented here provides eight bits of hexadecimal data and uses CMOS integrated circuits for low power consumption. Software loader programs are not needed to use this keyboard. Best of all, the system can be built for less than twenty dollars.

Circuit Description

The heart of this keyboard system is IC 1, National Semiconductor's 74C922 sixteen key encoder. Input to the encoder comes from sixteen pushbutton switches arranged in column-row format.

Pressing any key causes pin 12 on IC 1 to go high. Providing that the KEYBOARD ENABLE line is high, the signal at pin 12 is inverted via IC 3a and validates the decoded output lines of IC 1 through pin 13. The signal at pin 12 is simultaneously sent through R1 and C3, which provide a 1 microsecond delay, to pin 3 of flip-flop IC 4. The Q output line is clocked low to high and this transition is used to clock the encoder outputs into IC 5, which is used as a quad latch. The 1 microsecond delay is necessary to insure the outputs of IC 1 are valid before IC 5 is clocked.

Pressing a second key initiates the same sequence of events as above, except this time the $\overline{Q}$ line of IC 4 is clocked low to high and causes the encoder outputs to be clocked into latch IC 6. Thus any two consecutive keystrokes causes eight bits of hexadecimal data to appear on the data lines D4-D7 (MSB) and D0-D3 (LSB).

After the second key is pressed, a low to high transition appears on the DATA RDY line which can be used to clock the data into memory. Only when KEYBOARD ENABLE is high will keystrokes intitiate the clock signals. The OUTPUT ENABLE line must be high to allow the data from latches IC 5 and IC 6 to appear on the bus. Operating the CLEAR pushbutton resets flip-flop IC 4 and clears latches IC 5 and IC 6. CLEAR may be pressed before or after any two keystrokes to clear the keyboard. If pressed after the first keystroke in order to correct an error, the DATA RDY line will clock low to high as IC 4 resets, which may cause problems. An example of how to handle this situation appears later on in the section on interfacing the keyboard to the 1802 Elf.

Construction

Any type of construction may be used, such as vector board or printed circuit. IC sockets are advisable since CMOS ICs are sensitive to static discharges and should not be placed in the circuit until all soldering is complete.

A suitable keypad may be made from sixteen individual momentary-contact pushbutton switches. Complete hex keypads are available from various mailorder dealers such as Jameco Electronics (1021 Howard Ave., San Carlos, CA 94020) for about eleven dollars.

A very inexpensive alternative is to use a surplus calculator keypad. The basic four function (+, −, X, ÷) type is best. These keypads are usually wired in colum-row format already and can be bought as surplus for about 1 dollar. One place that has them is Computer Components Inc. (5848 Sepulveda Blvd., Van Nuys, CA 91411). The keytops can be marked with the appropriate numbers and letters using white tape and a marking pen.

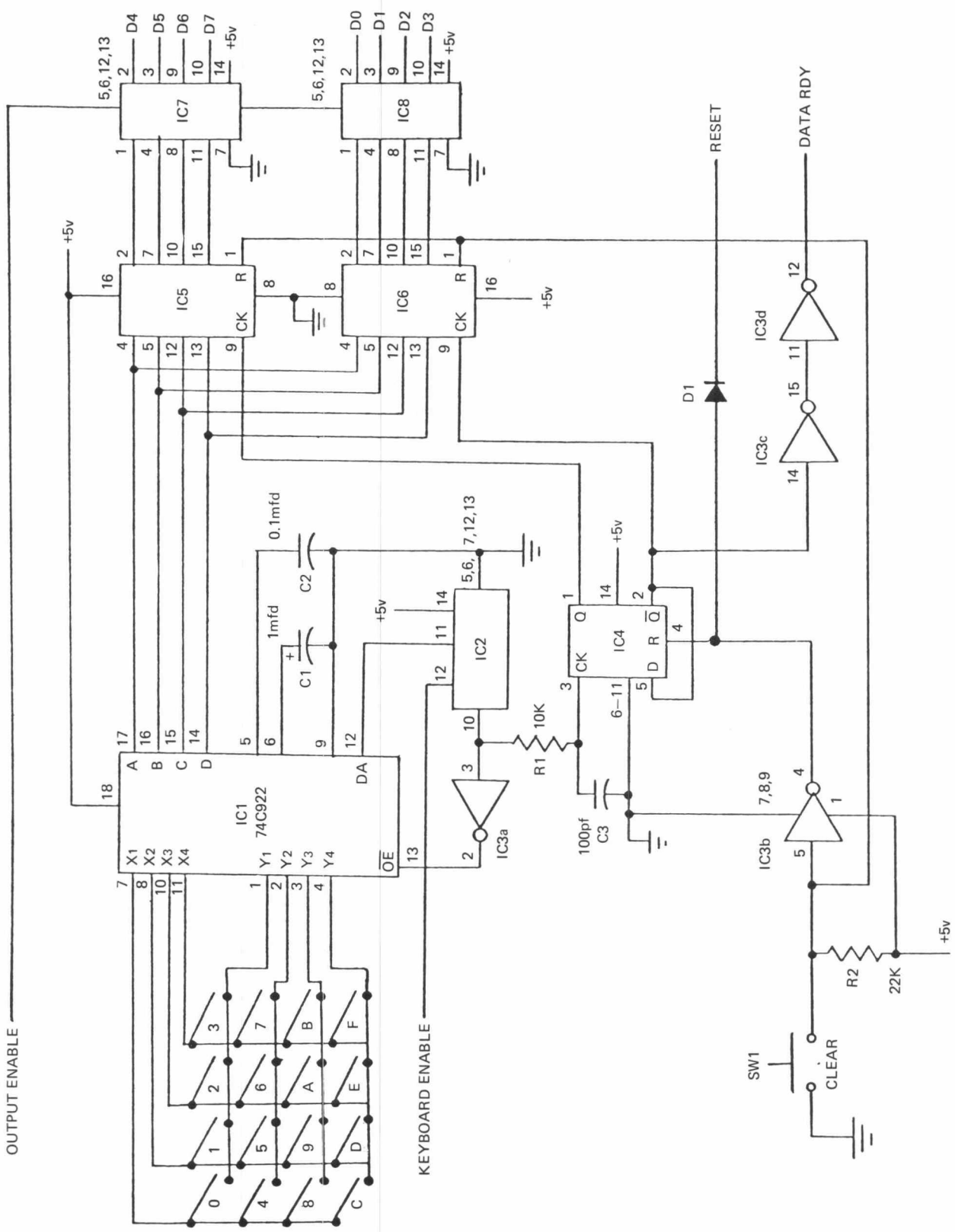


Figure 1. Hex keyboard

Interfacing the Elf 1802

The hex keyboard may be interfaced to the original Elf with two simple modifications.

First, locate the junction of pin 10, on IC 11, and the cathode of D5. Break the junction connection to pin 11 of IC 12 (see fig. 2) and insert the circuit shown in fig. 3. The new circuit allows the keyboard DATA RDY signal to clock the data on the bus into memory in the same manner as depressing the INPUT switch on the Elf.

The second modification involves locating the connection from pin 2 of IC 11 to pins 5, 6, 12, and 13 of IC 8 and 9 on the Elf. Break this connection and insert a DPDT toggle switch as shown in fig. 4. This switch is used to select operation of either the keyboard or the Elf's original toggle switches. This is necessary in order to prevent the toggle switch's data from reaching the bus while using the keyboard and vice versa. All other connections remain intact.

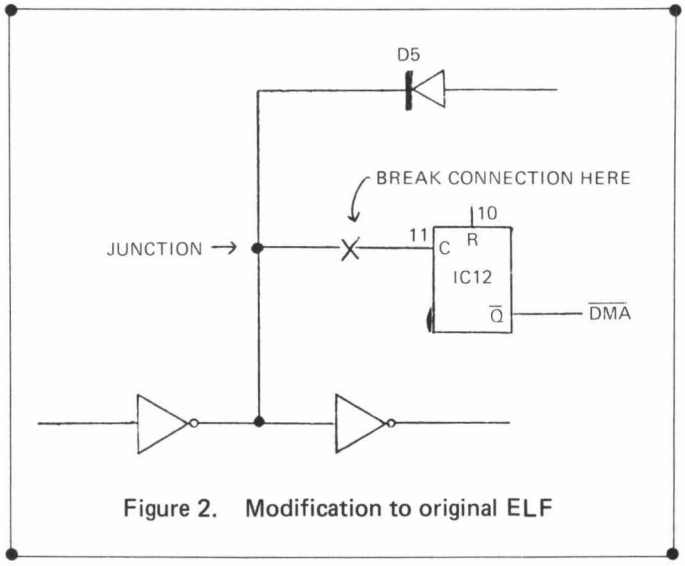


Figure 2. Modification to original ELF

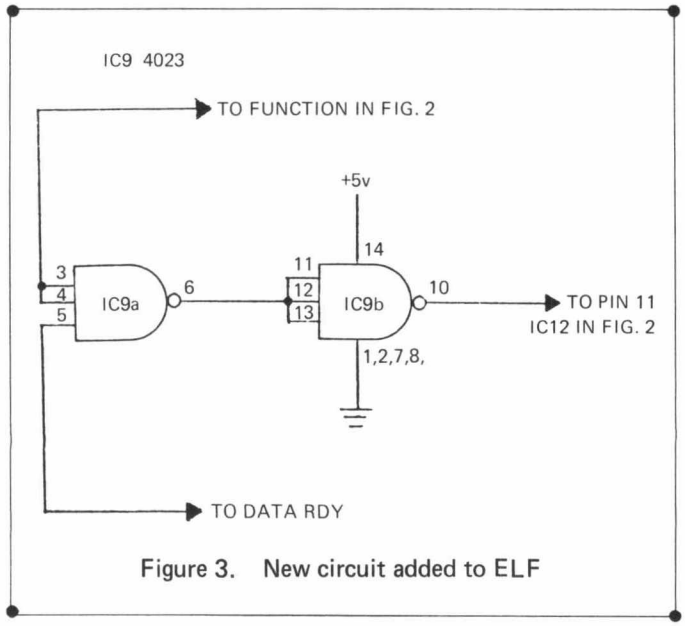


Figure 3. New circuit added to ELF

Finally, connect the KEYBOARD ENABLE line to the N2/LOAD line on the Elf (pin 1 of IC 10 will do) and connect RESET to pin 10 of IC 12 on the original Elf. The bus connections may be made directly to the microprocessor. All of the interface connections are summarized in figure 5.

The keyboard can now be used to program the Elf. Set the selector switch for keyboard operation, set the Elf LOAD switch to ON and press any two keys. The data will automatically appear on the display and be loaded into memory. All the Elf operating procedures remain the same.

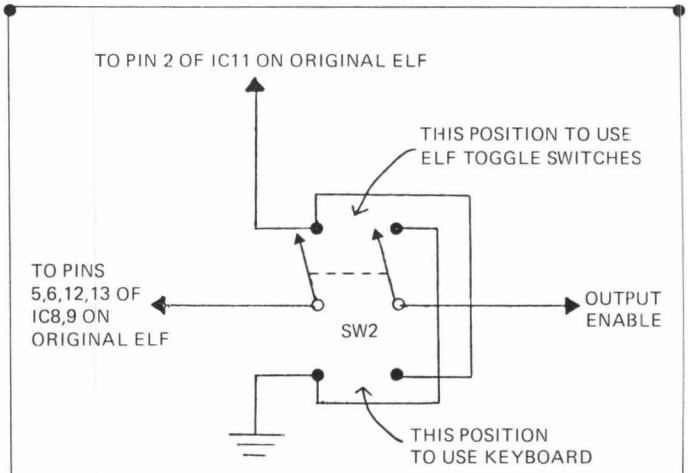


Figure 4. Connections for new selector switch

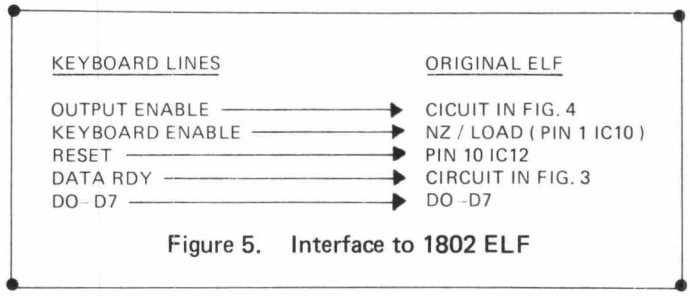


Figure 5. Interface to 1802 ELF

PARTS LIST

R1 – 10K ¼W resistor, 10%
R2 – 22K ¼K resistor, 10%
C1 – 1 mfd 10V. electrolytic capacitor
C2 – 0.1 mfd mylar or disk capacitor
C3 – 100 pf disk capacitor
IC1 – 74C922 16 key encoder
IC 2, 7, 8 – 4066 quad analog switch
IC – 34049 hex inverter
IC4 – 4013 dual D flip-flop
IC 5, 6 – 74C175 quad latch
IC 9 – 4023 triple 3-input NAND gate
SW 1 – momentary contact push button switch
D1 – 19914 switching diode
Misc. – hexadecimal keypad, vector board, etc.

Letters

KUDOS FOR JANUARY ISSUE

Dear Suzanne:

WOW! SUPER! EXTRAORDINARY! GREAT!

There are just not enough adjectives available to convey my impression of the January 1980 issue.

I'm sick of TRS-80, PET, APPLE, ATARI, etc. Why do they get all the play when the only true universal operating system around is CP/M? There must be 1000 mentions of TRS-80 to each mention of CP/M in the micro-computer press.

So you can well imagine how happy I was with your latest issue.

Only one request: PLEASE GIVE US LOTS MORE CP/M!

Thank you,
E. E. (Bud) Gramer

Gramer & Company
P. O. Box 35336
Phoenix, AZ 85069

Dear Ms. Rodriguez:

The all-CPM issue was excellent (of course!) and those of us who march to different beats will hope for an all-DOS issue or whatever. I have one small nit to pick. In your lead editorial you state: "Since CP/M is the most popular micro operating system in the world . . ." but is it really? Radio Shack pumps out a hundred thousand or so units each year and more and more of these are diskette based. So their DOS might well be the champ in terms of total use. If CP/M is the winner it must be by the combination of industrial and hobby market sales.

Our business (which is also our hobby, our crusade, our religion etc.) is plagued with these funny blind spots. *Dr. Dobb's* sometimes acts as if Radio Shack doesn't exist. Most writers on programming subjects ignore the very existence of RPG. PASCAL is praised as "structured" and COBOL condemned as "unstructured," and never mind the fact that top-down programming is easy in the latter and impossible in the former. By no means do I expect everyone or even a majority to share my particular likes and dislikes, but equal time would be fairer and would in fact make for some interesting debates.

So let's hear it for all the unloved ones in the land of the bit and the byte. Three cheers for Radio Shack, hurrah for COBOL, and *de facto* recognition at least for poor old RPG. Even a respectful bow to big bad IBM (or did you think I was using a Heathkit typewriter?)

Cheers,
John R. Culleton, Jr.

Culleton Group
2401 Haight Avenue
Sykesville, MD 21784

NEWBERRY ADDENDUMS

Dear Editor:

Some corrections to my article "An Overview of CP/M Compatible Software" in *DDJ* #41.

Page 9, column 1, middle of page, two occurrences of the name 'Sam Singer' are misspelled as 'Sam Sinser.' Page 11, column 1, top of page, line 4, change "'bmfMVI C, 1ZOtt: (or 'bmfMVI C,1')" to "'bmfMVI C, 1^ZOtt ' (or 'bmfMVI C,1^ZOtt .)"

Page 13, column 1, top of page, PASCAL/Z: I have since obtained the up-to-date documenation of PASCAL/Z (Version 2) which now has REALs (floating point) and pointers, and so is a full spec PASCAL *except* for the omission of variant records and the ability to pass procedures and functions as parameters.

Finally, I entirely omitted any mention of the TDL/Xitan line of software because of the collapse of the company. I have since learned that a new firm (Computer Design Labs, 342 Columbus Avenue, Trenton, N.J. 08629) has been formed to maintain and distribute the TDL/Xitan line. Available on either 5-¼" or 8" disks for standard CP/M, North Star CP/M and TRS-80 CP/M, the line requires a Z-80 processor. I particularly recommend the ZTEL, MACRO II, LINKER and ZAPPLE packages as outstanding values.

Cordially,
Steve Newberry

Newberry Microsystems
24225 Summerhill Ave.
Los Altos, CA 94022

POLYMORPHIC 8813 MATH FUNCTIONS

Dear Suzanne,

I use a Polymorphic 8813 as a home system (and sometimes think I am the only person in the world to own one). As a precision buff, I was delighted by their newest release of BASIC which among the many other features, has variable precision "settable" from 6 to 26 digits. By the way, this precision holds for all of the trigonometric and other math functions unlike the so-called double precision calculations in some other BASICs.

While playing with some mathematical diversions based on repeating fractions, I found a need to do some simple arithmetic with even greater precision that that already offered. After an hour or so of playing around with the disassembler, I found a patch to allow a limited amount of calculation to be done in precisions up to 62 digits.

The following code appears in the system overlay "Bdir.OV". A simple change of one byte on a COPY of the overlay using the "Szap" utility on the System Programmer's disc allows the increased precision.
;The following code resides at 2395H in the overlay area.

```
CD XX XX    CALL    PARAM    ;get number
7B          MOV     A, E     ;get returned byte
FE 06       CPI     06H      ;see if too small
OE 32       MVI     C, 32H   ;get small error code
DA XX XX    JC      ERROR    ;3FE5H is error location
FE 1B       CPI     1BH      ;test for too large
;The preceeding two bytes must be changed to "FE 3F" for
;increased precision operation.
OE 33       MVI     C, 33H   ;large error code
D2 XX XX    JNC     ERROR    ;do error
;etc, etc, . . . . . .
```

It should be noted that the math functions are not guaranteed to work well at the increased precision, in fact the trig functions will never be better than 26 digit because the constants built into the algorithms are not precise enough. The "Emedit" utility can be used to change the error message relating to precision limits as desired. The following operations have been used with success at 62 digits:

+, -, *, /, INPUT, PRINT, PRINT:n, SQRT, ABS, SGN, DO-loops and comparisons.

Hope this may be of help to some other hackers out there.

Sincerely,
John W. McGaw

23 Mack Drive
Las Vegas, NV 89110

INVISIBLE PROGRAMMER'S LAW

Dear Suzanne:

This letter is in reference to my article in *DDJ* #40, "An Interactive Heath H-8 Disassembler."

I would like to correct two errors in the article, one in the text and the other in the program listing.

In the 9th line from the end of the text (before the program listing), change the word "command" to "comment." My meaning here was to delete the text from the REM statement, but leave the REM.

In the program listing, line 3010 has a GOTO referencing a non-existent line 3150. This line was inadvertently omitted. Therefore, insert the following two lines:

```
3150 REM 01110110 'HLT' INSTRUCTION
3160 S$ = "HLT" : GOSUB 6000: RETURN
```

These lines are necessary for the disassembly of HLT commands. Their omission causes no problem until such a command is encountered. Although I had tested this program many times before submitting it, I apparently had not disassembled any programs using the HLT command until after I had mailed the program to you. I am now convinced of the programmer's law which states that every program contains at least one bug that is totally invisible to the programmer until after the program is published, at which time the bug will be immediately obvious to even the most casual observer.

I received a question about line 5530 which may have bothered someone else. The inquiry was whether the argument of CHR$(c) was a "c" or an "o." This character is a "c" and is the same "c" referred to in line 5510.

I now have a question. Do you know of any policy on rights to programs which have been converted from one machine to another, where the revision may be extensive (up to 50%), but some of the original good ideas have been kept? For example, I have converted several machine code programs written for the TRS-80 to run on my Heath H-8. Since the CPU chips are different and the architecture is quite different, the programs required substantial revision. I would like to publish these (especially a useful Monitor/Debugger) for use by other H-8 owners, since software is not so readily available for this machine. However, I don't want to infringe on any copyrights or get into any legal problems. I would appreciate any advice from you or your readers on this.

Thanks for the good job you're doing with *Dr. Dobb's*.

Sincerely,
Ronald T. Borochoff

3822 Murworth Drive
Houston, Texas 77025

The following mini program was sent to us by Holger Petersen of West Germany, whose Z80 Micro Disassembler was published in DDJ *#38.*

HOW TO CHESS-MATE MICROCHESS!

Dear Suzanne:

I send you this chess note to form a system-tape.

First step : Have a 16K level II TRS-80
second step : Have a Microchess-tape and T-BUG
third step : Load T-BUG, and with the T-BUG :
fourth step : Load the first take of MICROCHESS
fifth step : Change the following locations

```
4FB3 from   40 to   60
4FCA        4F      6F
4FD9        C2      00
4FdE        3C      70
4FE5        40      74
4FEA  CD  48 01 to  C3 XX YY (jump to your T-
```

sixth step : Load the second take of MICROCHESS by jumping from T-BUG to 4FA4!
seventh : Insert a short program at 6000 :

```
6000  21  CO 60   LD HL
6003  11  CO 40   LD DE
6006  01  00 10   LD BC
6009  ED  BO      LDIR
600B  21  00 70   LD HL
600E  11  00 3C   LD DE
6011  01  00 04   LD BC
6014  ED  BO      LDIR
6016  C3  FD 41   JUMP microchess        microchess
```

eighth step : Save all from 6000 to 73FF on a tape!
Start address is 6000!
ninth step: Try it out!

To note : And how wild you disguise your software, the first part of it must come into the machine the normal, vulnerable way!

UCSD PASCAL WITH DIGITAL SYSTEMS DDOS

Dear *DDJ:*

After our club made a group purchase of UCSD PASCAL over a year ago, I was anxious to try it out. I had enough memory, 48K, a Digital Systems (now Digital Microsystems) double density floppy disk, and CP/M 1.4, modified by Digital Systems to work with either double or single density.

The PASCAL was set up for single density, so following the instructions from Digital Systems, I constructed a single density operating system. It worked just as it was supposed to on regular CP/M files, and I was able to get into PASCAL and set it up to work with my terminal.

However, I soon became very frustrated. I could copy a complete PASCAL disk; I could change the date—which gets written to the disk; but I could not save a workfile or copy a file from one disk to another.

After trying all kinds of things with many suggestions from local club members, I despaired of getting PASCAL on my system. However, I eventually got a lead to Bruce Sherry in California, who had worked on Digital Systems software. He provided helpful suggestions that led to the solution of the problem.

The difficulty arises in the CP/M BIOS. Because the version from Digital Systems can handle single density in one drive and double density in another, a table in the BDOS has to be modified depending upon which density is in effect at a given time. This is done by the SELDSK routine in the BIOS.

Unfortunately, PASCAL uses the area normally occupied by the CP/M BDOS, so when SELDSK modifies BDOS, it is actually modifying part of the PASCAL operating system. As long as one is going to use PASCAL only in single density mode, there is no need for SELDSK to modify BDOS.

The cure is easy: simply eliminate a few lines from SELDSK, recompile BIOS and patch it into CP/M in the normal manner. If you are starting from scratch to make a single density operating system, don't forget to recompile BOOT so that the system comes up in single density.

Here, in condensed form, are the original routine and the modification. In the Digital Systems CBIOS version 4.1 listing, it is located just a few lines below the statement: " ; I/O DRIVERS FOR THE DISK FOLLOW".

```
SELDSK: ;SELECT DISK GIVEN BY REGISTER C
        XRA  A
        . . .
GETDR:
        . . .
SETSING:
        LXI  D,SINGTAB
        JMP  OVERBOTH       <--- Delete this line and all following
SETDOB:                          down through "* * * cure * * *"
        . . .
OVERBOTH:
        PUSH B
        LXI  H,BDOS+0FH     <--- This is the clue to the trouble!
        . . .
LP10:
        . . .
        POP B
                        * * * cure * * *
        LDA  PORT
        ANI  0F0H       ;CLEAR OUT OLD DISK SELECT BITS
        ORA  C          ;PUT IN NEW DISK SELECT BITS
        STA  PORT
        RET
```

The routine can be shortened even more, but I would gain nothing from saving the few bytes involved. After this has been tested to your satisfaction, you can construct the single disk boot that will take you straight into PASCAL following the UCSD instructions.

If anyone gets PASCAL going with Digital Microsystems double density software, I would be very pleased to hear about it.

Yours truly,
Charles F. Douds, Ph.D.

381 Poplar St.
Winnetka, IL 60093

PET TAPE CUrE

Dear *DDJ:*

As any PET user knows the most annoying part of loading from Cassette is waiting for the FOUND ' ' message.

This is mainly due to the lack of a tape counter, but even with a counter you would not know if the PET has passed the Program Header. You can sometimes miss the header, wait several minutes, only to find you are on blank tape. My method is as follows:

Connect a Soundbox to the user Port Pin 6 (Cassette No. 1 Read) The Soundbox connection is Pin M (CB2 Line).

On both SAVE and LOAD you can then hear the following:

a) The Header Tone, b) The Header Token, c) The Header 'Title', d) The Program DATA, e) The 'Half Way Point', f) Second copy of DATA, g) The end of file Token

By using the F .FWD, PLAY and REW keys you can then locate the header on a multi-program tape, press PLAY, and wait. If you do not get the message FOUND ' ' at the Header Title stage, rewind slightly and try again. Using this method you can CUE the tape to the right position.

Other advantages are that you can also hear:

a) DROPOUTS, b) CROSSTALK, c) NOISE, d) VARIATION in PITCH due to tight Cassettes, e) The difference between DATA and PROGRAM tapes.

This is an invaluable aid, and is best implemented by fitting a small toggle switch to the cover of the user port connector.

i.e. Position 1 SOUND (Pin M)
Position 2 OFF (No Connection)
Position 3 CASSETTE (Pin 6)
With Pin N being the 'earth'.

For those who like to keep a 'Working Copy' of their programs in addition to the 'MASTER' a separate cassette is an advantage.

I use an Hitachi TRQ 299 which has an automatic level control (ALC) and a Cue and Review facility. In my case the ALC gives perfect results on the PET recordings every time. The Cue and Review facility allows you to fast wind using Cue to find the 'nth' program on the tape.

Position the header using Review and transfer the tape to your PET Cassette.

Perhaps somebody will devise a method to convert the PET Cassette to 'Cue and Review'.

Incidentally, can anybody suggest a method of recovering data from a Program tape, on which the header and part of the first copy of DATA has been erased? (Caused by pushing RECORD instead of PLAY).

Sincerely,
R. Cason

15 Parkway,
Sawbridgeworth,
Herts, England

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 44 | April 1980 | Volume 5, Issue 4

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

An Emphasis on Algorithms!

An Introduction to Algorithm Design 5
Jon Louis Bentley

Algorithm: The first in a Monthly Column About Algorithms 44
Dennis Allison

Meta II: A Syntax-Oriented Complier Writing Language 17
D.V. Schorre

A Northstar BASIC Cross-Reference Tool 26
Titus Purdin

Some Useful Subroutines for a Z-80 Assembler 34
John W. Carter

A Fast Fourier Transform for the 8080 39
Robert D. Fusfeld

An Introduction to Algorithm Design

BY JON LOUIS BENTLEY
Carnegie-Mellon University
Pittsburgh, PA 15213

© 1979, IEEE Computer Society

"Algorithm design—that's the field where people *talk* about programs and *prove theorems* about programs instead of *writing* and *debugging* programs." Statements along these lines have been uttered by applications programmers and academicians alike. But there are also some who say, "No! Proper algorithm design has helped us to save kilobucks at our installation every month." Here we will investigate the field of algorithm design (also know as "analysis of algorithms" and "concrete computational complexity," among other names) and better equip the reader to judge the field for himself.

I trust that anyone with even the slightest love for mathematics burning somewhere in his heart (however deeply) will want to see how mathematical tools can be applied to the problems of programming. But for the other readers (whose interest in mathematics was probably squelched in freshman calculus), I offer the same bait that hooked me on this field. I trace my interest in the design of efficient algorithms to the time when I was a business data processing programmer and had just finished reading an introductory text on data structures. A colleague of mine had just had his program cancelled— the operators, by counting the turning rate of the tapes, had estimated that it would take about three hours to process his one reel of data. The program itself was fairly short and a quick glance told us that the time was spent in scanning a 1000-element table. I suggested that instead of scanning we try a new-fangled technique I had just read about—binary search. We did, and the modified program processed the reel of tape in *five minutes*, and actually spent most of its time waiting for the tape! Around that time I was also asked to help another programmer who had already spent a month producing over a thousand cards of code for a particular program. A simple change in data structure and a few days' work allowed us to redo the program in less than two hundred lines of code. The redone program was faster than the original would have been, used far less code, and was much easier to understand. So even if you have no aesthetic interest in algorithm design (yet), please read on—the practical benefits alone are rewarding enough.

Reprinted with permission from Computer, *February 1979.*

Throughout this article I refer only to the *discrete* aspects of algorithm design. I do not address problems in the domain of numerical analysis, such as stability, truncation error, and error propagation. Even with this restriction, I include some matrices, in which almost all elements are zero, and the fast Fourier transform.

A number of survey papers on discrete algorithm design have appeared recently. Hopcroft[1] and Tarjan[2] both give a thorough picture of the field. Weide's survey[3] concentrates on techniques for analyzing discrete algorithms, and accomplishes that task expertly. For those who are skeptical of sweeping surveys, Knuth's excellent introductions[4,5] examine a few problems in detail. And if one is ready to become a serious student of the field, the standard texts are Aho, Hopcroft, and Ullman's one-semester, graduate-level introduction[6] and Knuth's first three volumes[7-9] of his proposed seven-volume definitive work on algorithms. I hope to supplement those works by providing a broad survey for the novice. For this reason, I have kept my list of references short; both Tarjan and Weide, cited above, contain excellent bibliographies.

This survey discusses algorithm analysis from three viewpoints. We first examine five problems and some algorithms for solving them. Having considered these concrete examples, we next take a systematic tour of the field. We then investigate some current research directions and conclude with an assessment of the field's value to practitioners.

Some problems and fast algorithms for solving them

The five problems presented here serve as background to our general discussions of algorithm design. We will not only give an algorithm for solving each, but will also mention some real-world situations in which our problems could exist and to which our algorithms could be applied. We will analyze the algorithms' efficiency and discuss what this analysis shows. We will study the first problem—"subset testing"—in detail and then treat the remaining problems at a more superficial level. Before moving on to our overview of algorithm design, we will consider what all this attention to efficiency gets us.

But first let me offer an explanation of our concern with these examples. The subset testing problem will raise familiar issues and should cover some old ground for many; it also gives us a nice illustration of the tremendous time savings we can get by using proper algorithms. The next example—the substring searching problem—provides an extremely interesting blend of theory and practice. The fast Fourier transform—FFT—is the third example. It is known to many, uses some important algorithmic techniques, and is eminently practical. Fourth, we examine a very old problem—matrix multiplication—and a recent and remarkably counterintuitive solution. We conclude our examination of specific problems by investigating algorithmic aspects of a public-key cryptosystem. This system has recently revolutionized the world of cryptography and promises to have a substantial impact on "secure" computing.

Subset testing. Given a set A of size n and a set B of size $m \leqslant n$, is B a subset of A?[10] We can state this "subset testing" problem as a programming exercise: given an array A[1:n] and B[1:m], both of (say) 32-bit words, is every word in B also in A? Disguised versions of this problem arise in many contexts: set A could be an employee master file, set B a list of weekly transactions, and the problem, a question of finding whether a master-file record exists for each weekly transaction. Or A might be a table of real numbers x with an associated table S containing sine x, B then being a set of x values at which the sine function is to be evaluated. But even though this problem does have many such practical applications, that is not our main motive in examining it here. Rather, it leads to many of the basic problems in sorting and searching, and points to interrelationships between those problems. It also exposes us to some common algorithm design methods.

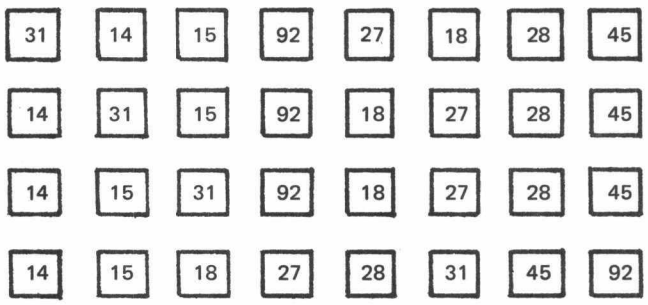
We will examine three solutions to this problem. Finding the running time of each, by counting the number of comparisons between elements, will enable us to compare the solutions. The following may also encourage the reader as he follows the different solutions: our first algorithm will require over six days of CPU time to solve a sample problem, while our final algorithm will solve it in four seconds.

Brute force. The simplest way to accomplish this task is to compare every element in B to each of the elements of A until either its equal is found or we have examined all of A and determined that every element of B has no equal in A (in which case B is not A's subset); this approach gives a simple, two-loop program. If B is indeed contained in A, then each scan for an element that is B's mate in A takes $n/2$ comparisons on the average (you have to look halfway down the list). Since there are m such scans made, the total number of comparisons made by this program is about $m(n/2)$. So if m is very close to the size of n, then we will make about $n^2/2$ comparisons on the average. We won't try to analyze the case that B is not a subset of A; to do so we would have to say exactly how it is not a subset, and that is very dependent on the particular problem. Although this algorithm is exceptionally simple to understand and to code, its slow running time might prohibit its use in certain appplications. We will now turn our attention to a faster algorithm.

Sorting. If you were given a randomly ordered list of phone numbers B (say a list of phone numbers in a town) and another randomly ordered list A (say all phone numbers in the county) and you were asked to check whether B was a subset of A (make sure every town phone number is included in the county list), then you might use the brute-force algorithm just discussed. If, however, you were handed a town phone book and a county phone book and asked to perform the same task, then your job would be much easier. Since the two phone books are already sorted by name, we can just scan through the two books together, ensuring that the county book contains all the town names. This of course immediately gives us another algorithm for subset testing: sort A, sort B, then sequentially scan through the two, checking for matches. To analyze the run time of this strategy we observe that the scan will take about n comparisons, and we heard somewhere that you can sort a list of size n in about $n \log_2 n$ comparisons, so the total running time is $(n \log_2 n) + (m \log_2 m) + n$ comparisons.

We could pull a sorting routine out of thin air, but it is not much more difficult to describe one called Mergesort. The basic operation of Mergesort is merging two sorted lists of numbers, say X and Y, with these lists either stored as arrays or linked together with pointers. To do this we compare the first element of X with the first element of Y and give the smallest as the first element of the new list, deleting it from its source. We repeat this remove-the-smallest step until both X and Y are empty. Since we used one comparison for each step, if there were a total of m elements in X and Y, we will have used about m comparisons. We can now use this tool of

Mergesort



The above illustrates the operation of Mergesort applied to a list of eight elements. The top row shows the eight elements in the order they were presented for sorting. We then view each successive pair of elements (31 and 14, 15 and 92, etc.) as pairs of one-element lists and merge those lists—this amounts to placing the smaller element as the first element in the list of length two. The second row depicts this result. We now have four lists, each of length two. As the next step of our algorithm, we merge the two successive pairs of lists, giving two (sorted) lists of length four. As the final step of our algorithm, we merge that pair of lists, resulting in the list of all elements in sorted order as shown in the bottom row.

This example had three "levels" of merging, each requiring work proportional to the number of elements in the list. In sorting an n-element list we will have $\log_2 n$ levels, with work proportional to n on each level. Careful analysis shows that if n is a power of two, we will use exactly $n \log_2 n - n + 1$ comparisons. If we assume that in a program implementing this algorithm each comparison costs approximately one microsecond, then we can sort a million elements in approximately 20 seconds. Using the well-known bubble sort for the same problem could require up to *six days.* (We can be encouraged by the fact that bubble sort will only take three days on the average—practice saying that with a straight face to your DP manager!)

merging to Mergesort a set S of n elements. We start by viewing S as a set of n sorted one-element lists. We then merge adjacent pairs of one-element lists, giving $n/2$ two-element lists. The next step is to merge adjacent pairs of those lists giving $n/4$ four-element lists, and the process continues. After $\log_2 n$ iterations we have one sorted n-element list, and our task is complete. To analyze this we note that we use about n comparisons for the merges at each of the $\log_2 n$ iterations, so the total number of comparisons used is the promised $n \log_2 n$.

We have thus shown how to solve the subset problem with $n(\log_2 n + 1) + m \log_2 m$ comparisons. If m is about the same size as n then our algorithm takes approximately $2n \log_2 n$ comparisons. Can we do better?

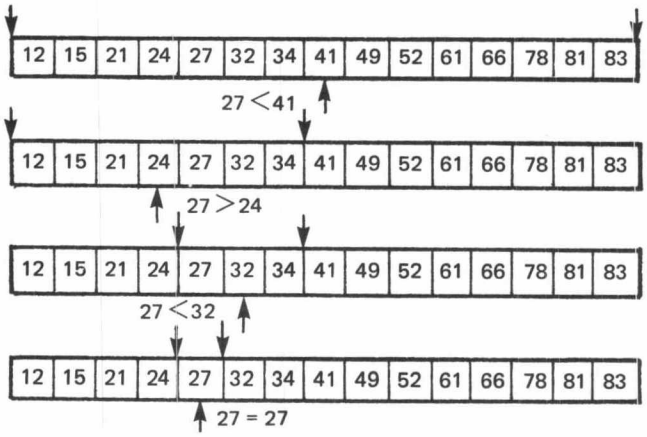
Hashing. Thinking about the phone book problem leads to an interesting sorting approach to the subset problem; if we rephrase the phone book problem then the "human" approach will lead to an even faster subset algorithm. Suppose that the county phone book (A) was sorted and the town phone list (B) was not; to ensure that A contains B we can "look up" in A each number in B by the name of the subscriber. For each of the m elements in B we would do a "binary search" among the n elements of A. It is not hard to see that a binary search in an n-element sorted table takes at most $\log_2 n$ comparisons, so this algorithm is easily analyzed: it takes $n \log_2 n + m \log_2 n$ comparisons, or approximately $2n \log_2 n$ if m is the same size as n. We therefore have a searching solution to the subset problem: store the elements of A in a table, then for each element of B ensure that it is in the table.

Although binary search is the best searching method for many problems, there is another strategy—hashing—even more appropriate for this problem. By hashing we can store an element in a table or check to see if it already is in a table in about two comparisons, on the average. For pessimists, however, we note that the worst case of hashing is as bad as brute force—we might have to look at all of the elements in the table. With this approach we will be able to do subset testing in $2n + 2m$ comparisons—$2n$ to store A and then $2m$ to look up each element of B. To store the n elements of A we will have to allocate a *hash table*—an array of length $(1.5)n$. We can even use a smaller array; $(1.1)n$ would probably work almost as well. We then store the elements of A in the table one-by-one by the use of a *hash function.* This maps a data value into an integer in the bounds of the hash table. If that position in the hash table is empty, fine. We insert the element. If the position is occupied, however, we have a *collision* and must employ a *collision resolution strategy,* such as scanning up the elements of the array until a free position is found. Analysis shows that a proper collision resolution strategy allows us to find an empty spot very quickly—say, in two comparisons. When an empty spot is finally found the element is inserted. After inserting all of A's elements into the table we then look up all of B's elements. For any particular element we calculate its hash function and look in that position. If that position is empty then it is not in A; if the element is in the position then we have found it; otherwise we must employ the same collision resolution strategy to see where it should be. Hashing is something that a human would never use in searching (humans are much better at comparing things and then looking in one of two directions than at calculating weird hash functions), but it leads to a very efficient algorithm. If m is about the same size as n then the hashing approach uses only about $4n$ comparisons, on the average, to do subset testing.

The subset testing problem, although a very simple one, led us straight to some of the fundamental issues in algorithm design. We very quickly arrived at searching—the scan of the brute force algorithm was just a naive search. From there we moved to sorting, then to binary search, and finally to hashing, which introduced us to a non-obvious data structure—the hash table. A more thorough examination of searching is contained in Knuth.[11] The approaches that we used to solve the problem are some of the fundamental tools of algorithm designers. We have also touched on aspects of algorithmic problems such as time and space analysis and worst-case versus expected-time analysis, which we will study later.

But what has all this gained us? Although we may have a better understanding of some of the basic computational issues, does our understanding make any difference in practice? Let's assume that we are writing a program for subset testing where A and B both contain one million elements, and that one comparison takes, say, one microsecond of computer time. Under these assumptions, the $n^2/2$ comparisons required by brute force translates to 138 hours, or a little shy

Binary search



This figure shows the operation of binary search in locating the position of the number 27 in a sorted table. At the start of binary search we only know that the element, if in the table at all, lies between 12 and 83, inclusive. This is denoted by the two "down-arrows" at the top row. We then compare the middle element of that sequence—41—27. Since 27 is less than 41 we can exclude position 41 to 83 from our search, denoted by the down-arrows in the second row. Binary search continues by always examining the middle element of the sequence in which the "target" value is known to lie. We show that middle element by an "up-arrow" below it. The binary search terminates by either finding the desired element or showing that the desired element is not in the table.

To analyze binary search we note that each time it makes a comparison the size of the subtable containing the desired elment is *cut by one-half.* For example, if we search a 1000-element table, after one comparison we are down to only 500 elements, after two comparisons to 250, and so on. After about ten comparisons we either find our man or discover he is not in the table. In general if there are n elements in the table, binary search requires approximately $\log_2 n$ comparisons. Thus, if we use it to search a one-million-element table, it will rquire only 20 comparisons, a substantial improvement over the ***half-million*** comparisons used by the average linear scan.

of six days, of machine time; the $2n \log_2 n$ for sorting will give 40 seconds; and the $4n$ of hashing will yield 4 seconds. Although we haven't calculated all the costs of implementation, this example shows how a simple analysis is sometimes all we need to make an informed choice.

Substring searching. Does a given *string* contain a specified substring *pattern*, and if so, where? This is the substring searching problem, one familiar to most who have used computer text editors. As I sat down to type this paragraph, for example, I told the editor to find the substring "Substring searching." in my text file so I would know where to begin my paragraph. The text editor I use looks at a file as one long string of text, sprinkled with special characters representing "carriage return." Information retrieval systems use this same operation as they identify abstracts which contain certain keywords. Similar problems are encountered in many text formatting and macro processing programs.

It is not hard to write a program to solve this problem. We first hold *pattern*'s leftmost character under *string*'s leftmost character and start comparing. If all the characters of *pattern* match the characters above them, fine—we have found the substring in position 1. If we find a mismatch then we slide *pattern* over one and do the same thing again. This continues until we either find a match or come to the end of *string*. The worst-case behavior of this algorithm is very slow—for each n positions of *string* we might have to compare all m positions of *pattern*. Thus in the worst case we might have to make mn comparisons. Strings and patterns that realize this worst-case behavior are fairly pathological and the performance of this algorithm in practice is fairly good, but the question still haunts us—can we design an algorithm that will always do better?

Knuth, Morris, and Pratt[12] offer an algorithm that beats the mn performance. They preprocess *pattern* into a data structure that represents a program; that program then looks for *pattern* in *string*. Preprocessing *pattern* takes only m operations (where m is the length of *pattern*) and the "program" they produce looks at each character of *string* only once, so the total running time of their algorithm is proportional to $m + n$ instead of mn. (Of course if the *pattern* is in the ith position in *string*, then their algorithm takes time proportional to $i + m$.) This result is exceptionally interesting from a theoretical viewpoint, and also provides a faster substring searching algorithm in practice.

Boyer and Moore[13] recently used the basic idea of the Knuth, Morris, and Pratt algorithm to derive an even faster method of substring searching. Their method has the same worst-case performance (proportional to $m + n$) but is faster on the average. They accomplish this by making it unnecessary to examine every element of *string*. They have implemented their algorithm on a PDP-10 so efficiently that when *string* contains typical English text and *pattern* is a five letter word in *string*, the number of PDP-10 instructions executed is less than $i + n$. This is at least an order of magnitude faster than the naive algorithm.

The history of the substring searching problem provides an insight into the relation of theory to practice. Knuth relates that his use of a machine from automata theory, the "two-way deterministic push-down automaton," led him to his discovery of the algorithm. The easiest way to understand fast algorithms is through the use of finite state automata, which are commonly used in digital systems design. It is noteworthy that in this one problem we can talk about such diverse ideas as abstract automata and PDP-10 instructions, with a lot of combinatorial analysis in between!

The fast Fourier transform. The Fourier transform is often studied in mathematics and engineering. It can be viewed in a number of ways, such as the transformation of a function from the "time domain" into the "frequency domain" or as the decomposition of a function into its "sinusoidal components." The continuous Fourier transform has a discrete counterpart, which applies an operation to one set of n reals yielding a "transformed" set of n reals. This problem has applications in signal processing, interpolation methods, and many other discrete problems.

The naive algorithm for computing the Fourier transform of n reals requires approximately n^2 arithmetic operations (adds and multiplies). The fast Fourier transform of Cooley and Tukey[14] accomplishes this task in approximately $n \log_2 n$ arithmetics. It achieves this by doing about n arithmetics on each of $\log_2 n$ levels; in this sense it is quite similar to the Mergesort algorithm discussed earlier. There are many different expositions of the algorithm, such as those in Aho, Hopcroft, and Ullman[15] or Borodin and Munro.[16] It is interesting to note that in addition to being faster, many of the numeric properties of the FFT are better than those of the naive transform.

The fast Fourier transform has had a substantial impact on computing. It forms the backbone of many "numeric"

Matrix multiplication

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} p & q \\ r & s \end{bmatrix}$$

We can compute a product matrix in the following (non-straightforward) fashion. First we compute the seven "temporary" values by the following seven multiplications:

$$t1 = (b - d) * (g + h)$$
$$t2 = (a + d) * (e + h)$$
$$t3 = (a - c) * (e + f)$$
$$t4 = (a + c) * h$$
$$t5 = a * (e - f)$$
$$t6 = d * (g - e)$$
$$t7 = (c + d) * e$$

We can now combine these temporary values to yield the product matrix:

$$p = t1 + t2 - t4 + t6$$
$$q = t4 + t5$$
$$r = t6 + t7$$
$$s = t2 - t3 + t5 - t7$$

The "high school" method of multiplying two-by-two matrices requires eight multiplications and four additions. This method requires only seven multiplications, but 18 additions and subtractions. This hardly seems like a worthwhile tradeoff!

Amazingly enough, however, we can use the above scheme *recursively* to oproduce an algorithm for matrix multiplication that is asympototically more efficient than the $O(n^3)$ high school algorithm—it requires $O(n^{2.81})$ time. Although the savings it achieves are offset by the additional overhead in implementation (it is probably inferior to the high school algorithm until the matrix contains at least 10,000 elements), it is remarkable to find we can multiply faster than people have been doing it for the past hundred years.

programs. Practitioners in diverse fields use the FFT to find hidden periodicities of stationary time series. Signal processing uses it in filters to remove noise from signals and eradicate blurring in digital pictures. Numerical analysis utilizes it for the interpolation and convolution of functions. Applications of the FFT in such diverse areas as electrical engineering, acoustics, geophysics, medicine, economics, and psychology are listed by Brillinger.[17] Many special-purpose processors have been built which implement this algorithm; some are multiprocessors which operate in parallel. The FFT is also widely used in the design of "discrete" algorithms. It is the primary element in many algorithms which operate on polynomials, performing such operations as multiplication, division, evaluation, and interpolation. Not surprisingly, it is also employed in some of the fastest known algorithms for operating on very long integers, such as multiplying two 1000-bit integers. We will see an application of this problem later.

Matrix multiplication. One of the most common ways of representing many different kinds of data is in a matrix, and one of the most common operations on matrices is multiplication. How hard is it to multiply two $n \times n$ matrices? Using the standard high school method takes about $2n$ arithmetic operations to calculate each of the n^2 elements of the product matrix, so the total amount of time required by that algorithm is proportional to n^3. People have been multiplying matrices by this method for a century. Surely this must be the best possible way to multiply matrices—our intuition tells us that we just can't do any better.

The high school algorithm for multiplying two-by-two matrices uses eight multiplications and four additions. It is fairly counterintuitive to learn that the product can be computed using only seven multiplications at the cost of an increase of 15 additions. But if that is counterintuitive, then it is absolutely mindboggling to find that this fact alone allows us to construct an algorithm for multiplying $n \times n$ matrices that runs in less than n^3 time! This algorithm, devised by Strassen,[18] decomposes each $n \times n$ matrix into four $(n/2) \times (n/2)$ matrices. It does seven multiplications of $(n/2) \times (n/2)$ matrices and then fifteen additions on matrices of that size to find the product of the original matrices. Notice, however, that the cost of those additions is proportional to n^2. If we let $T(n)$ be the time required to multiply $n \times n$ matrices, then $T(n)$ satisfies the recurrence

$$T(1) = 1,$$
$$T(n) = 7T(n/2) + O(n^2)$$

having the solution $T(n) = O(n^{2.81})$ (where 2.81 is an approximation to $\log_2 7$). The Strassen algorithm as originally presented is less efficient than the high school algorithm until n is in the thousands. Recent work, however, shows that it can be practical when n is as small as 40. But practice aside, who can help but be amazed by the fact that we can multiply matrices faster than we thought we could?

The fast matrix multiplication algorithm provided the basis for one of the all-time great revolutions in the history of "theoretical" algorithm design, during which a number of "best" known algorithms were toppled from their thrones. Many of these were n^3 matrix algorithms which we can now do in $O(n^{2.81})$ time; among these are matrix inversion, LU decomposition, solutions to systems of linear equations, and calculations of determinants. A number of problems which seemed to be totally unrelated to matrices were phrased in terms of them. As a result $O(n^{2.81})$ algorithms followed for such diverse problems as finding the transitive closure of a graph, parsing context-free languages (an important problem in compilers), and finding distances between n points in Euclidean n-space. All of these algorithms stem from the fact that two-by-two matrices can be multiplied with seven multiplications.

Public-key cryptography. Communications systems which deal with the problem of transmitting a message from a sender to a receiver across an insecure and possibly bugged channel, while protecting the privacy of the message, are known as *cryptosystems*. Such security needs often arise in military applications; they also promise to play an ever increasing role in applications such as electronic mail and electronic funds transfer. A cryptosystem is usually implemented by *encoding* and *decoding algorithms* which transform their inputs according to *keys* they are given. To send person X message M we use the encoding algorithm and key to produce message M′, transmitting M′ to X across the possibly insecure channel. When X receives M′ he uses the decoding algorithm and key to determine M, while any "eavesdropper" has only M′. One difficulty with this system is the distribution of keys to the appropriate parties. This must usually be accomplished by the use of expensive secure channels such as human couriers.

An alternative, called a *public-key* cryptosystem, was recently invented by Hellman and Diffie.[19] In their system each person has an encoding key and a decoding key, as before. To send a message to person X we encode it using X's encoding key, and then X can decode it with his decoding key. The novel aspect of this system is that we can make the encoding key public without revealing the corresponding decoding key. We can then view the encoding key as the address of X's mail box, and anyone can put mail into that

O-notation

Algorithm analysts frequently employ the "big-oh" notation of mathematics. In this notation we can write an expression like

$$T(n) = 6n^2 + 9.3n = O(n^2)$$

to show the running time—$T(n)$—of an algorithm. The more accurate expression for $T(n)$ is much more complicated to express than the simple $O(n^2)$; we therefore see a tradeoff between the accuracy of the statement and the ease of expressing the formula (as well as the ease of deriving the expression). There is a well-developed mathematical theory of "order arithmetic" that allows derivations to be made without regard to the constant factors of the leading terms. Most calculations in this arithmetic are very simple.

"Big-oh" notation is a useful tool for analyzing algorithms. The running times of most algorithms to within "big-ohs" are much more easily obtained than are the exact analyses of the algorithms. If we know that algorithm A takes $O(n^2)$ time and that algorithm B takes $O(n \log n)$ time, then regardless of the constants "hidden" in the notation, algorithm B will be faster than algorithm A for large enough values of n. The "big-oh" analysis will therefore be of great help in choosing an algorithm that is to process a huge amount of data, as long as we are careful not to be caught unaware by the constants hidden in the notation.

box simply by encoding the message. Actually unlocking the box, however, requires the decoding key, which only X possesses. Although such a system solves almost all of the difficulties of previous cryptosystems, one major obstacle remains; for most codes knowledge of the encryption key immediately reveals the decryption key. Thus all we need to complete our public-key cryptosystem is an appropriate encoding/decoding algorithm, provided such an algorithm is possible.

A suitable encoding/decoding method was recently developed by Rivest, Shamir, and Adleman.[20] Their method is based on algorithmic issues in the theory of numbers. They view messages as multiprecision (long) integers of 200 decimal digits, for example. The coding procedure then transforms these messages by sophisticated use of modular arithmetic and prime number theory. The transformations require many sophisticated algorithms. For instance, the process of key selection is based on fast algorithms for multiplying multiprecision integers and testing such integers for primality; fast multiprecision exponentiation algorithms then perform encoding and decoding. The fact that any method that "breaks" the system must essentially find the factors of a very large number is the basis of the system's security. Although no one knows precisely how difficult this factoring is, mathematicians have been working on the problem for many centuries, and no one yet knows of a fast method. Let me put this in perspective: if we deal with 200 decimal digit integers, then the key selection, encoding, and decoding algorithms require only a few seconds of CPU time; the best known method for breaking the system, however, would require *ten million centuries* of CPU time.

We have only scratched the surface of the fascinating field of public-key cryptography. In addition to use in cryptosystems, these methods can also be used to provide "electronic signatures," i.e., verifications of identity. This cryptosystem is another interesting example of the interactions between theory and practice. It is based on number theory (perhaps the purest of the areas of pure mathematics) and complexity theory, yet it promises to revolutionize the practice or cryptography. The interested reader should refer to the article by Rivest, Shamir, and Adleman, cited above, or the exposition in Gardner.[21] Although the algorithms we have mentioned do not really solve an existing computational problem, they solve a problem in a totally different area by casting it in a computational light.

We have now examined five cases in which proper algorithm design has led to a sophisticated algorithm that is much faster than a naive algorithm. Algorithm designers have invested a lot of work in developing these algorithms—but what difference will it all make in practice?

To be honest, most of the time a fast algorithm makes no difference at all. Knuth has gathered empirical evidence showing that most of the run time of a program is spent in just three precent of the code. If an algorithm for a problem is not in the critical three percent of a code, it makes little difference if the algorithm is fast or not. A complicated algorithm is often a liability rather than an asset. It will usually mean more coding and more debugging time, and can sometimes even increase the run time (when the overhead of "starting up" a fancy algorithm costs more than the time it saves).

Sometimes, however, a fast algorithm can make all the difference in the world. If a certain computation is indeed the bottleneck in the system flow, then an algorithm of half the running time almost doubles system throughput. Many text editors spend a vast majority of their time in string searching; the fast algorithm for substring searching can speed up many text editors by a factor of five. My experience with the searching program I mentioned in the introduction, where we reduced the running time from three hours to five minutes, is another good example of an appropriate use for a fast algorithm. In the inner loops of many programs proper algorithm design is critical.

Fast algorithm design, although only occasionally yielding large financial savings, almost always gives us something of a different value—a fundamental understanding of our computation problems. This is usually reflected in cleaner programs; but even more important is the understanding of how diffcult it is to compute something. After a student has spent a month or two investigating the problem of searching, he knows not only how to search faster, but also why he can do it that fast and why he can't do it any faster. Such a student has learned something of the foundations of his field.

Algorithm design—a systematic view

In the section above we saw a number of specific problems and specific solutions; here we will show that there is more to the field than isolated examples. Algorithm design is a coherent discipline—one needs a specific set of concepts to define a computation problem and specific set of tools to design an optimal algorithm to solve it.

Problem dimensions—a microscopic view. The subset testing problem showed that there can be many different algorithms for solving a particular problem. In order to say which one is best in a particular application we have to know certain *dimensions* along which to measure properties of the algorithm. For example, in one application we may need a subset algorithm that must be very space-efficient and have good worst-case running time; in another context, we might have a lot of available space and only require good expected running time, not caring if we infrequently must take a lot of time. We have thus identified three dimensions of a computational problem: time analysis, space analysis, and expected vs. worst-case analysis. We will now investigate these and other dimensions of computational problems.

Time and space analysis. The two most important resources in real computational systems are time (CPU cycles) and space (memory words). These are therefore the two dimensions of a problem most frequently studied. Running time was the primary subject we examined in the examples cited earlier. Most of the algorithms we discussed used very little extra space after storing the inputs and outputs—the hashing algorithm we used for subset testing was the only exception. Since in large computer sytsems we can have huge quantities of extra space for the asking and the paying, we have often ignored the space requirements of algorithms. With the rise in popularity of mini-and microprocessors with very small memories, however, space analysis is once again an important issue.

Model of computation. In covering the examples earlier in this article, we were able to make reference to the time and space requirements of various algorithms without reference to their implementation on any particular computer. Our intuitive notions were robust enough to lead to sophisticated

algorithms that would certainly beat their naive competitors on any existing machine. But to analyze an algorithm in detail we must have a precise mathematical *model* of the machine on which the algorithm will run.

We could choose as our model a particular computer, such as an IBM650 or a DEC-PDP-10, and then ask how many microseconds of time or bits of storage a particular algorithm requires. There are two problems with this approach. First, we will problably analyze the expertise of the algorithm's implementer more than we will the algorithm's intrinsic merit; and second, once we have completed an analysis using an IBM 650 we still know very little about the algorithm's behavior on a PDP-10. One way of dealing with these difficulties is to invent a representative computer and then compare the performances of competing algorithms on that machine. Knuth[22] has described one such machine which he named the MIX computer. It has much in common with most existing machines without many idiosyncrasies of its own. If algorithm A is faster than algorithm B when implemented on MIX, then it is likely to be faster on most real machines, too.

Another solution is not to analyze the implementation of the algorithm on any particular machine at all, but simply to count the number of times the algorithm performs some *critical operation*. For the analysis of the FFT and matrix multiplication we chose to count the number of arithmetic operations. We know that the FFT uses exactly $n \log_2 n$ multiplications. To estimate its running time for a given implementation we can look up the execution speeds of the instructions around the muliplication instruction, sum those, and then multiply by $n \log_2$ to get an estimate for the running time. It is usually easy to determine the running time of a particlar program if we know the number of times the critical operation is to be performed. We must be careful, however, not to ignore certain "bookkeeping" operations that may become critical in implementations. Once we have chosen a critical operation to count it is very easy to specify a model of computation. To count arithmetic operations we usually employ the "straight-line program" model in which an algorithm for a particular value of the problem size (n) is represented by a sequence of statements of the form

$$X_i \leftarrow X_j \text{ OP } X_k$$

where OP is add, subtract, multiply, or divide. If the sequence for a particular value of n is m instructions long then we say that the execution time of our program is $T(n) = m$. If our critical operation were comparison, then we would probably choose the "decision tree" model. These and other models are described by Aho, Hopcroft, and Ullman.[23]

The above models allow us to analyze algorithms for their suitability as "in-core" programs on single-processor machines. If a program has very little main memory available and must store most of its data on tape, then some tape-oriented model such as the Turing machine is the most accurate model of the computation. If a program is to be run on a multiprocessor machine then our model must express this fact, and the model employed will vary with the multiprocessor architecture. The interested reader should refer to Kung[25] for a discussion of some of these issues from an algorithmic viewpoint. Many other models of computation have been proposed to describe diverse computing devices. The two important things in choosing a model are that it be *realistic*, so the results will apply to the situation it purports to model, and that it be mathematically *tractable*, so we can derive those results.

Exact or approximate analysis. Once we have chosen a model of computation we can analyze an algorithm's performance by counting the resources (time or space) it uses as a function of n, the problem size. How accurately should we do that counting? We could be very precise, calculating the answer exactly, or we might settle for an approximate answer. There are levels of approximation, all the way from the first two terms of the answer to rough upper and lower bounds. It is certainly desirable to get the exact answer, but this is sometimes very difficult. The first one or two terms of the cost function are adequate for most purposes, and in many cases we need only the function's asymptotic growth rate. We saw in the subset testing problem an example in which the run time for one program was 138 hours while another program took just 4 seconds to carry out the same task. Even if our analysis had missed a factor of ten, that could not affect our choice for large problems.

We often use the "big-oh" notation to describe the complexity of a problem. No matter what the respective constants are, an $o(n \log_2 n)$ algorithm will be faster than an $O(n^2)$ algorithm for large enough n. As we solve larger and larger problems by computer, we are more and more frequently in the domain of "large enough n." Asymptotically fast algorithms also have another advantage. If we get a new machine one hundred times faster than our old one, using an $O(n \log^2 n)$ algorithm will allow us to solve a problem almost one hundred times larger in the same period of time. Using an $O(n^2)$ algorithm will permit the new machine to solve a problem only ten times larger in that time. Thus, a function's asymptotic growth rate is usually enough to tell us how much an increase in problem size will cost under each type of algorithm.

Average or worst-case analysis. Many algorithms perform a sequence of operations independent of their input data; the FFT and matrix multiplication algorithms are both data-independent. The analysis of a data-independent algorithm is straightforward—we simply count the number of operations used. The operation of algorithms such as those for sorting and substring searching, however, are dependent on their input data. One algorithm can have very different running times for two inputs of the same size. How do we describe the running time of such an algorithm? Pessimists want to know the worst-case running time over all inputs and realists want to know the average running time. (We are rarely concerned with the best-case running time, for there are very few optimists in computing.)

Most mathematical analysis of algorithms has been done for the worst case. Even in data-dependent algorithms it is usually easy to identify the worst possible occurrence and analyze it as in a data-independent algorithm. In certain applications (air traffic control is often cited) it is important to have an algorithm that never surprises us with a very slow case. For most applications, however, we are more interested in what will usually happen; expected-time analysis provides this information. Relatively little work has been done on expected-time analysis. The two major stumbling blocks are in the choice of a realistic and tractable probability model of the inputs and in the intrinsic difficulty of dealing with expectations instead of single cases. It would be desirable to have a single algorithm efficient in both expected and worst-case performances.

Upper and lower bounds. Most naive sorting algorithms (such as "bubble sort") require $O(n^2)$ comparisons in the worst case. Earlier we investigated Mergesort, which never uses more than $O(n\log_2 n)$ comparisons. Should we continue our search, hoping to find an algorithm that uses perhaps only $O(n)$ comparisons? The answer to this question is no, for we can show that *every sorting algorithm must take at least* $O(n\log_2 n)$ *comparisons in the worst case*. The proof of this theorem uses the "decision tree" model of computation, described nicely by Aho, Hopcroft, and Ullman.[24] The Mergesort algorithm gave us an *upper bound* of $O(n\log_2 n)$ on the complexity of sorting; this theorem gives us a *lower bound*. Since the two have the same growth rate, we can say that Mergesort is *optimal* to within a constant factor under the decision tree model of computation. Notice that we have now made the important jump from speaking of the complexity of an algorithm to speaking of the complexity of the problem.

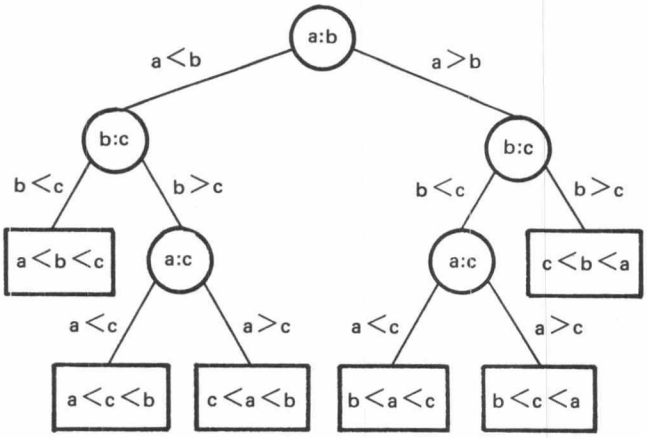
Lower bound results are usually much more difficult to obtain than upper bounds. To find an upper bound we need only give a particular algorithm and analyze it. For a lower bound, however, we must show that in the set of algorithms for solving the problem there are none more efficient than the lower bound. There are some trivial lower bounds we can easily achieve: most problems require examination of all their inputs, so we usually have an easy lower bound of the input size. The number of nontrivial lower bounds discovered to date is very small.

A precise model of computation is important in proving lower bounds. Previously we gave three algorithms we can use for testing set equality: brute force, sorting, and hashing. We gave them originally for subset testing, but recall that two sets are equal if and only if each is a subset of the other. The importance of the computational model becomes clear when we learn that we can prove each of those algorithms optimal under different computational models! Brute force's $O(n^2)$ performance is optimal if only equal/not-equal comparisons can be made between elements of the two sets. If the model of computation includes only less-than/not-less-than comparisons, sorting's $O(n\log_2 n)$ comparisons are optimal. If the model is a random access computer such as MIX, hashing's average-case linear performance is provably best.

Exact and approximation algorithms. The best-known algorithms for many problems are quite slow, requiring, say, $O(2^n)$ time. A very few of these problems have actually been proved to have exponential lower bounds. Others belong to a fascinating class called the NP-complete problems, either all solvable in time polynomial in problem size, or all of exponential complexity. Unfortunately, nobody yet knows which, but most of the money is on exponential. Examples of NP-complete problems include the traveling salesman problem (finding a minimal-length tour through a set of cities), bin packing, and the knapsack problem. Literally hundreds of problems are known to be NP-complete. Lewis and Papidimitriou[26] provide an excellent introduction to this class. There are other problems, not proven to be hard, for which no one has yet been able to design fast algorithms. So when we have a problem we don't know how to solve efficiently, what can we do?

The answer is amazingly simple: don't solve it. Solve a related problem instead. Instead of designing an algorithm to produce the exact answer, we can build one to produce an approximation to the exact answer. So instead of finding a minimal tour for the traveling salesman, we might provide him with a tour we know is no more than 50 percent longer than the true minimum. Or if someone asks if a number is prime or composite, instead of providing the true answer we might respond "I don't know, but I'm 99.999999 percent sure that it's prime." Problems abound for which the best known exact algorithms require exponential time, but for which we can quickly find approximate solutions. Garey and Johnson[27] examine these issues.

Decision trees



This figure illustrates a decision tree program to sort three numbers. The program "starts" at the top node and compares the value of *a* with the value of *b*. If *a* is the lesser it goes down the left branch of the tree and if *a* is greater it goes down the right branch (we assume for clarity that all elements are distinct). It continues until it reaches a "leaf" (leaf nodes are depicted by rectangles); when we arrive at a "leaf" we know the total ordering of *a, b,* and *c*. I encourage the reader to study this example to see that it correctly sorts the three elements.

Decision trees accurately model computations based on comparisons. Theoretically, we can represent any comparison-based program for sorting n numbers with a corresponding decision tree. For example, we can "translate" the above tree into a set of nested IF statements in a high-level programming language. We can use the mathematical properties of decision trees to prove lower bounds on the complexity of certain computations. Notice that the height of the decision tree (i.e., the maximum distance from the root to one of the leaves) corresponds to the maximum number of comparisons ever used by the "decision tree program." It is a well-known mathematical fact that if a binary tree has m leaves, it must have neight of at least $\log_2 m$. It is also known that there are $n!$ (i.e., $n * (n-1) * \ldots * 1$) different permutations of n elements. Since a sorting program must correctly distinguish between all permutations of its input variables, its decision tree must have at least $n!$ leaves. Combining these two facts, we see that any decision tree for sorting must use at least $\log_2 n!$ comparisons, which is approximately $n \log_2 n$. We thus see that the Mergesort algorithm for sorting is very close to the best possible algorithm in terms of number of comparisons used.

We must be very careful in interpreting lower bounds—they only apply in the given model of computation. It might be possible to sort numbers faster if we use more powerful operations than comparisons. (In fact, it is!) The lower bound, however, still serves a useful purpose. It tells us that if we want to do something faster, we had better adopt a *fundamentally different approach* to solving the problem.

These problem dimensions—such as time analysis, space analysis, and expected vs. worst-case analysis—are the categories in which algorithm designers think. When someone brings a designer a problem, his first task is to understand the abstract problem. Next he must understand what kind of solution the person wants, and he uses the dimensions to describe the desired solution. Using the vocabulary introduced here it is easy to describe concepts such as a "fast expected time and low worst-case storage approximation algorithm for task X, to be run on a multiprocessor machine." There are other infrequently used dimensions we have not covered, such as code complexity—how long is the shortest program to solve a problem? But the dimensions we have examined can describe most algorithmic results.

Problem areas—a macroscopic view. The vocabulary above can describe a particular algorithmic problem at a precise level of detail. Here we change our perspective and examine large classes of problems, but use the same terminology.

Ordered sets. There are many problems on sets that depend only on a "less than" relationship being defined between the elements of the set. In many cases the set contains integers or real numbers, while in other cases we define a "less than" relation between character strings (JONES is less than SMITH). The problems discussed earlier in the section on subset testing all deal with ordered sets. The algorithms in that section are appropriate if the elements of the sets are numbers, character strings, or any other type of "orderable" object. Knuth[28] provides an excellent introduction to the applications of, and algorithms for, ordered sets.

The "median problem" is another problem defined for ordered sets: given an n-element set we are to find an element which is less than half of the elements and not less than the other half. A naive algorithm would count for each element the number of elements less than it, and then report the median as the element with exactly half the others less than it. This algorithm makes approximately n^2 comparisons. We can devise an $O(n\log_2 n)$ algorithm by sorting the elements and then reporting the middle of the sorted list. In 1962 C. A. R. Hoare described an algorithm for the median problem, with linear expected time. For over ten years no one knew if there was an algorithm with linear worst-case time; Blum et al. finally offered one in 1973.[29] Much additional work has been done on this problem, exploring facets such as minimal storage, detailed worst-case vs. expected running time analysis (upper and lower bounds), and approximation algorithms.

Algebraic and numeric problems. Many aspects of algebraic and numeric problems have a discrete flavor, and discrete algorithm design can play a significant role in such problems. Matrix multiplication is perhaps the clearest example: we can describe (and appreciate) the fast algorithm without reference to any of its numeric properties. We can also view the FFT "non-numerically." Another numeric problem that can assume a purely discrete character is sparse matrix manipulation (in a sparse matrix almost all elements are zero). Borodin and Munro, cited earlier, describe applications of discrete algorithm design to numeric problems such as polynomial manipulation, extended precision arithmetic, and multiprocessor implementations of numeric problems.

Graphs. Graphs can represent many different kinds of relations, from the interconnections of an airline system to the configuration of a computer system. Tarjan's survey[30] discusses many computational problems relating to graphs. One important problem involves determining if we can imbed a given graph in a plane without any edges crossing. We might use this to check if we can imbed the connections of a given circuit on a printed circuit board or integrated circuit. The first algorithms for testing planarity ran in $O(n^3)$ time on n-node graphs; after much effort by many researchers, Hopcroft and Tarjan finally designed a linear-time planarity algorithm in 1974.[31] Another graph problem is constructing a minimal spanning tree of a weighted graph, i.e., a minimal-weight set of edges connecting all nodes. Researchers have proposed and analyzed a wide variety of algorithms for this problem. Some are superior for very dense graphs, others for relatively sparse graphs, and still others for planar graphs. Efficient graph algorithms exist for problems such as computer program flow analysis and finding maximal flows in networks. And since a sparse matrix is usually represented by a graph, graph algorithms can apply to sparse matrix problems.

Geometry. Shamos' paper[32] is an outstanding introduction to computational geometry, the field concerned with developing optimal algorithms for geometric problems. Many application areas are inherently geometric (such as circuit layout) and other problems can be viewed geometrically (such as treating a set of multivariate observations as points in a multidimensional space). Shamos describes an important structure, the Voronoi diagram, which allows many geometric problems dealing with n points in a plane to be solved in $O(n\log_2 n)$ time. Among these are determining the nearest neighbor of every point, and constructing the minimal spanning tree of the point sets, both important tasks in many data analysis procedures and previously requiring $O(n^2)$ time. Many other important problems have been solved after being cast in a geometric framework. One result, for example, revealed that the standard simplex method of linear programming is not optimal for two- and three-variable programs with n constraints. The simplex method has worst-case running times of $O(n^2)$ and $O(n^3)$, respectively; an $O(n\log_2 n)$ method has proved optimal for both the two- and three-variable case.

Other areas. We have glimpsed a few fields studied by algorithm designers. The results in many other areas must go unmentioned. These include algorithms for problems in compilation, operations research, data base management, statistics, and character string handling. These results have led both to fast algorithms and to a new *algorithmic* understanding of the various fields.

The "building blocks" of algorithm design. Wandering through a computer room we cannot help but be impressed by the complexity of a large-scale computing system. The novice might find it hard to believe that the human mind could design anything so complicated. Although he is not too far from the truth, many undergraduates are able to understand the basics of computer organization after only one or two semesters. They are able to comprehend the complexity not by sheer force of concentration, but rather by understanding the "building blocks" from which computers are made. A similar experience awaits the novice algorithm designer. The algorithms surveyed above deal with many problem areas, but are rather simple to comprehend once we understand the "building blocks" of algorithm design. Here we will describe three important classes of these "blocks."

Data structures. Algorithms deal with data, and the algorithm designer uses *data structures* as tools to organize data. Earlier we saw simple data structures such as arrays and matrices, and a fairly complex data structure, the hash table.

There are more exotic types, such as linked lists, stacks, queues, priority queues, and trees. Each provides an appropriate way to structure data for a particular task. Tarjan[33] briefly describes many of these structures; Knuth[34] provides detailed descriptions of a large number of interesting structures.

Algorithmic techniques. Structured programming demands that a programmer express a complicated sequence of commands as a series of refinements by which the program can be understood at different levels. In each of these refinements the programmer applies a basic, well understood method to a well defined problem. Good programmers used this technique long before it was verbalized; good algorithm designers use a similar strategy even though they infrequently discuss it. The constructs available to the algorithm designer are similar to those available in structured programming, though somewhat more powerful. We will describe some of these constructs very briefly; more detail can be found in Tarjan[35] and Aho, Hopcroft, and Ullman.[36]

We have already seen many common algorithmic constructs. Most of the algorithms we described use *iteration* in one form or another—this says "do x over and over until the task is accomplished." Iteration is present in almost all programming languages as DO and WHILE loops. A more powerful construct is *recursion*, giving us a way to express recursive problem solving in programming languages. To define a recursive solution to a problem we say (essentially) "to solve a problem of a certain size, solve the same problem of a smaller size." We used recursion to describe a binary search: to binary search a table of size n we binary searched a table of size $n/2$. A particular application of recursion, *divide-and-conquer*, says "to solve a problem of size n, first divide it into subproblems each of size only a fraction of n, then solve those subproblems recursively, and finally combine the subsolutions to yield a solution to the original problem." We can view Mergesort as a textbook example of divide-and-conquer: to sort a list of n elements we first break the list into two sublists each of $n/2$ elements, then sort those recursively, and finally merge those together. The FFT and the $O(n^{2.81})$ matrix multiplication algorithms are other applications of the divide-and-conquer technique. Once we understand the fundamental principles of divide-and-conquer algorithms, we can easily grasp each of these instances.

Researchers have identified and studied many other algorithmic techniques. *Dynamic programming*, a technique from operations research, has found many applications in algorithm design. Search strategies such as *breadth-first search* and *depth-first search* have yielded efficient graph algorithms. *Transformation* allows us to turn an instance of one problem into another; we saw earlier that there are many transformations to turn almost totally unrelated problems into instances of matrix multiplication. Perhaps the single most important algorithmic tecnhique is the use of optimal tools to solve the subproblems we create for ourselves in designing a new algorithm.

Proof techniques. Once an algorithm designer has devised an algorithm and "knows in his heart" that it has certain properties, he must prove that it does. (Perhaps this separates practitioner from theorist.) He must first prove that his algorithm indeed computes what it purports to; he will use many of the tools of program verification here. Next he must analyze his algorithm's resource requirements, using many different mathematical tools. Finally he must prove his algorithm optimal by giving a lower bound proof. Weide[37] discusses the different methods of analysis used in these steps.

Current directions

Algorithm design has grown rapidly in the past decade. Essentially unknown as a field ten years ago, it is not one of theoretical computer science's most active areas and sees widespread use in applications. Continued emphasis on performance ensures even more algorithm research. Here we will examine some of the directions in which the field is moving.

One constant trend has been from "toy" problems to "real" problems. This involves many detailed analyses and expected-resource analyses, for in applications we are often seriously concerned about 20-percent differences in average running time. Much recent work has centered on approximation algorithms, since many applications do not require exact answers. On the theoretical side, the outstanding question is the complexity of the NP-complete problems—are they exponential or not? Another important theoretical problem is the search for some underlying theory of algorithm design; we still have no theoretical explanation for what makes a class of problems easy or hard. Tarjan has mentioned the need for a "calculus of data structures"—a set of rules leading to the best possible structure for a given situation.

An outgrowth of this work will be the development of "algorithm engineering." This field will supply the programmer with tools similar to those electrical engineering gives the circuit designer. Before algorithm design turns into algorithm engineering we will need to develop many more specific algorithms and provide a theoretical base for the field. We will know that the field has become an engineering discipline as soon as theoretical computer scientists assert that designing algorithms is no longer bona fide research because "it's such a well-understood process."

Algorithm design offers much to individuals involved in computing-related activities. The mathematician and theoretical computer scientist can view the field as a rich source of problems needing precise mathematical treatment. These problems are mathematically fascinating and call for some of the most powerful tools of discrete mathematics. The applications programmer with little interest in elegant theorems can also benefit, since proper algorithm design occasionally yields significant financial savings. But even placing aside benefits to specific practitioners, I feel that anyone involved with computing, regardless of his position in the practical-to-theoretical continuum, should have some familiarity with the field. The study of algorithms is the study of the very heart of computing, and it provides us with a new way of thinking about our computational problems.

Acknowledgments

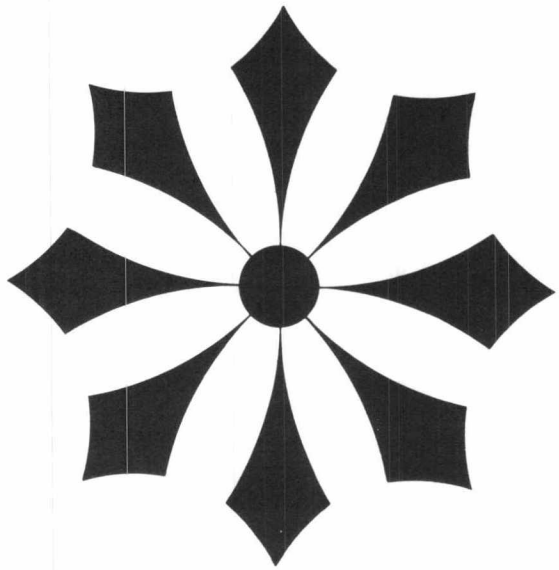
This paper has profited from the helpful criticism of many people; the comments of David Jefferson, Len Shustek, and Bruce Weide have been particularly helpful. The research described here was supported in part by the Office of Naval Research, under Contract N000014-76-C-0370.

References

1. J. E. Hopcroft, "Complexity of Computer Computations," *Proc. IFIP Congress 74,* Vol. 3, 1974, pp. 620-626.
2. R. E. Tarjan, "Complexity of Combinatorial Algorithms," *SIAM Review,* Vol. 20, No. 3, July 1978, pp. 457-491.
3. B. Weide, "A Survey of Analysis Techniques for Discrete Algorithms," *Computing Surveys,* Vol. 9, No. 4, Dec. 1977, pp. 291-313.
4. D. E. Knuth, "Mathematical Analysis of Algorithms," *IFIP Congress 71,* Vol. 1, 1971, pp. 135-143.
5. D. E. Knuth, "Algorithms," *Scientific American,* Vol. 236, No. 4, Apr. 1977, pp. 63-80.
6. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, Mass., 1974.
7. D. E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms,* Addison-Wesley, Reading, Mass., 1968.
8. D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms,* Addison-Wesley, Reading, Mass., 1969.
9. D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching,* Addison-Wesley, Reading, Mass., 1973.
10. Knuth, *Art of Computer Programming, Volume 3,* p. 391.
11. Knuth, *Scientific American,* Vol. 236, No. 4, Apr. 1977, p. 63ff.
12. D. E. Knuth, J. H. Morris, and V. R. Pratt, "Fast Pattern Matching in Strings," *SIAM Journal of Computing,* Vol. 6, No. 2, June 1977, pp. 323-350.
13. R. S. Boyer and J. S. Moore, "A Fast String Searching Algorithm." *CACM,* Vol. 20, No. 10, Oct. 1977, pp. 762-772.
14. J. M. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation,* Vol. 19, 1965, pp. 297-301.
15. Aho, Hopcroft, and Ullman, *Design and Analysis...,* pp. 252-276.
16. A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems,* Elsevier-North Holland Pub. Co., N.Y., Theory of Computation Series No. 1, 1975.
17. D. R. Brillinger, *Time Series: Data Analysis and Theory,* Holt, Rinehart, and Winston, N.Y., 1975.
18. V. Strassen, "Gaussian Elimination is Not Optional," *Numerishe Mathematik,* Vol. 13, 1969, pp. 345-356.
19. W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Trans. Information Theory,* Vol. IT-22, No. 6, Nov. 1976, pp. 644-654.
20. R. L. Rivest, A. Shamir, and L. Adleman, "A Mathod for Obtaining Digital Signatures and Public-Key Cryptosystems," *CACM,* Vol. 21, No. 3, Feb. 1978, pp. 120-126.
21. M. Gardner, "A New Kind of Cipher That Would Take Millions of Years to Break" (Mathematical Games), *Scientific American,* Vol. 236, No. 8, Aug. 1977, pp. 120-125.
22. Knuth, *Art of Computer Programming, Volume 1,* pp. 120-160.
23. Aho, Hopcroft, and Ullman, *Design and Analysis...,* pp. 2-41.
24. *Ibid.,* pp. 24-25.
25. H. T. Kung, "Synchronized and Asynchronous Parallel Algorithms for Multiprocessors," in *Algorithms and Complexity: New Directions and Recent Results,* J. F. Traub, Ed., Academic Press, N.Y., 1976, pp. 153-200.
26. H. R. Lewis and C. H. Papidimitriou, "The Efficiency of Algorithms," *Scientific American,* Vol. 238, No. 1, Jan. 1978, pp. 97-109.
27. M. R. Garey and D. S. Johnson, "Approximation Algorithms for Combinatorial Problems: An Annotated Bibliography," in *Algorithms and Complexity: New Directions and Recent Results,* J. F. Traub, ed., Academic Press, N.Y., 1976, pp. 41-52.
28. Knuth, *Art of Computer Programming, Volume 3.*
29. M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. Tarjan, "Time Bounds for Selection," *Journal of Computer and Systems Sciences,* Vol. 7, No. 4, Aug. 1973, pp. 448-461.
30. Tarjan, *SIAM Review,* Vol. 20, No. 3.
31. J. Hopcroft and R. Tarjan, "Efficient Planarity Testing," *JACM,* Vol. 21, No. 4, Oct. 1974, pp. 549-568.
32. M. I. Shamos, "Geometric Complexity," *Proc. Seventh ACM Symposium on the Theory of Computing,* 1975, pp. 224-233.
33. Tarjan, *SIAM Review,* Vol. 20, No. 3.
34. Knuth, *Art of Computer Programming, Volume 1.*
35. Tarjan, *SIAM Review,* Vol. 20, No. 3.
36. Aho, Hopcroft, and Ullman, *Design and Analysis...,* pp. 44-74.
37. Weide, *Computing Surveys,* Vol. 9, No. 4, pp. 291-313.

Jon Louis Bentley is an assistant professor of computer science and mathematics at Carnegie-Mellon University, Pittsburgh, Pennsylvania. He worked as a research intern at the Xerox Palo Alto Research Center from 1973 to 1974. During the summer of 1975 he was a visiting scholar at the Stanford Linear Accelerator Center. From 1975 to 1976 he was a National Science Foundation graduate fellow. His current research interests include the design and analysis of computer algorithms (especially for geometrical and statistical problems) and the mathematical foundations of computation.

A member of the ACM and Sigma Xi, Bentley received the BS in mathematical sciences from Stanford University in 1974, and the MS and PhD in computer science from the University of North Carolina in 1976.



ALL THE SECRETS YOU EVER WANTED ABOUT POLY 88 BASIC

BY MARVIN A. KONOPIK
American Embassy
APO San Francisco 96503

I have broken some of the secrets of the Poly 88 BASIC version A00 that might be helpful to others in understanding their BASIC as well, or at least help to get a handle on it. The Poly BASIC Manual contains a great deal of technical information, but the many ways this information can be used is not fully explained. After retyping a program for the third time, I threw in the towel, and decided there must be an easier way. So, I have listed below some short-cuts that make computer life a whole new game and a lot more fun.

SCR is a dirty word: type SCR or LOAD and all is lost! Wrong . . . The key to the problem is in location 205D and 205E. The address listed here tells you where BASIC started loading your program. Go to that location, and scroll ahead. You will see several rows of zeros, which is what SCR is all about. Scroll ahead further and look through the memory until you see an 0D. This is the Carriage Return in Hex. Remember the memory location of the 0D. Put this location into position 205D, type in the warm start SPJ2003 and a G and your program is up. Restore your program, record it onto tape, and after you verify it, reload your BASIC and continue. I have never reloaded my BASIC; it seems to operate perfectly with the new starting point, but why take a chance. You lost the first couple of lines, but this is no big deal. Next time begin your programs with a full REM line of 63 spaces, to minimize your losses in case of an accident.

If that was easy, then try the following hard way for greater program salvage. Using the solution above, you have lost some valuable program material between the end of the zeros and the first 0D. This may have been valuable text that SCR or LOAD did not destroy, but which *you* did. If you want to restore your program and minimize the losses, call up location 205D to see where your program starts. Go to the location shown in 205D and you should see 0D and several lines of zeros. Hit the spacebar to go to the next location after the 0D. There is probably an 01 already in that box, but load 2C into it. 2C is Hex for 44. Now back up until the 0D is in the memory window again. Now hit the spacebar 44 times and if the memory is still full of zeros, type in an 0D. Use control H or LF back to the location following the 2C. You can use any number instead of 2C, except that 44 should be near the last of the zeros and it seems to work about right. Put in a 01 00 8F and fill the remaining locations with 20 until you reach the 0D that you just loaded. As you may have guessed, the first 2C was the length of the line from 0D to 0D. The 01 00 was the wasy BASIC stores line number 0001, the 8F is what BASIC calls REM when it stores it into memory, and the 20s are all spaces in Hex. You have just typed in a REM line of 40 or so spaces in BASIC, the hard way. Now for line 0002. In the next block after the OD of line 0001 that you just typed, put in FF so you can find it easily later. Back up again to the 0D before the FF and count the number of spacebar hits until the next 0D appears. Write this number down. If this number is less than 64, then continue, if not, read on, I'll help you later. Your number, which is less than 64, should be converted to HEX and put in place of the FF you previously loaded as the start of line 0002. The next locations will be 02 00 8F and the remaining zeros should be filled with 20's as you did in line one. If you had over 64 in your count, then re-do the procedure above for line 0001 using 40 (64 decimal) or some number in between, to lengthen line 0001 until the OD is closer to the end of the zeros and the beginning of the program to be saved. Another way is to make a very short line 0002 and use 0003 for the first partial line to be salvaged. The drawback is that 3 could be too high if only line 01 was lost, and you had line 2 twice. Another way out is to renumber the next line since you now know how they are stored in memory by the BASIC program. Now that you have filled all the zeros with something, type in your warm start location, and your program should be there with the first partial line of your program included on the end of a REM line, but at least it is not lost. If it did not come up, you simply miscounted your line length, so go back and count again. If it still will not come up, look ahead in memory and practice counting the lines there between 0D's until you have the system down pat and then try again.

Merging programs is a complicated procedure! Not Really. Before you load BASIC, Hit Cmnd Z, and load the following machine program into memory starting at location 2000. 97 21 09 20 77 23 C3 04 20 and then SPJ2000 and a G. When the screen flashes bright white, all memory has been set to zeros. Now load in BASIC and your first program. Type REN 1,1 and LIST your program. Remember to copy the last line number on a piece of paper so you don't forget it. Hit Cmnd Z and scroll through the memory a hundred lines or so at a time, starting at 4A00 until you find the zeros. Back up until you find the last 0D of the program, before the zeros start. Write down this memory location, and call up location 205D. Also remember the numbers that are in 205D: you will need them later. Load the location of the last 0D into 205D and 205E. Type in SPJ2000 (cold start) and a G. Load the next program just like you did the first. Type LIST to see if the second program is there. RENumber it, but use a number higher than the last line of the previous program. Do a Cmnd Z again, and change location 205D back to what it originally was, and Warm Start your program using 2003. Hit LIST, and behold! Two programs in a row. Mark down the new last line number, go back to the memory to check for the last 0D again, change location 205D to this location, cold start 2000, load the third program, return 205D to its original state, warm start on 2003 and you're up with 3 programs in a row.

You bought a printer and now you found out that you cannot warm start and save either the program or the printer/driver! It's simple: change the data at location 4A4D from 00 20 to 03 20 and warm start using 49C0. You can re-record BASIC this way, but dump from locations 2000 through at least 4C02 just to be safe. BASIC will now always come on warm, but you may occasionally have to do a SCR before you can load your programs. If you zeroize memory before you re-record BASIC, it will not load. You may have to load a small program to put some text in the buffer first, but it's worth the effort to have 2003 in location 4A4D when you need it for a warm start and still retain the use of the Printer/Driver.

Can't warm start your Assembler? I cannot find this in the manual. I guess the Poly people thought everybody should go around retyping lost programs even if it took days. Well I did my share of retyping and then discovered that when you type MNTR you can return to the Editor by hitting a G. For this, Poly puts the address 20FF in the program counter. So if Poly can do that, you can too, and warm start any time you have a lock up by using 20FF.

Have you ever wanted to exchange text in the Editor, or switch the end of your Assembler program to the beginning and the beginning to the end? Rather than retype it, let the Editor help you. Dump the text on tape and make a note of the length of the dump as well as the reading of the index counter on the recorder. For instance, we want the last half first and the first half last. KILL the old text from memory, and back the recorder up half way. Now load from that point onward to the end. When you hit the end, rewind to the beginning and start loading again until you feel you have loaded all the text on tape somewhere at least one time. Delete duplicate text and tidy it up for use. I am not a very fast typist, but watch me go with the delete and Cmnd X line remover. Using this procedure, you can load bits and pieces of different programs together into one blend.

So much for the serious part of life! Do you want some fun? Type 'flip' in small letters while you are up in BASIC. It reverses the keyboard. You, who have to use this, probably don't think it's funny, but it makes the keyboard the darndest thing you have ever tried to type with.

The data stored at location 205D shows where BASIC begins storing your program in memory. I hope somebody can find the location where BASIC stores the last 0D (CR) of the program, if there is a location.

And so as you see, the discoveries go on, and on, and on, and each time, life becomes a little easier with Poly.

META II

A SYNTAX-ORIENTED COMPILER WRITING LANGUAGE

BY D. V. SCHORRE
UCLA Computing Facility
University of California at Los Angeles
Los Angeles, California

© 1964, Association for Computing Machinery, Inc.
Reprinted by Permission

Introduction

This classic paper describes a metacompiler developed as a project of the Los Angeles Chapter of SIGPLAN, the Association for Computing Machinery's special interest group in programming languages. Several artifacts of the time appear – the use of "., " rather than ";" and the use of "/" rather than "|" is due to the limited character set on the IBM 026 keypunch. The original Meta II ran on the IBM 1401, a machine with limited resources by today's standards. We've reprinted it here because it is still a good tool for experimenting with languages and their implementation. To install a Meta II system on your computer, you must hand-compile the Meta II compiler given in the article (an instructive exercise in itself) and write an interpreter for the Meta II machine in some suitable language (Microsoft BASIC wouldn't be a bad choice). To check the whole thing, the compiler should compile itself exactly. If not, fix the bug(s) and try again. In the end, you'll have a nice tool for experimenting with programming languages and learn a bit about the compiling process to boot. – *Dennis R. Allison*

META II is a compiler writing language which consists of syntax equations resembling Backus normal form and into which instructions to output assembly language commands are inserted. Compilers have been written in this language for VALGOL I and VALGOL II. The former is a simple algebraic language designed for the purpose of illustrating META II. The latter contains a fairly large subset of ALGOL 60.

The method of writing compilers which is given in detail in the paper may be explained briefly as follows. Each syntax equation is translated into a recursive subroutine which tests the input string for a particular phrase structure, and deletes it if found. Backup is avoided by the extensive use of factoring in the syntax equations. For each source language, an interpreter is written and programs are compiled into that interpretive language.

META II is not intended as a standard language which everyone will use to write compilers. Rather, it is an example of a simple working language which can give one a good start in designing a compiler-writing compiler suited to his own needs. Indeed the META II compiler is written in its own language, thus lending itself to modification.

Reprinted from the Proceedings of the 19th National Conference of the Association for Computing Machinery, 1964.

History

The basic ideas behind META II were described in a series of three papers by Schmidt,[1] Metcalf,[2] and Schorre.[3] These papers were presented at the 1963 National A.C.M. Convention in Denver, and represented the activity of the Working Group on Syntax-Directed Compilers of the Los Angeles SIGPLAN. The methods used by that group are similar to those of Glennie and Conway, but differ in one inportant respect. Both of these researchers expressed syntax in the form of diagrams, which they subsequently coded for use on a computer. In the case of META II, the syntax is input to the computer in a notation resembling Backus normal form. The method of syntax analysis discussed in this paper is entirely different from the one used by Irons[6] and Bastian.[7] All of these methods can be traced back to the mathematical study of natural languages, as described by Chomsky.[8]

Syntax Notation

The notation used here is similar to the meta language of the ALGOL 60 report. Probably the main difference is that this notation can be keypunched. *Symbols* in the target language are represented as strings of characters, surrounded by quotes. *Metalinguistic variables* have the same form as identifiers in ALGOL, viz., a letter followed by a sequence of letters or digits. Items are written consecutively to indicate *concatenation* and separated by a slash to indicate alternation. Each equation ends with a semicolon which, due to keypunch limitation, is represented by a period followed by a comma. An example of a syntax equation is:

```
LOGICALVALUE = '.TRUE' / '.FALSE' .,
```

In the versions of ALGOL described in this paper the symbols which are usually printed in boldface type will begin with periods, for example:

```
.PROCEDURE  .TRUE  .IF
```

To indicate that a syntactic element is *optional,* it may be put in alternation with the word .EMPTY. For example:

```
SUBSECONDARY = '*' PRIMARY / .EMPTY .,
SECONDARY = PRIMARY SUBSECONDARY .,
```

By factoring, these two equations can be written as a single equation.

```
SECONDARY = PRIMARY ( ' * ' PRIMARY / . EMPTY) . ,
```

Built into the META II language is the ability to recognize three symbols which are:

1. Identifiers – represented by . ID,
2. Strings – represented by . STRING,
3. Numbers – represented by . NUMBER.

The definition of identifier is the same in META II as in ALGOL, viz., a letter followed by a sequence of letters or digits. The definition of a string is changed because of the limited character set available on the usual keypunch. In ALGOL, strings are surrounded by opening and closing quotation marks, making it possible to have quotes within a string. The single quotation mark on the keypunch is unique, imposing the restriction that a string in quotes can contain no other quotation marks.

The definition of number has been radically changed. The reason for this is to cut down on the space required by the machine subroutine which recognizes numbers. A number is considered to be a string of digits which may include imbedded periods, but may not begin or end with a period; moreover, periods may not be adjacent. The use of the subscript 10 has been eliminated.

Now we have enough of the syntax defining features of the META II language so that we can consider a simple example in some detail.

The example given here is a set of four syntax equations for defining a very limited class of algebraic expressions. The two operators, addition and multiplication, will be represented by + and * respectively. Multiplication takes precedence over addition; otherwise precedence is indicated by parentheses. Some examples are:

```
A
A + B
A + B * C
(A + B) * C
```

The syntax equations which define this class of expressions are as follows:

```
EX3 = . ID / ' ( ' EX1 ' ) ' . ,
EX2 = EX3 ( ' * ' EX2 / . EMPTY ) . ,
EX1 = EX2 ( ' + ' EX1 / . EMPTY ) . ,
```

EX is an abbreviation for expression. The last equation, which defines an expression of order 1, is considered the main equation. The equations are read in this manner. An expression of order 3 is defined as an identifier or an open parenthesis followed by an expression of order 1 followed by a closed parenthesis. An expression of order 2 is defined as an expression of order 3, which may be followed by a star which is followed by an expression of order 2. An expression of order 1 is defined as an expression of order 2, which may be followed by a plus which is followed by an expression of order 1.

Although sequences can be defined recursively, it is more convenient and efficient to have a special operator for this purpose. For example, we can define a sequence of the letter A as follows:

```
SEQA = $ ' A ' . ,
```

The equations given previously are rewritten using the sequence operator as follows:

```
EX3 = . ID / ' ( ' EX1 ' ) ' . ,
EX2 = EX3 $ ( ' * ' EX3 ) . ,
EX1 = EX2 $ ( ' + ' EX2 ) . ,
```

Output

Up to this point we have considered the notation in META II which describes object language syntax. To produce a compiler, output commands are inserted into the syntax equations. Output from a compiler written in META II is always in an assembly language, but not in the assembly language for the 1401. It is for an interpreter, such as the interpreter I call the META II machine, which is used for all compilers, or the interpreters I call the VALGOL I and VALGOL II machines, which obviously are used with their respective source languages. Each machine requires its own assembler, but the main difference between the assemblers is the operation code table. Constant codes and declarations may also be different. These assemblers all have the same format, which is shown below.

LABEL		CODE		ADDRESS	
1–	– 6	8–	– 10	12–	– 70

An assembly language record contains either a label or an op code of up to 3 characters, but never both. A label begins in column 1 and may extend as far as column 70. If a record contains an op code, then column 1 must be blank. Thus labels may be any length and are not attached to instructions, but occur between instructions.

To produce output beginning in the op code field, we write .OUT and then surround the information to be reproduced with parentheses. A string is used for literal output and an asterisk to output the special symbol just found in the input. This is illustrated as follows:

```
EX3 = . ID . OUT ( ' LD ' * ) / ' ( ' EX1 ' ) ' . ,
EX2 = EX3 $ ( ' * ' EX3 . OUT ( ' MLT ' ) ) . ,
EX1 = EX2 $ ( ' + ' EX2 . OUT ( ' ADD ' ) ) . ,
```

To cause output in the label field we write .LABEL followed by the item to be output. For example, if we want to test for an identifier and output it in the label field we write:

```
. ID . LABEL *
```

The META II compiler can generate labels of the form A01, A02, A03, . . . A99, B01, ... To cause such a label to be generated, one uses *1 or *2. The first time *1 is referred to in any syntax equation, a label will be generated and assigned to it. This same label is output whenever *1 is referred to within that execution of the equation. The symbol *2 works in the same way. Thus a maximum of two different labels may be generated for each execution of any equation. Repeated executions, whether recursive or externally initiated, result in a continued sequence of generated labels. Thus all syntax equations contribute to the one sequence. A typical example in which labels are generated for branch commands is now given.

```
IFSTATEMENT = ' . IF ' EXP ' . THEN ' . OUT ( ' BFP ' *1 )
      STATEMENT ' . ELSE ' . OUT ( ' B ' *2 ) . LABEL *1
      STATEMENT . LABEL *2 . ,
```

The opcodes BFP and B are orders of the VALGOL I machine, and stand for "branch false and pop" and "branch" respectively. The equation also contains references to two other equations which are not explicitly given, viz., EXP and STATEMENT.

VALGOL I – A Simple Compiler Written in META II

Now we are ready for an example of a compiler written in META II. VALGOL I is an extremely simple language, based on ALGOL 60, which has been designed to illustrate the META II compiler.

The basic information about VALGOL I is given in figure 1 (the VALGOL I compiler written in META II) and figure 2 (order list of the VALGOL I machine). A sample program is given in figure 3. After each line of the program, the VALGOL I commands which the compiler produces from that line are shown, as well as the absolute

interpretive language produced by the assembler. Figure 4 is output from the sample program. Let us study the compiler written in META II (figure 1) in more detail.

The identifier PROGRAM on the first line indicates that this is the main equation, and that control goes there first. The equation for PRIMARY is similar to that of EX3 in our previous example, but here numbers are recognized and reproduced with a "load literal" command. TERM is what was previously EX2; and EXP1 what was previously EX1 except for recognizing minus for subtraction. The equation EXP defines the relational operator "equal," which produces a value of 0 or 1 by making a comparison. Notice that this is handled just like the arithmetic operators but with a lower precedence. The conditional branch commands, "branch true and pop" and "branch false and pop," which are produced by the equations defining UNTILST and CONDITIONALST respectively, will test the top item in the stack and branch accordingly.

The "assignment statement" defined by the equation for ASSIGNST is reversed from the convention in ALGOL 60, i.e., the location into which the computed value is to be stored is on the right. Notice also that the equal sign is used for the assignment statement and that period equal (.=) is used for the relation discussed above. This is because assignment statements are more numerous in typical programs than equal compares, and so the simpler representation is chosen for the more frequently occurring.

The omission of statement labels from the VALGOL I and VALGOL II seems strange to most programmers. This was not done because of any difficulty in their implementation, but because of a dislike for statement labels on the part of the author. I have programmed for several years without using a single label, so I know that they are superfluous from a practical, as well as from a theoretical, standpoint. Nevertheless, it would be too much of a digression to try to justify this point here. The "until statement" has been added to facilitate writing loops without labels.

The "conditional" statement is similar to the one in ALGOL 60, but here the "else" clause is required.

The equation for "input/output," IOST, involves two commands, "edit" and "print." The words EDIT and PRINT do not begin with periods so that they will look like subroutines written in code. "EDIT" copies the given string into the print area, with the first character in the print position whish is computed from the given expression. "PRINT" will print the current contents of the print area and then clear it to blanks. Giving a print command without previous edit commands results in writing a blank line.

IDSEQ1 and IDSEQ are given to simplify the syntax equation for DEC (declaration). Notice in the definition of DEC that a branch is given around the data.

From the definition of BLOCK it can be seen that what is considered a compound statement in ALGOL 60 is, in VALGOL I, a special case of a block which has no declaration.

In the definition of statement, the test for an IOST precedes that for an ASSIGNST. This is necessary, because if this were not done the words PRINT and EDIT would be mistaken as identifiers and the compiler would try to translate "input/output" statements as if they were "assignment" statements.

Notice that a PROGRAM is a block and that a standard set of commands is output after each program. The "halt" command causes the machine to stop on reaching the end of the outermost block, which is the program. The operation code SP is generated after the "halt" command. This is a completely 1401-oriented code, which serves to set a word mark at the end of the program. It would not be used if VALGOL I were implemented on a fixed word-length machine.

How the META II Compiler Was Written

Now we come to the most interesting part of this project, and consider how the META II compiler was written in its own language. The interpreter called the META II machine is not a much longer 1401 program than the VALGOL I machine. The syntax equations for META II (figure 5) are fewer in number than those for the VALGOL I machine (figure 1).

The META II compiler, which is an interpretive program from the META II Machine, takes the syntax equations given in figure 5 and produces an assembly language version of this same interpretive program. Of course, to get this started, I had to write the first compiler-writing compiler by hand. After the program was running, it could produce the same program as written by hand. Someone always asks if the compiler really produced exactly the program I had written by hand and I have to say that it was "almost" the same program. I followed the syntax equations and tried to write just what the compiler was going to produce. Unfortunately I forgot one of the redundant instructions, so the results were not quite the same. Of course, when the first machine-produced compiler compiled itself the second time, it reproduced itself exactly.

The compiler originally written by hand was for a language called META I. This was used to implement the improved compiler for META II. Sometimes, when I wanted to change the metalanguage, I could not describe the new metalanguage directly in the current metalanguage. Then an intermediate language was created—one which could be described in the current language and in which the new language could be described. I thought that it might sometimes be necessary to modify the assembly language output, but it seems that it is always possible to avoid this with the intermediate language.

The order list of the META II machine is given in figure 6.

All subroutines in META II programs are recursive. When the program enters a subroutine a stack is pushed down by three cells. One cell is for the exit address and the other two are for labels which may be generated during the execution of the subroutine. There is a switch which may be set or reset by the instructions which refer to the input string, and this is the switch referred to by the conditional branch commands.

The first thing in any META II machine program is the address of the first instruction. During the initialization for the interpreter, this address is placed into the instruction counter.

VALGOL II Written in META II

VALGOL II is an expansion of VALGOL I, and serves as an illustration of a fairly elaborate programming language implemented in the META II system. There are several features in the VALGOL II machine which were not present in the VALGOL I machine, and which require some explanation. In the VALGOL II Machine, addresses as well as numbers are put in the stack. They are marked appropriately so that they can be distinguished at execution time.

The main reason that addresses are allowed in the stack is that, in the case of a subscripted variable, an address is the result of a computation. In an assignment statement each left member is compiled into a sequence of code which leaves an address on top of the stack. This is done for simple variables as well as subscripted variables, because the philosophy of this compiler writing system has been to compile everything in the most general way. A variable, simple or subcripted, is always compiled into a sequence of instructions which leaves an address on top of the stack. The address is not replaced by its contents until the actual value of the variable is needed, as in an arithmetic expression.

A formal parameter of a procedure is stored either as an address or as a value which is computed when the procedure is called. It is up to the load command to go through any number of indirect address in order to place the address of a number onto the stack. An argument of a procedure is always an algebraic expression. In case this expression is a variable, the value of the formal parameter will be an address computed upon entering the procedure.

The operation of the load command is now described. It causes the given address to be put on top of the stack. If the content of this top item happens to be another address, then it is replaced by that other address. This continues until the top item on the stack is the address of something which is not an address. This allows for formal parameters to refer to other formal parameters to any depth.

No distinction is made between integer and real numbers. An integer is just a real number whose digits right of the decimal point are zero. Variables initially have a value called "undefined," and any attempt to use this value will be indicated as an error.

An assignment statement consists of any number of left parts followed by a right part. For each left part there is compiled a sequence of commands which puts an address on top of the stack. The right part is compiled into a sequence of instructions which leaves on top of the stack either a number or the address of a number. Following the instruction for the right part there is a sequence of store commands, one for each left part. The first command of this sequence is "save and store," and the rest are "plain" store commands. The "save and store" puts the number which is on the top of the stack (or which is referred to by the address on top of the stack) into a register called SAVE. It then stores the contents of SAVE in the address which is held in the next to top position of the stack. Finally it pops the top two items, which it has used, out of the stack. The number, however, remains in SAVE for use by the following store commands. Most assignment statements have only one left part, so "plain" store commands are seldom produced, with the result that the number put in SAVE is seldom used again.

The method for calling a procedure can be explained by reference to illustrations 1 and 2. The arguments which are in the stack are moved to their place at the top of the procedure. If the number of arguments in the stack does not correspond to the number of arguments in the procedure, an error is indicated. The "flag" in the stack works like this. In the VALGOL II machine there is a flag register. To set a flag in the stack, the contents of this register is put on top of the stack, then the address of the word above the top of the stack is put into the flag register. Initially, and whenever there are no flags in the stack, the flag register contains blanks. At other times it contains the address of the word in the stack which is just above the uppermost flag. Just before a call instruction is executed, the flag register contains the address of the word in the stack which is two above the word containing the address of the procedure to be executed. The call instruction picks up the arguments from the stack, beginning with the one stored just above the flag, and continuing to the top of the stack. Arguments are moved into the appropriate places at the top of the procedure being called. An error message is given if the number of arguments in the stack does not correspond to the number of places in the procedure. Finally the old flag address, which is just below the procedure address in the stack, is put in the flag register. The exit address replaces the address of the procedure in the stack, and all the arguments, as well as the flag, are popped out. There are just two op codes which affect the flag register. The code "load flag" puts a flag into the stack, and the code "call" takes one out.

XXXXXXXX	Function
XXXXXXXX XXXXXXXX XXXXXXXX	Arguments
b	Word of one blank character to mark the end of the arguments.
. . . .	Body. Branch commands cause control to go around data stored in this area. Ends with a "return" command.

Illustration 1

Storage Map for VALGOL II Procedures

XXXXXXXX XXXXXXXX XXXXXXXX	Arguments in reverse order		
XXX	Flag		
XXX	Address of procedure	Exit	XXX
. . . .			. . . .
Stack before executing the call instruction			Stack after executing the call instruction

Illustration 2

Map of the Stack Relating to Procedure Calls

The library function "WHOLE" truncates a real number. It does not convert a real number to an integer, because no distinction is made between them. It is substituted for the recommended function "ENTIER" primarily because truncation takes fewer machine instructions to implement. Also, truncation seems to be used more frequently. The procedure ENTIER can be defined in VALGOL II as follows:

```
.PROCEDURE ENTIER(X) .,
   .IF 0 .L = X .THEN X .ELSE
   .IF WHOLE(X) = X .THEN X .ELSE
   WHOLE(X) -1
```

The "for statement" in VALGOL II is not the same as it is in ALGOL. Exactly one list element is required. The "step . . . until" portion of the element is mandatory, but the "while" portion may be added to terminate the loop immediately upon some condition. The iteration continues so long as the value of the variable is less than or equal to the maximum, irrespective of the sign of the increment. Illustration 3 is an example of a typical "for statement." A flow chart of this statement is given in illustration 4.

```
.FOR I = 0 .STEP 1   .UNTIL N .DO
        [statement]
        SET             Set switch to indicate first time through.
A91
        LD      I
        FLP             } Test for first time through.
        BFP     A92     }
        LDL     0       }
        SST             } Initialize variable.
        B       A93     }
A92
        LDL     1       } Increment variable.
        ADS             }
A93
        RSR             } Compare variable to maximum.
        LD      N       }
        LEQ             }
        BFP     A94     }
        [statement]
        RST             Reset switch to indicate not first time
                        through.
        B       A91
A94
```

Illustration 3
Compilation of a typical "for statement"
in VALGOL II

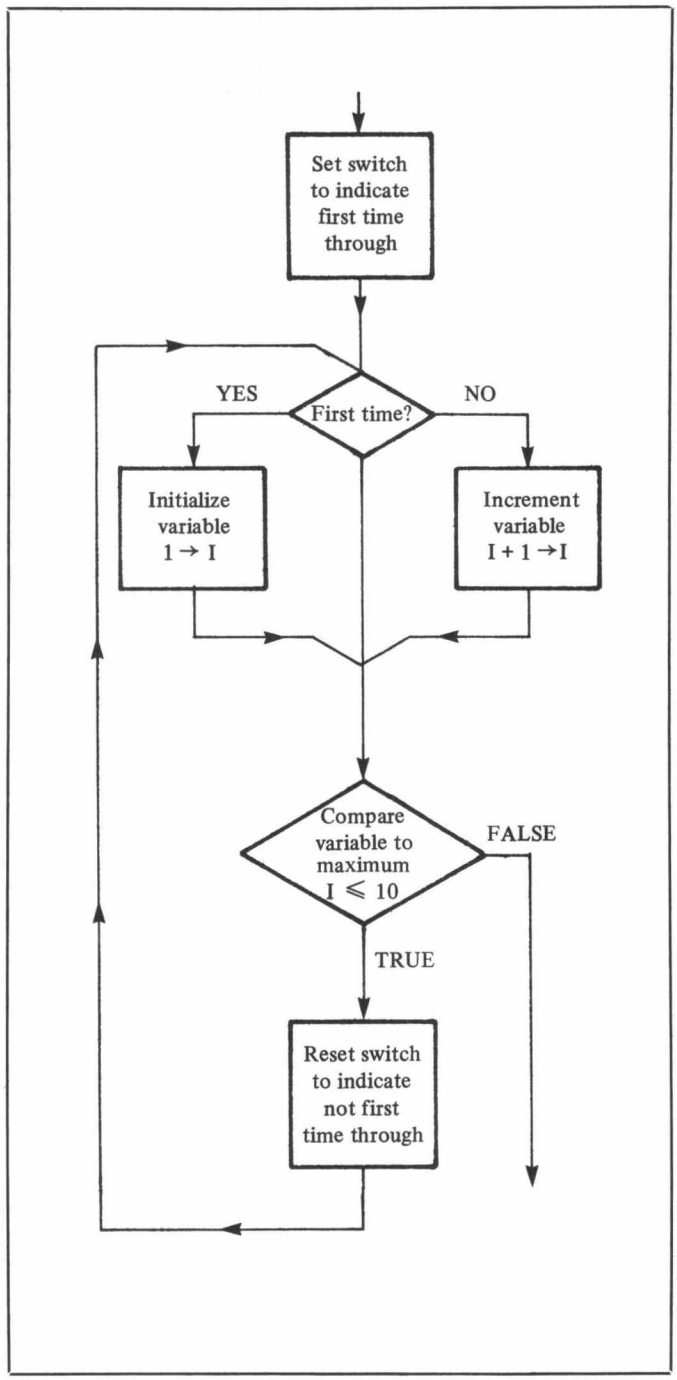


Illustration 4

Flow chart of the "for statement" given in figure 12

Figure 7 is a listing of the VALGOL II compiler written in META II. Figure 8 gives the order list of the VALGOL II machine. A sample program to take a determinant is given in figure 9.

Backup vs. No Backup

Suppose that, upon entry to a recursive subroutine, which represents some syntax equation, the positions of the input and output are saved. When some non-first term of a component is not found, the compiler does not have to stop with an indication of a syntax error. It can back-up the input and output and return false. The advantages of backup are as follows:

1. It is possible to describe languages, using backup, which cannot be described without backup.
2. Even for a language which can be described without backup, the syntax equations can often be simplified when backup is allowed.

The advantages claimed for non-backup are as follows:

1. Syntax analysis is faster.
2. It is possible to tell whether syntax equations will work just by examining them, without following through numerous examples.

The fact that rather sophisticated languages such as ALGOL and COBOL can be implemented without backup is pointed out by various people, including Conway, [5] and they are aware of the speed advantages of so doing. I have seen no mention of the second advantage of no-backup, so I will explain this in more detail.

Basically one writes alternations in which each term begins with a different symbol. Then it is not possible for the compiler to go down the wrong path. This is made more complicated because of the use of ".EMPTY". An optional item can never be followed by something that begins with the same symbol it begins with.

The method described above is not the only way in which backup can be handled. Variations are worth considering, as a way may be found to have the advantages of both backup and no-backup.

Further Development of META Languages

As mentioned earlier, META II is not presented as a standard language, but as a point of departure from which a user may develop his own META language. The term "META Language," with "META" in capital letters, is used to denote any compiler-writing language so developed.

The language which Schmidt[1] implemented on the PDP-1 was based on META I. He has now implemented an improved version of this language for a Beckman machine.

Rutman[9] has implemented LOGIK, a compiler for bit-time simulation, on the 7090. He uses a META Language to compile Boolean expressions into efficient machine code. Schneider and Johnson[10] have implemented META 3 on the IBM 7094, with the goal of producing an ALGOL compiler which generates efficient machine code. They are planning a META langauge which will be suitable for any block structured language. To this compiler-writing language they give the name META 4 (pronounced metaphor).

References

1. Schmidt, L., "Implementation of a Symbol Manipulator for Heuristic Translation," 1963 ACM Natl. Conf., Denver, Colo.
2. Metcalfe, Howard, "A Parameterized Compiler Based on Mechanical Linguistics," 1963 ACM Natl. Conf., Denver, Colo.
3. Schorre, Val, "A Syntax-Directed SMALGOL for the 1401," 1963 ACM Natl. Conf., Denver, Colo.
4. Glennie, A., "On the Syntax Machine and the Construction of a Universal Compiler," Tech. Report No. 2, Contract NR 049-141, Carnegie Inst. of Tech., July, 1960.
5. Conway, Melvin E., "Design of a Separable Transition-Diagram Compiler," Comm. ACM, July 1963.
6. Irons, E.T., The Structure and Use of the Syntax-Directed Compiler," Annual Review in Automatic Programming, The Macmillan Co., New York.
7. Bastian, Lewis, "A Phrase-Structured Language Translator," AFCRL-Rept-62-549, Aug. 1962.
8. Chomsky, Noam, "Syntax Structures," Mouton and Co., Publishers, The Hague, Netherlands.
9. Rutman, Roger, "LOGIK, A Syntax Directed Compiler for Computer Bit-Time Simulation," Master Thesis, UCLA, August 1964.
10. Schneider, F.W., and G.D. Johnson, "A Syntax-Directed Compiler-Writing Compiler to Generate Efficient Code," 1964 ACM Natl. Conf., Philadelphia.

THE VALGOL I COMPILER WRITTEN IN META II LANGUAGE
FIGURE 1

```
.SYNTAX PROGRAM

PRIMARY = .ID .OUT('LD ' *) /
     .NUMBER .OUT('LDL' *) /
     '(' EXP ')' .,

TERM = PRIMARY $('*' PRIMARY .OUT('MLT') ) .,

EXP1 = TERM $('+' TERM .OUT('ADD') /
     '-' TERM .OUT('SUB') ) .,

EXP = EXP1 ( '.=' EXP1 .OUT('EQU') / .EMPTY) .,

ASSIGNST = EXP '=' .ID .OUT('ST ' *) .,

UNTILST = '.UNTIL' .LABEL *1 EXP '.DO' .OUT('BTP' *2)
     ST .OUT('B  ' *1) .LABEL *2 .,

CONDITIONALST = '.IF' EXP '.THEN' .OUT('BFP' *1)
     ST '.ELSE' .OUT('B  ' *2) .LABEL *1
     ST .LABEL *2 .,

IOST = 'EDIT' '(' EXP ',' .STRING
     .OUT('EDT' *) ')' /
     'PRINT' .OUT('PNT') .,

IDSEQ1 = .ID .LABEL * .OUT('BLK 1') .,

IDSEQ = IDSEQ1 $(',' IDSEQ1) .,

DEC = '.REAL' .OUT('B  ' *1) IDSEQ .LABEL *1 .,

BLOCK = '.BEGIN' (DEC '.,' / .EMPTY)
     ST $('.,' ST) '.END' .,

ST = IOST / ASSIGNST / UNTILST /
     CONDITIONALST / BLOCK .,

PROGRAM = BLOCK .OUT('HLT')
     .OUT('SP  1') .OUT('END') .,

.END
```

ORDER LIST OF THE VALGOL I MACHINE
FIGURE 2

MACHINE CODES

LD AAA	LOAD	PUT THE CONTENTS OF THE ADDRESS AAA ON TOP OF THE STACK.
LDL NUMBER	LOAD LITERAL	PUT THE GIVEN NUMBER ON TOP OF THE STACK.
ST AAA	STORE	STORE THE NUMBER WHICH IS ON TOP OF THE STACK INTO THE ADDRESS AAA AND POP UP THE STACK.
ADD	ADD	REPLACE THE TWO NUMBERS WHICH ARE ON TOP OF THE STACK WITH THEIR SUM.
SUB	SUBTRACT	SUBTRACT THE NUMBER WHICH IS ON TOP OF THE STACK FROM THE NUMBER WHICH IS NEXT TO THE TOP. THEN REPLACE THEM BY THIS DIFFERENCE.
MLT	MULTIPLY	REPLACE THE TWO NUMBERS WHICH ARE ON TOP OF THE STACK WITH THEIR PRODUCT.
EQU	EQUAL	COMPARE THE TWO NUMBERS ON TOP OF THE STACK. REPLACE THEM BY THE INTEGER 1, IF THEY ARE EQUAL, OR BY THE INTEGER 0, IF THEY ARE UNEQUAL.
B AAA	BRANCH	BRANCH TO THE ADDRESS AAA.
BFP AAA	BRANCH FALSE AND POP	BRANCH TO THE ADDRESS AAA IF THE TOP TERM IN THE STACK IS THE INTEGER 0. OTHERWISE, CONTINUE IN SEQUENCE. IN EITHER CASE, POP UP THE STACK.
BTP AAA	BRANCH TRUE AND POP	BRANCH TO THE ADDRESS AAA IF THE TOP TERM IN THE STACK IS NOT THE INTEGER 0. OTHERWISE, CONTINUE IN SEQUENCE. IN EITHER CASE, POP UP THE STACK.
EDT STRING	EDIT	ROUND THE NUMBER WHICH IS ON TOP OF THE STACK TO THE NEAREST INTEGER N. MOVE THE GIVEN STRING INTO THE PRINT AREA SO THAT ITS FIRST CHARACTER FALLS ON PRINT POSITION N. IN CASE THIS WOULD CAUSE CHARACTERS TO FALL OUTSIDE THE PRINT AREA, NO MOVEMENT TAKES PLACE.
PNT	PRINT	PRINT A LINE, THEN SPACE AND CLEAR THE PRINT AREA.
HLT	HALT	HALT.

CONSTANT AND CONTROL CODES

SP N	SPACE	N = 1--9. CONSTANT CODE PRODUCING N BLANK SPACES.
BLK NNN	BLOCK	PRODUCES A BLOCK OF NNN EIGHT CHARACTER WORDS.
END	END	DENOTES THE END OF THE PROGRAM.

A PROGRAM AS COMPILED FOR THE VALGOL I MACHINE
FIGURE 3

```
.BEGIN
.REAL X .,  0 = X .,
                B   A01                          0000 G 0012
          X                                      0004
                BLK 001                          0004
          A01                                    0012
                LDL 0                            0012 A
                ST  X                            0021 B 0004
.UNTIL X .= 3 .DO .BEGIN
          A02                                    0025
                LD  X                            0025 D 0004
                LDL 3                            0029 A
                EQU                              0033 F
                BTP A03                          0039 K 0097
     EDIT( X**X * 10 + 1., '*') ., PRINT ., X + 0.1 = X
                LD  X                            0043 D 0004
                LD  X                            0047 D 0004
                MLT                              0051 E
                LDL 10                           0052 A
                MLT                              0061 E
                LDL 1                            0062 A
                ADD                              0071 C
                EDT 01'*'                        0072 I
                PNT                              0074 O
                LD  X                            0075 D 0004
                LDL 0.1                          0079 A
                ADD                              0083 C
                ST  X                            0089 B 0004
          .END
                B   A02                          0093 G 0025
           A03                                   0097
.END
                HLT                              0097 J
                SP  1                            0098
                END                              0099
```

OUTPUT FROM THE VALGOL I PROGRAM GIVEN IN FIGURE 3
FIGURE 4

THE META II COMPILER WRITTEN IN ITS OWN LANGUAGE
FIGURE 5

```
.SYNTAX PROGRAM

OUT1 = '*1' .OUT('GN1') / '*2' .OUT('GN2') /
'*' .OUT('CI') / .STRING .OUT('CL ' *) .,

OUTPUT = ('.OUT' '('
$ OUT1 ',' / '.LABEL' .OUT('LB') OUT1) .OUT('OUT') .,

EX3 = .ID .OUT ('CLL' *) / .STRING
.OUT('TST' *) / '.ID' .OUT('ID') /
'.NUMBER' .OUT('NUM') /
'.STRING' .OUT('SR') / '(' EX1 ')' /
'.EMPTY' .OUT('SET') /
'$' .LABEL *1 EX3
.OUT ('BT ' *1) .OUT('SET') .,

EX2 = (EX3 .OUT('BF ' *1) / OUTPUT)
$(EX3 .OUT('BE') / OUTPUT)
.LABEL *1 .,

EX1 = EX2 $('/' .OUT('BT ' *1) EX2 )
.LABEL *1 .,

ST = .ID .LABEL * '=' EX1
'.,' .OUT('R') .,

PROGRAM = '.SYNTAX' .ID .OUT('ADR' *)
$ ST '.END' .OUT('END') .,

.END
```

ORDER LIST OF THE META II MACHINE
FIGURE 6

MACHINE CODES

TST STRING	TEST	AFTER DELETING INITIAL BLANKS IN THE INPUT STRING, COMPARE IT TO THE STRING GIVEN AS ARGUMENT. IF THE COMPARISON IS MET, DELETE THE MATCHED PORTION FROM THE INPUT AND SET SWITCH. IF NOT MET, RESET SWITCH.
ID	IDENTIFIER	AFTER DELETING INITIAL BLANKS IN THE INPUT STRING, TEST IF IT BEGINS WITH AN IDENTIFIER, IE., A LETTER FOLLOWED BY A SEQUENCE OF LETTERS AND/OR DIGITS. IF SO, DELETE THE IDENTIFIER AND SET SWITCH. IF NOT, RESET SWITCH.
NUM	NUMBER	AFTER DELETING INITIAL BLANKS IN THE INPUT STRING, TEST IF IT BEGINS WITH A NUMBER. A NUMBER IS A STRING OF DIGITS WHICH MAY CONTAIN IMBEDED PERIODS, BUT MAY NOT BEGIN OR END WITH A PERIOD. MOREOVER, NO TWO PERIODS MAY BE NEXT TO ONE ANOTHER. IF A NUMBER IS FOUND, DELETE IT AND SET SWITCH. IF NOT, RESET SWITCH.
SR	STRING	AFTER DELETING INITIAL BLANKS IN THE INPUT STRING, TEST IF IT BEGINS WITH A STRING, IE., A SINGLE QUOTE FOLLOWED BY A SEQUENCE OF ANY CHARACTERS OTHER THAN SINGLE QUOTE FOLLOWED BY ANOTHER SINGLE QUOTE. IF A STRING IS FOUND, DELETE IT AND SET SWITCH. IF NOT, RESET SWITCH.
CLL AAA	CALL	ENTER THE SUBROUTINE BEGINNING IN LOCATION AAA. IF THE TOP TWO TERMS OF THE STACK ARE BLANK, PUSH THE STACK DOWN BY ONE CELL. OTHERWISE, PUSH IT DOWN BY THREE CELLS. SET A FLAG IN THE STACK TO INDICATE WHETHER IT HAS BEEN PUSHED BY ONE OR THREE CELLS. THIS FLAG AND THE EXIT ADDRESS GO INTO THE THIRD CELL. CLEAR THE TOP TWO CELLS TO BLANKS TO INDICATE THAT THEY CAN ACCEPT ADDRESSES WHICH MAY BE GENERATED WITHIN THE SUBROUTINE.

Figure 6.1

R	RETURN	RETURN TO THE EXIT ADDRESS, POPPING UP THE STACK BY ONE OR THREE CELLS ACCORDING TO THE FLAG. IF THE STACK IS POPPED BY ONLY ONE CELL, THEN CLEAR THE TOP TWO CELLS TO BLANKS, BECAUSE THEY WERE BLANK WHEN THE SUBROUTINE WAS ENTERED.
SET	SET	SET BRANCH SWITCH ON.
B AAA	BRANCH	BRANCH UNCONDITIONALLY TO LOCATION AAA.
BT AAA	BRANCH IF TRUE	BRANCH TO LOCATION AAA IF SWITCH IS ON. OTHERWISE, CONTINUE IN SEQUENCE.
BF AAA	BRANCH IF FALSE	BRANCH TO LOCATION AAA IF SWITCH IS OFF. OTHERWISE, CONTINUE IN SEQUENCE.
BE	BRANCH TO ERROR IF FALSE	HALT IF SWITCH IS OFF, OTHERWISE, CONTINUE IN SEQUENCE.
CL STRING	COPY LITERAL	OUTPUT THE VARIABLE LENGTH STRING GIVEN AS THE ARGUMENT. A BLANK CHARACTER WILL BE INSERTED IN THE OUTPUT FOLLOWING THE STRING.
CI	COPY INPUT	OUTPUT THE LAST SEQUENCE OF CHARACTERS DELETED FROM THE INPUT STRING. THIS COMMAND MAY NOT FUNCTION PROPERLY IF THE LAST COMMAND WHICH COULD CAUSE DELETION FAILED TO DO SO.
GN1	GENERATE 1	THIS CONCERNS THE CURRENT LABEL 1 CELL, IE., THE NEXT TO TOP CELL IN THE STACK, WHICH IS EITHER CLEAR OR CONTAINS A GENERATED LABEL. IF CLEAR, GENERATE A LABEL AND PUT IT INTO THAT CELL. WHETHER THE LABEL HAS JUST BEEN PUT INTO THE CELL OR WAS ALREADY THERE, OUTPUT IT. FINALLY, INSERT A BLANK CHARACTER IN THE OUTPUT FOLLOWING THE LABEL.
GN2	GENERATE 2	SAME AS GN1, EXCEPT THAT IT CONCERNS THE CURRENT LABEL 2 CELL, IE., THE TOP CELL IN THE STACK.
LB	LABEL	SET THE OUTPUT COUNTER TO CARD COLUMN 1.
OUT	OUTPUT	PUNCH CARD AND RESET OUTPUT COUNTER TO CARD COLUMN 8.

Figure 6.2

CONSTANT AND CONTROL CODES

ADR IDENT	ADDRESS	PRODUCES THE ADDRESS WHICH IS ASSIGNED TO THE GIVEN IDENTIFIER AS A CONSTANT.
END	END	DENOTES THE END OF THE PROGRAM.

Figure 6.3

VALGOL II COMPILER WRITTEN IN META II
FIGURE 7

```
.SYNTAX PROGRAM

ARRAYPART = '(.' EXP '.)' .OUT('AIA') .,

CALLPART = '(' .OUT('LDF') (EXP $(',' EXP) /
     .EMPTY) ')' .OUT('CLL') .,

VARIABLE = .ID .OUT('LD ' *) (ARRAYPART / .EMPTY) .,

PRIMARY = 'WHOLE' '(' EXP ')' .OUT('WHL') /
     .ID .OUT('LD ' *) (ARRAYPART / CALLPART / .EMPTY) /
     '.TRUE' .OUT('SET') / '.FALSE' .OUT('RST') /
     '0 ' .OUT('RST') / '1 ' .OUT('SET') /
     .NUMBER .OUT('LDL' *) /
     '(' EXP ')' .,

TERM = PRIMARY $ ('*' PRIMARY .OUT('MLT') /
     '/' PRIMARY .OUT('DIV') /
     './.' PRIMARY .OUT('DIV') .OUT('WHL') ) .,

EXP2 = '-' TERM .OUT('NEG') /
     '+' TERM / TERM .,

EXP1 = EXP2 $('+' TERM .OUT('ADD') /
     '-' TERM .OUT('SUB')) .,

RELATION = EXP1 (
     '.L=' EXP1 .OUT('LEQ') /
     '.L' EXP1 .OUT('LES') /
     '.=' EXP1 .OUT('EQU') /
     '.-=' EXP1 .OUT('EQU') .OUT('NOT') /
     '.G=' EXP1 .OUT('LES') .OUT('NOT') /
     '.G' EXP1 .OUT('LEQ') .OUT('NOT') /
     .EMPTY) .,

BPRIMARY = '.-' RELATION .OUT('NOT') /
     RELATION .,

BTERM = BPRIMARY $ ('.&' .OUT('BF ' *1)
     .OUT('POP') BPRIMARY)
     .LABEL *1 .,

BEXP1 = BTERM $( '.V' .OUT('BT ' *1)
     .OUT('POP') BTERM)
     .LABEL *1 .,

IMPLICATION1 = '.IMP' .OUT('NOT')
     .OUT('BT ' *1) .OUT('POP')
     BEXP1 .LABEL *1 .,

IMPLICATION = BEXP1 $ IMPLICATION1 .,
```

Figure 7.1

```
EQUIV = IMPLICATION $('.EQ' .OUT('EQU') ) .,

EXP = '.IF' EXP '.THEN' .OUT('BFP' *1)
     EXP .OUT('B  ' *2) .LABEL *1
     '.ELSE' EXP .LABEL *2 /
     EQUIV .,

ASSIGNPART = '=' EXP ( ASSIGNPART .OUT('ST') /
     .EMPTY .OUT('SST') ) .,

ASSIGNCALLST = .ID .OUT('LD ' *) (ARRAYPART ASSIGNPART /
     ASSIGNPART / (CALLPART / .EMPTY
     .OUT('LDF') .OUT('CLL') )
     .OUT('POP') ) .,

UNTILST = '.UNTIL' .LABEL *1 EXP
     '.DO' .OUT('BTP' *2) ST
     .OUT('B  ' *1) .LABEL *2 .,

WHILECLAUSE = '.WHILE' .OUT('BF ' *1)
     .OUT('POP') EXP .LABEL *1 / .EMPTY .,

FORCLAUSE = VARIABLE '=' .OUT('FLP')
     .OUT('BFP' *1) EXP '.STEP'
     .OUT('SST') .OUT('B  ' *2)
     .LABEL *1 EXP '.UNTIL' .OUT('ADS')
     .LABEL *2 .OUT('RSR') EXP
     .OUT('LEQ') WHILECLAUSE '.DO' .,

FORST = '.FOR' .OUT('SET') .LABEL *1
     FORCLAUSE .OUT('BFP' *2) ST
     .OUT('RST') .OUT('B  ' *1)
     .LABEL *2 .,

IOCALL = 'READ' '(' VARIABLE ',' EXP ')' .OUT('RED') /
     'WRITE' '(' VARIABLE ',' EXP ')' .OUT('WRT') /
     'EDIT' '(' EXP ',' .STRING
     .OUT('EDT' *) ')' /
     'PRINT' .OUT('PNT') /
     'EJECT' .OUT('EJT') .,
```

```
IDSEQ1 = .ID .LABEL* .OUT('BLK 1') ..

IDSEQ = IDSEQ1 $(',' IDSEQ1) ..

TYPEDEC = '.REAL' IDSEQ ..

ARRAY1 = .ID .LABEL * '(.' '0' '..' .NUMBER
     .OUT('BLK 1') .OUT('BLK' *) ').' ..

ARRAYDEC = '.ARRAY' ARRAY1 $( ',' ARRAY1) ..

PROCEDURE = '.PROCEDURE' .ID .LABEL *
     .LABEL *1 .OUT('BLK 1') '('
     (IDSEQ / .EMPTY) ')' .OUT('SP  1') '.,'
     ST .OUT('R  ' *1) ..
```

Figure 7.2

```
DEC = TYPEDEC / ARRAYDEC / PROCEDURE ..

BLOCK = '.BEGIN' .OUT('B  ' *1) $(DEC '.,')
     .LABEL *1 ST $('.,' ST) '.END'
     (.ID / .EMPTY) ..

UNCONDITIONALST = IOCALL / ASSIGNCALLST /
     BLOCK ..

CONDST = '.IF' EXP '.THEN' .OUT('BFP' *1)
     (UNCONDITIONALST '.ELSE' .OUT('B  ' *2)
     .LABEL *1 ST .LABEL *2 / .EMPTY
     .LABEL *1) / (FORST / UNTILST)
     .LABEL *1) ..

ST = CONDST / UNCONDITIONALST / FORST /
     UNTILST / .EMPTY ..

PROGRAM = BLOCK
     .OUT('HLT') .OUT('SP  1') .OUT('END') ..

.END
```

Figure 7.3

ORDER LIST OF THE VALGOL II MACHINE
FIGURE 8

MACHINE CODES

LD AAA	LOAD	PUT THE ADDRESS AAA ON TOP OF THE STACK.
LDL NUMBER	LOAD LITERAL	PUT THE GIVEN NUMBER ON TOP OF THE STACK.
SET	SET	PUT THE INTEGER 1 ON TOP OF THE STACK.
RST	RESET	PUT THE INTEGER 0 ON TOP OF THE STACK.
ST	STORE	STORE THE CONTENTS OF THE REGISTER, STACK1, IN THE ADDRESS WHICH IS ON TOP OF THE STACK, THEN POP UP THE STACK.
ADS	ADD TO STORAGE NOTE 1	ADD THE NUMBER ON TOP OF THE STACK TO THE NUMBER WHOSE ADDRESS IS NEXT TO THE TOP, AND PLACE THE SUM IN THE REGISTER, STACK1. THEN STORE THE CONTENTS OF THAT REGISTER IN THAT ADDRESS, AND POP THE TOP TWO ITEMS OUT OF THE STACK.
SST	SAVE AND STORE NOTE 1	PUT THE NUMBER ON TOP OF THE STACK INTO THE REGISTER, STACK1. THEN STORE THE CONTENTS OF THAT REGISTER IN THE ADDRESS WHICH IS THE NEXT TO TOP TERM OF THE STACK. THE TOP TWO ITEMS ARE POPPED OUT OF THE STACK.
RSR	RESTORE	PUT THE CONTENTS OF THE REGISTER, STACK1, ON TOP OF THE STACK.
ADD	ADD NOTE 2	REPLACE THE TWO NUMBERS WHICH ARE ON TOP OF THE STACK WITH THEIR SUM.
SUB	SUBTRACT NOTE 2	SUBTRACT THE NUMBER WHICH IS ON TOP OF THE STACK FROM THE NUMBER WHICH IS NEXT TO THE TOP, THEN REPLACE THEM BY THIS DIFFERENCE.
MLT	MULTIPLY NOTE 2	REPLACE THE TWO NUMBERS WHICH ARE ON TOP OF THE STACK WITH THEIR PRODUCT.
DIV	DIVIDE NOTE 2	DIVIDE THE NUMBER WHICH IS NEXT TO THE TOP OF THE STACK BY THE NUMBER WHICH IS ON TOP OF THE STACK, THEN REPLACE THEM BY THIS QUOTIENT.

Figure 8.1

NEG	NEGATE NOTE 1	CHANGE THE SIGN OF THE NUMBER ON TOP OF THE STACK.
WHL	WHOLE	TRUNCATE THE NUMBER WHICH IS ON TOP OF THE STACK.
NOT	NOT	IF THE TOP TERM IN THE STACK IS THE INTEGER 0, THEN REPLACE IT WITH THE INTEGER 1, OTHERWISE, REPLACE IT WITH THE INTEGER 0.
LEQ	LESS THAN OR EQUAL NOTE 2	IF THE NUMBER WHICH IS NEXT TO THE TOP OF THE STACK IS LESS THAN OR EQUAL TO THE NUMBER ON TOP OF THE STACK, THEN REPLACE THEM WITH THE INTEGER 1. OTHERWISE, REPLACE THEM WITH THE INTEGER 0.
LES	LESS THAN NOTE 2	IF THE NUMBER WHICH IS NEXT TO THE TOP OF THE STACK IS LESS THAN THE NUMBER ON TOP OF THE STACK, THEN REPLACE THEM WITH THE INTEGER 1. OTHERWISE, REPLACE THEM WITH THE INTEGER 0.
EQU	EQUAL NOTE 2	COMPARE THE TWO NUMBERS ON TOP OF THE STACK. REPLACE THEM BY THE INTEGER 1, IF THEY ARE EQUAL, OR BY THE INTEGER 0, IF THEY ARE UNEQUAL.
B AAA	BRANCH	BRANCH TO THE ADDRESS AAA.
BT AAA	BRANCH TRUE	BRANCH TO THE ADDRESS AAA IF THE TOP TERM IN THE STACK IS NOT THE INTEGER 0. OTHERWISE, CONTINUE IN SEQUENCE. DO NOT POP UP THE STACK.
BF AAA	BRANCH FALSE	BRANCH TO THE ADDRESS AAA IF THE TOP TERM IN THE STACK IS THE INTEGER 0. OTHERWISE, CONTINUE IN SEQUENCE. DO NOT POP UP THE STACK.
BTP AAA	BRANCH TRUE AND POP	BRANCH TO THE ADDRESS AAA IF THE TOP TERM IN THE STACK IS NOT THE INTEGER 0. OTHERWISE, CONTINUE IN SEQUENCE. IN EITHER CASE, POP UP THE STACK.
BFP AAA	BRANCH FALSE AND POP	BRANCH TO THE ADDRESS AAA IF THE TOP TERM IN THE STACK IS THE INTEGER 0. OTHERWISE, CONTINUE IN SEQUENCE. IN EITHER CASE, POP UP THE STACK.

Figure 8.2

CLL	CALL	ENTER A PROCEDURE AT THE ADDRESS WHICH IS BELOW THE FLAG.
LDF	LOAD FLAG	PUT THE ADDRESS WHICH IS IN THE FLAG REGISTER ON TOP OF THE STACK, AND PUT THE ADDRESS OF THE TOP OF THE STACK INTO THE FLAG REGISTER.
R AAA	RETURN	RETURN FROM PROCEDURE.
AIA	ARRAY INCREMENT ADDRESS	INCREMENT THE ADDRESS WHICH IS NEXT TO THE TOP OF THE STACK BY THE INTEGER WHICH IS ON TOP OF THE STACK, AND REPLACE THESE BY THE RESULTING ADDRESS.
FLP	FLIP	INTERCHANGE THE TOP TWO TERMS OF THE STACK.
POP	POP	POP UP THE STACK.
EDT STRING	EDIT NOTE 1	ROUND THE NUMBER WHICH IS ON TOP OF THE STACK TO THE NEAREST INTEGER N. MOVE THE GIVEN STRING INTO THE PRINT AREA SO THAT ITS FIRST CHARACTER FALLS ON PRINT POSITION N. IN CASE THIS WOULD CAUSE CHARACTERS TO FALL OUTSIDE THE PRINT AREA, NO MOVEMENT TAKES PLACE.
PNT	PRINT	PRINT A LINE, THEN SPACE AND CLEAR THE PRINT AREA.
EJT	EJECT	POSITION THE PAPER IN THE PRINTER TO THE TOP LINE OF THE NEXT PAGE.
RED	READ	READ THE FIRST N NUMBERS FROM A CARD AND STORE THEM BEGINNING IN THE ADDRESS WHICH IS NEXT TO THE TOP OF THE STACK. THE INTEGER N IS THE TOP TERM OF THE STACK. POP OUT BOTH THE ADDRESS AND THE INTEGER. CARDS ARE PUNCHED WITH UP TO 10 EIGHT DIGIT NUMBERS. DECIMAL POINT IS ASSUMED TO BE IN THE MIDDLE. AN 11-PUNCH OVER THE RIGHTMOST DIGIT INDICATES A NEGATIVE NUMBER.

Figure 8.3

WRT	WRITE	PRINT A LINE OF N NUMBERS BEGINNING IN THE ADDRESS WHICH IS NEXT TO THE TOP OF THE STACK. THE INTEGER N IS THE TOP TERM OF THE STACK. POP OUT BOTH THE ADDRESS AND THE INTEGER. TWELVE CHARACTER POSITIONS ARE ALLOWED FOR EACH NUMBER. THERE ARE FOUR DIGITS BEFORE AND FOUR DIGITS AFTER THE DECIMAL. LEADING ZEROES IN FRONT OF THE DECIMAL ARE CHANGED TO BLANKS. IF THE NUMBER IS NEGATIVE, A MINUS SIGN IS PRINTED IN THE POSITION BEFORE THE FIRST NON-BLANK CHARACTER.
HLT	HALT	HALT.

CONSTANT AND CONTROL CODES

SP N	SPACE	N = 1--9. CONSTANT CODE PRODUCING N BLANK SPACES.
BLK NNN	BLOCK	PRODUCES A BLOCK OF NNN EIGHT CHARACTER WORDS.
END	END	DENOTES THE END OF THE PROGRAM.

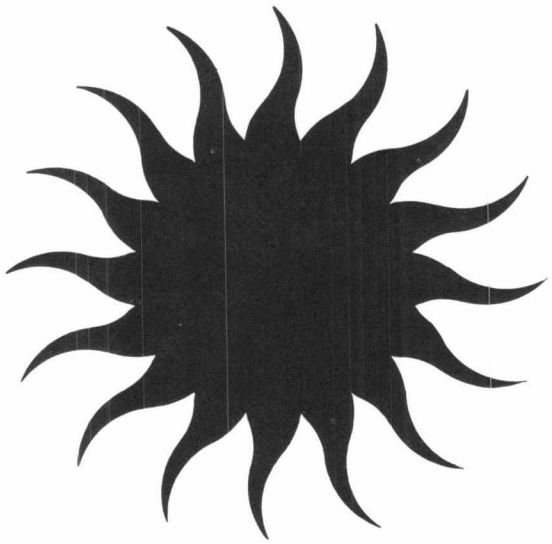
NOTE 1. IF THE TOP ITEM IN THE STACK IS AN ADDRESS, IT IS REPLACED BY ITS CONTENTS BEFORE BEGINNING THIS OPERATION.

NOTE 2. SAME AS NOTE 1, BUT APPLIES TO THE TOP TWO ITEMS.

Figure 8.4

EXAMPLE PROGRAM IN VALGOL II
FIGURE 9

```
.BEGIN
.PROCEDURE DETERMINANT(A, N) ;;
.BEGIN
.PROCEDURE DUMP() ;;
.BEGIN
.REAL D ;;
.FOR D = 0 .STEP 1 .UNTIL N-1 .DO
     WRITE(MATRIX(. N*D .), N) ;;
PRINT
.END DUMP ;;
.PROCEDURE ABS(X) ;;
     ABS = .IF 0 .L= X .THEN X .ELSE -X ;;
.REAL PRODUCT, FACTOR, TEMP, R, I, J ;;
PRODUCT = 1 ;;
.FOR R = 0 .STEP 1 .UNTIL N-2
.WHILE PRODUCT .-= 0 .DO .BEGIN
     I = R ;;
     .FOR J = R+1 .STEP 1 .UNTIL N-1 .DO
          .IF ABS( A(. N*I + R .) ) .L
          ABS( A(. N*J + R .) ) .THEN
               I = J ;;
     .IF A(. N*I + R .) .= 0 .THEN
          PRODUCT = 0
     .ELSE
          .IF I .-= R .THEN .BEGIN
               PRODUCT = -PRODUCT ;;
               .FOR J = R .STEP 1 .UNTIL N-1 .DO
               .BEGIN
                    TEMP = A(. N*R + J .) ;;
                    A(. N*R + J .) = A(. N* I + J .) .
                    A(. N*I + J .) = TEMP .END .END ;;
          TEMP = A(. N*R + R .) ;;
          .FOR I = R+1 .STEP 1 .UNTIL N-1 .DO
          .BEGIN
               FACTOR = A(. N*I + R .) / TEMP ;;
               .FOR J = R .STEP 1 .UNTIL N-1 .DO
                    A(. N*I + J .) = A(. N*I + J .)
                    -FACTOR * A(. N*R + J .) ;;
          DUMP
          .END .END ;;
.FOR I = 0 .STEP 1 .UNTIL N-1 .DO
     PRODUCT = PRODUCT * A(. N*I + I .) ;;
DETERMINANT = PRODUCT
.END DETERMINANT ;;
.REAL M, W, T ;; .ARRAY MATRIX (. 0 ;; 24 .) ;;
.UNTIL .FALSE .DO .BEGIN
     EDIT(1, 'FIND DETERMINANT OF' ) ;; PRINT.; PRINT.;
     READ(M, 1) ;;
     .FOR W = 0 .STEP 1 .UNTIL M-1 .DO .BEGIN
          READ(MATRIX (. M*W .), M) ;;
          WRITE(MATRIX (. M*W .), M) .END ;;
     PRINT ;;  T = DETERMINANT (MATRIX, M) ;;
     WRITE(T, 1) ;; PRINT;; PRINT .END
.END PROGRAM
```



A Northstar BASIC Cross-Reference Tool

BY TITUS PURDIN
5901 JFK # 101
N. Little Rock, Arkansas 72116

In the course of two years of micro computing, I cannot count the number of times friends and acquaintances have offered me copies of programs they have written, only to be disappointed because their BASIC and my BASIC were so vastly different. In many cases, when the offered programs were just too good to pass up, I have locked myself in the study for an evening and made the necessary conversions. It isn't a job that I relish. And while I haven't found an easy way to accomplish it, I have developed, over time, some tools to ease the burden.

This particular program accepts programs in BASIC on Northstar disks and compiles cross-reference lists for all variables, string variables, array variables, and line numbers. I have found this to be a valuable assist when converting programs from one BASIC to another and when compacting a program by placing multiple statements on a single line. Additionally, I have discovered that such lists, annotated with the purpose of each variable, are excellent pieces of documentation.

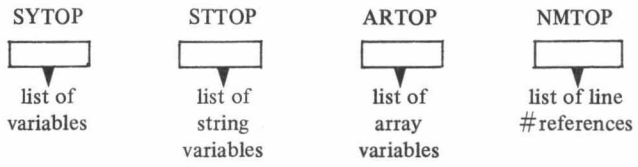
This program is written in 8080 assembly language and requires 13.4K of RAM to accommodate the program and the table space. The program itself requires 1.64K, and 12K is allocated for space in which to build the cross-reference tables. This latter space is more than sufficient for the largest programs I have. It could be reduced if necessary to save space by changing the allocation in line 5090 and changing the byte against which the high order byte of the address is compared at line 0945 to agree with the space allocated.

The program is designed to run at 2A00H and use the I/O and disk drivers currently in your Northstar DOS. References to these drivers are contained in the EQU vectors at the opening of the assembly language listing for ease in changing them if your drivers or your COS are not standard.

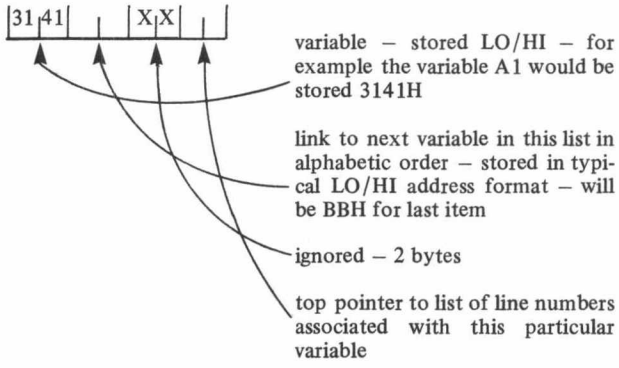
The program creates four separate lists:

1. Variables
2. String variables
3. Array variables
4. Line # references

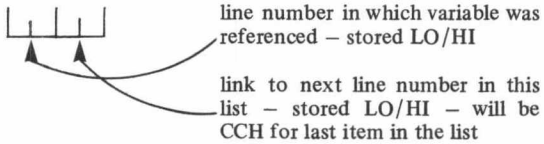
Each of these are supported as linear linked lists in alphabetic soft order. The storage areas SYTOP, STTOP, ARTOP, and NMTOP are the top pointers for these four lists, and the marker BBH is used to mark the end of each of these lists.



Each entry in each of the four lists has an accompanying top pointer for its associated list of line numbers. The mark CCH is used to mark the end of each line number list. A typical variable item looks like this:

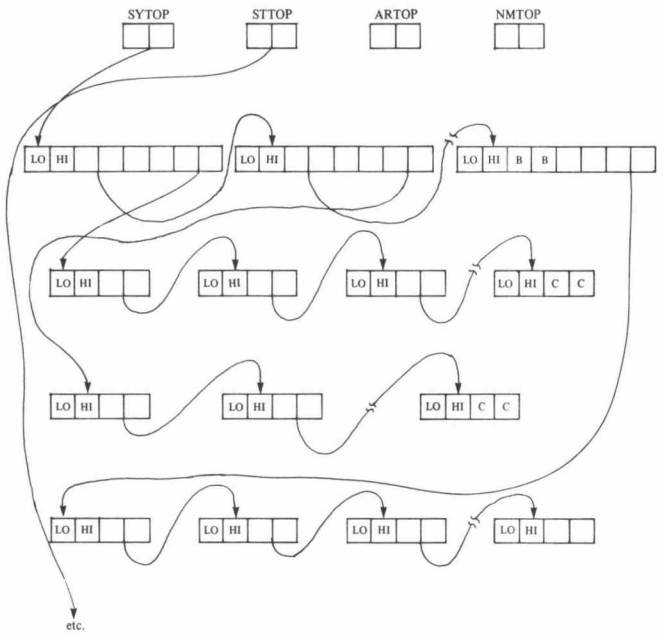


The list of line numbers associated with each variable in each list is constructed in a similar, but simpler, fashion:



The entire program is read from the disk, one 256 byte block at a time, and scanned for the purpose of constructing these lists. When this is completed, the lists are followed in order from beginning to end and the stored variables and line numbers are formated and printed to produce the cross-reference lists. Each list is printed with its own header, i.e. STRING VARIABLES. And the entry NULL LIST is printed for any list that is empty.

The diagram below represents the multi-list structure in somewhat stylized fashion:



The variables which were found in the program are stored in ASCII in the appropriate list so their conversion for printing is straightforward. The line numbers that are placed in the lists, however, are stored as two byte LO/HI binary values. The algorithm used to convert these to ASCII for printing is fairly simple, if somewhat bulky. Taking the binary value as an ordered 16 bit field, the program uses a 5 byte work area to construct the resulting decimal number. A table is provided (BASE:) which represents, as 5 single digits, the decimal number associated with each bit position of the 16 bit binary value. If a bit in the binary value is "on," then the five digits associated with that bit position are added to the five byte work area along with any necessary "carry's" from byte to byte. If, on the other hand, a bit in the binary value is "off," then the pointer into the table (BASE:) is bumped down 5 to point to the next series of 5 values. When this process is complete, 30H is added to each byte of the 5 byte work area to yield the desired ASCII value.

This program has proven useful in its own right. But it is worth mentioning that a modification of this same idea, directed solely at line number references, would be the basis for a program which would "compact" BASIC programs by removing unnecessary line numbers. Similarly, this sort of cross-reference compilation would form the basis of "pass one" of more sophisticated software such as a compiler. A compiler! Now there's an idea.

Listing

```
                        0005 ; THIS PROGRAM PROVIDES SOFTWARE SUPPORT FOR
                        0010 ; PROGRAMS WRITTEN IN NORTHSTAR BASIC. IT COMPILES
                        0015 ; AND PRINTS FOUR LISTS: 1)A LIST OF VARIABLE NAMES
                        0020 ; USED AND LINES IN WHICH EACH OF THEM OCCURS, 2)A
                        0025 ; LIST OF STRING VARIABLE NAMES USED AND LINES IN
                        0030 ; WHICH EACH OCCURS, 3)A LIST OF ARRAY VARIABLE
                        0035 ; NAMES AND LINES IN WHICH EACH OCCURS, AND 4)A
                        0040 ; LIST OF LINE NUMBERS REFERENCED IN GOTO AND GOSUB
                        0045 ; STATEMENTS AND LINE NUMBERS IN WHICH THEY ARE
                        0050 ; REFERENCED.
                        0055 ;
2A00                    0060        ORG   2A00H
2A00 214233             0065        LXI   H,STACK+49
2A03 F9                 0070        SPHL
2010 =                  0075 CIN:   EQU   2010H
200D =                  0080 COUT:  EQU   200DH
201C =                  0085 DIRCK: EQU   201CH
2022 =                  0090 RDDSK: EQU   2022H
2028 =                  0095 EXIT:  EQU   2028H
                        0100 ;
                        0105 ; PRINT HEADER
                        0110 ;
2A04 0E1F               0115 START: MVI   C,31
2A06 218730             0120        LXI   H,HDR1
2A09 CD5E30             0125        CALL  PRINT
2A0C CD6830             0130        CALL  CRLF
                        0135 ;
                        0140 ; PRINT FILE NAME PROMPT
                        0045 ;
2A0F 0E11               0150 ASKFL: MVI   C,17
2A11 21A630             0155        LXI   H,HDR2
2A14 CD5E30             0160        CALL  PRINT
2A17 0E0A               0165        MVI   C,10
2A19 218731             0170        LXI   H,PNAME
                        0175 ;
                        0180 ; BLANK PROGRAM NAME AREA
                        0185 ;
2A1C 3620               0190 BLNAM: MVI   M,20H
2A1E 23                 0195        INX   H
2A1F 0D                 0200        DCR   C
2A20 C21C2A             0205        JNZ   BLNAM
2A23 0E09               0210        MVI   C,9
2A25 218731             0215        LXI   H,PNAME
                        0220 ;
                        0225 ; ACCEPTS FILE NAME. USES REG C AS COUNTER AND
                        0230 ; ACCEPTS UP TO 8 CHARACTERS. TAKES CARE OF
                        0235 ; BLANKING CHAR IN RESPONSE TO BACKSPACE (5FH).
                        0240 ; ROUTINE TERMINATES WHEN CR (0DH) IS ENCOUNTERED.
                        0245 ;
2A28 CD1020             0250 INFIL: CALL  CIN
2A2B 32B232             0255        STA   HOLDT
2A2E FE0D               0260        CPI   0DH
2A30 CA6F2A             0265        JZ    INUNT
2A33 FE5F               0270        CPI   5FH
2A35 C2542A             0275        JNZ   STORE
2A38 3E09               0280        MVI   A,9
2A3A B9                 0285        CMP   C
2A3B CA282A             0290        JZ    INFIL
2A3E 3AB232             0295        LDA   HOLDT
2A41 2B                 0300        DCX   H
2A42 0C                 0305        INR   C
2A43 47                 0310        MOV   B,A
2A44 CD0D20             0315        CALL  COUT
2A47 0620               0320        MVI   B,20H
2A49 CD0D20             0325        CALL  COUT
2A4C 065F               0330        MVI   B,5FH
2A4E CD0D20             0335        CALL  COUT
2A51 C3282A             0340        JMP   INFIL
                        0345 ;
                        0350 ; PLACES CHARACTERS FROM INFIL: SEQUENTIALLY
                        0355 ; INTO AREA CALLED 'PNAME' TO BUILD INPUT
                        0360 ; FILE NAME. PRINTS 'NAME TOO LONG' ERROR
                        0365 ; IF 8 CHARACTERS ARE EXCEEDED.
                        0370 ;
2A54 77                 0375 STORE: MOV   M,A
2A55 47                 0380        MOV   B,A
2A56 CD0D20             0385        CALL  COUT
2A59 23                 0390        INX   H
2A5A 0D                 0395        DCR   C
2A5B C2282A             0400        JNZ   INFIL
2A5E CD6830             0405        CALL  CRLF
2A61 0E10               0410        MVI   C,16
2A63 211031             0415        LXI   H,HDR7
2A66 CD5E30             0420        CALL  PRINT
2A69 CD6830             0425        CALL  CRLF
2A6C C30F2A             0430        JMP   ASKFL
                        0435 ;
                        0440 ; ACCEPTS DISK UNIT NUMBER. CHECKS TO MAKE
                        0445 ; SURE INPUT IS ASCII 1, 2, OR 3.
                        0450 ;
2A6F 0E0C               0455 INUNT: MVI   C,12
2A71 21B730             0460        LXI   H,HDR3
2A74 CD6830             0465        CALL  CRLF
2A77 CD5E30             0470        CALL  PRINT
2A7A 219131             0475        LXI   H,DRIVE
2A7D CD1020             0480        CALL  CIN
2A80 47                 0485        MOV   B,A
2A81 CD0D20             0490        CALL  COUT
2A84 FE31               0495        CPI   31H
2A86 CAA42A             0500        JZ    GETCR
2A89 FE32               0505        CPI   32H
2A8B CAA42A             0510        JZ    GETCR
2A8E FE33               0515        CPI   33H
2A90 CAA42A             0520        JZ    GETCR
                        0525 ;
```

```
                        0530 ; PRINTS 'INVALID UNIT' ERROR IF INPUT
                        0535 ; IS NOT EQUAL TO ASCII 1, 2, OR 3.
                        0540 ;
2A93 CD6830             0545 ERUNT: CALL   CRLF
2A96 0E0C               0550        MVI    C,12
2A98 212031             0555        LXI    H,HDR8
2A9B CD5E30             0560        CALL   PRINT
2A9E CD6830             0565        CALL   CRLF
2AA1 C36F2A             0570        JMP    INUNT
                        0575 ;
                        0580 ; TRIMS ASCII INPUT UNIT # TO BINARY VALUE
                        0585 ; AND STORES IT IN AREA CALLED 'DRIVE'. ACCEPTS
                        0590 ; CR TO CONTINUE. ANY OTHER INPUT CAUSES
                        0595 ; 'INVALID UNIT' ERROR. PUTS ADDRESS OF PROGRAM
                        0600 ; NAME IN REG H,L AND VALUE OF DRIVE # IN REG
                        0605 ; A. CALLS DOS DIRECTORY LOOKUP ROUTINE
                        0610 ; AT 201CH. IF CARRY BIT IS SET ON RETURN
                        0615 ; 'NO PROGRAM' ERROR IS PRINTED.
                        0620 ;
2AA4 E60F               0625 GETCR: ANI    0FH
2AA6 77                 0630        MOV    M,A
2AA7 CD1020             0635        CALL   CIN
2AAA 47                 0640        MOV    B,A
2AAB CD0D20             0645        CALL   COUT
2AAE FE0D               0650        CPI    0DH
2AB0 C2932A             0655        JNZ    ERUNT
2AB3 060A               0660        MVI    B,0AH
2AB5 CD0D20             0665        CALL   COUT
2AB8 3A9131             0670        LDA    DRIVE
2ABB 218731             0675        LXI    H,PNAME
2ABE 37                 0680        STC
2ABF 3F                 0685        CMC
2AC0 CD1C20             0690        CALL   DIRCK
2AC3 D2D42A             0695        JNC    CKTYP
2AC6 0E19               0700        MVI    C,25
2AC8 21C330             0705        LXI    H,HDR4
2ACB CD5E30             0710        CALL   PRINT
2ACE CD6830             0715        CALL   CRLF
2AD1 C30F2A             0720        JMP    ASKFL
                        0725 ;
                        0730 ; CHECKS FILE TYPE TO INSURE IT IS TYPE 2.
                        0735 ;
2AD4 0600               0740 CKTYP: MVI    B,0
2AD6 0E04               0745        MVI    C,4
2AD8 09                 0750        DAD    B
2AD9 3E02               0755        MVI    A,2
2ADB BE                 0760        CMP    M
2ADC CAED2A             0765        JZ     STADD
2ADF 0E12               0770        MVI    C,18
2AE1 21DC30             0775        LXI    H,HDR5
2AE4 CD5E30             0780        CALL   PRINT
2AE7 CD6830             0785        CALL   CRLF
2AEA C30F2A             0790        JMP    ASKFL
                        0795 ;
                        0800 ; STORES DATA FROM DISK DIRECTORY FOR THE FILE
                        0805 ; IN AREA CALLED 'DIR'.
                        0810 ;
2AED 2B                 0815 STADD: DCX    H
2AEE 2B                 0820        DCX    H
2AEF 2B                 0825        DCX    H
2AF0 2B                 0830        DCX    H
2AF1 0E08               0835        MVI    C,8
2AF3 110933             0840        LXI    D,DIR
2AF6 7E                 0845 STAD1: MOV    A,M
2AF7 12                 0850        STAX   D
2AF8 23                 0855        INX    H
2AF9 13                 0860        INX    D
2AFA 0D                 0865        DCR    C
2AFB C2F62A             0870        JNZ    STAD1
                        0875 ;
                        0880 ; SETS LINKS IN AREA CALLED 'TABLE'. INITIALIZES
                        0885 ; THE TOP POINTERS FOR THE FOUR LISTS 'SYTOP',
                        0890 ; 'STTOP', 'ARTOP', AND 'NMTOP'. INITIALIZES THE
                        0895 ; AVAILABILITY POINTER 'AVAIL'.
                        0900 ;
2AFE 2600               0905        MVI    H,0
2B00 2E00               0910        MVI    L,0
2B02 224333             0915        SHLD   ADDR
2B05 0600               0920        MVI    B,0
2B07 0E04               0925        MVI    C,4
2B09 118F33             0930        LXI    D,TABLE+2
2B0C 2A4333             0935 SLINK: LHLD   ADDR
2B0F 23                 0940        INX    H
2B10 3E0C               0945        MVI    A,0CH
2B12 BC                 0950        CMP    H
2B13 CA232B             0955        JZ     STOPS
2B16 224333             0960        SHLD   ADDR
2B19 62                 0965        MOV    H,D
2B1A 6B                 0970        MOV    L,E
2B1B 09                 0975        DAD    B
2B1C EB                 0980        XCHG
2B1D 73                 0985        MOV    M,E
2B1E 23                 0990        INX    H
2B1F 72                 0995        MOV    M,D
2B20 C30C2B             1000        JMP    SLINK
2B23 3EFF               1005 STOPS: MVI    A,0FFH
2B25 23                 1010        INX    H
2B26 77                 1015        MOV    M,A
2B27 26BB               1020        MVI    H,0BBH
2B29 2EBB               1025        MVI    L,0BBH
2B2B 22A632             1030        SHLD   SYTOP
2B2E 22A832             1035        SHLD   STTOP
2B31 22AA32             1040        SHLD   ARTOP
2B34 22AC32             1045        SHLD   NMTOP
2B37 218F33             1050        LXI    H,TABLE+2
2B3A 22AE32             1055        SHLD   AVAIL
2B3D CD2F30             1060        CALL   RDBLK
                        1065 ;
                        1070 ; BEGINS PROCESSING FOR A LINE OF BASIC CODE.
                        1075 ; STORES LINE NUMBER AND LINE LENGTH.
                        1080 ;
2B40 AF                 1085 NEWLN: XRA    A
2B41 329732             1090        STA    MARK
2B44 3C                 1095        INR    A
2B45 EB                 1100        XCHG
2B46 BE                 1105        CMP    M
2B47 CABC2D             1110        JZ     PRTRN
2B4A EB                 1115        XCHG
2B4B CD7330             1120        CALL   NXTCR
2B4E 1A                 1125        LDAX   D
2B4F 32A232             1130        STA    LOLIN
2B52 CD7330             1135        CALL   NXTCR
2B55 1A                 1140        LDAX   D
2B56 32A332             1145        STA    HILIN
2B59 AF                 1150        XRA    A
2B5A 32A132             1155        STA    LNTAB
2B5D CD7330             1160        CALL   NXTCR
2B60 62                 1165 NEWL1: MOV    H,D
2B61 6B                 1170        MOV    L,E
2B62 3E20               1175        MVI    A,20H
2B64 BE                 1180        CMP    M
2B65 CAB62B             1185        JZ     NEWL6
2B68 3E8F               1190        MVI    A,8FH
2B6A BE                 1195        CMP    M
2B6B C2732B             1200        JNZ    NEWL2
2B6E 3E01               1205        MVI    A,1
2B70 329732             1210        STA    MARK
2B73 3E22               1215 NEWL2: MVI    A,22H
2B75 BE                 1220        CMP    M
2B76 C2902B             1225        JNZ    NEWL4
2B79 3A9732             1230        LDA    MARK
2B7C FE02               1235        CPI    02H
2B7E C2882B             1240        JNZ    NEWL3
2B81 AF                 1245        XRA    A
2B82 329732             1250        STA    MARK
2B85 C3B62B             1255        JMP    NEWL6
2B88 3E02               1260 NEWL3: MVI    A,2
2B8A 329732             1265        STA    MARK
2B8D C3B62B             1270        JMP    NEWL6
2B90 3A9732             1275 NEWL4: LDA    MARK
2B93 FE00               1280        CPI    0H
2B95 C2B62B             1285        JNZ    NEWL6
2B98 3E9A               1290        MVI    A,9AH
2B9A BE                 1295        CMP    M
2B9B C2A42B             1300        JNZ    NEWL5
2B9E CDC82B             1305        CALL   STNUM
2BA1 C3B62B             1310        JMP    NEWL6
2BA4 3E40               1215 NEWL5: MVI    A,40H
2BA6 BE                 1320        CMP    M
2BA7 D2B62B             1325        JNC    NEWL6
2BAA 3E5A               1330        MVI    A,5AH
2BAC BE                 1335        CMP    M
2BAD DAB62B             1340        JC     NEWL6
2BB0 CDEA2B             1345        CALL   GETSY
2BB3 C3B92B             1350        JMP    NEWL7
2BB6 CD7330             1355 NEWL6: CALL   NXTCR
2BB9 3E0D               1360 NEWL7: MVI    A,0DH
2BBB 21A132             1365        LXI    H,LNTAB
2BBE BE                 1370        CMP    M
2BBF C2602B             1375        JNZ    NEWL1
2BC2 CD7330             1380        CALL   NXTCR
2BC5 C3402B             1385        JMP    NEWLN
                        1390 ;
                        1395 ; EXTRACTS A LINE NUMBER REFERENCE
                        1400 ; AND PLACES IT IN 'VALHI' AND 'VALLO'
                        1405 ; THEN CALLS THE SEARCH AND INSERT
                        1410 ; NODE ROUTINES
                        1415 ;
2BC8 CD7330             1420 STNUM: CALL   NXTCR
2BCB 1A                 1425        LDAX   D
2BCC 32A532             1430        STA    VALLO
2BCF CD7330             1435        CALL   NXTCR
2BD2 1A                 1440        LDAX   D
2BD3 32A432             1445        STA    VALHI
2BD6 AF                 1450        XRA    A
2BD7 32A132             1455        STA    LNTAB
2BDA 2AAC32             1460        LHLD   NMTOP
2BDD 22B032             1465        SHLD   TOP
2BE0 CD7A2C             1470        CALL   SERCH
2BE3 2AB032             1475        LHLD   TOP
2BE6 22AC32             1480        SHLD   NMTOP
2BE9 C9                 1485        RET
                        1490 ;
                        1495 ; EXTRACT A SYMBOL AND DETERMINE WHICH
                        1500 ; LIST IT BELONGS IN.  CALLS THE
                        1505 ; SEARCH AND INSERT NODE ROUTINES.
                        1510 ;
2BEA 3E20               1515 GETSY: MVI    A,20H
2BEC 32A532             1520        STA    VALLO
2BEF 32A432             1525        STA    VALHI
2BF2 1A                 1530        LDAX   D
2BF3 32A432             1535        STA    VALHI
2BF6 CD7330             1540        CALL   NXTCR
2BF9 3E0D               1545        MVI    A,0DH
2BFB 21A132             1550        LXI    H,LNTAB
2BFE BE                 1555        CMP    M
2BFF C2082C             1560        JNZ    GETS1
2C02 CD4A2C             1565        CALL   STRSY
2C05 C3492C             1570        JMP    GETS5
2C08 62                 1575 GETS1: MOV    H,D
2C09 6B                 1580        MOV    L,E
2C0A 3E2F               1585        MVI    A,2FH
2C0C BE                 1590        CMP    M
2C0D D22C2C             1595        JNC    GETS2
2C10 3E39               1600        MVI    A,39H
2C12 BE                 1605        CMP    M
2C13 DA2C2C             1610        JC     GETS2
2C16 1A                 1615        LDAX   D
2C17 32A532             1620        STA    VALLO
2C1A CD7330             1625        CALL   NXTCR
2C1D 3E0D               1630        MVI    A,0DH
2C1F 21A132             1635        LXI    H,LNTAB
2C22 BE                 1640        CMP    M
2C23 C22C2C             1645        JNZ    GETS2
```

```
2C26 CD4A2C    1650         CALL   STRSY
2C29 C3492C    1655         JMP    GETS5
2C2C 62        1660 GETS2:  MOV    H,D
2C2D 6B        1665         MOV    L,E
2C2E 3E24      1670         MVI    A,24H
2C30 BE        1675         CMP    M
2C31 C23A2C    1680         JNZ    GETS3
2C34 CD5A2C    1685         CALL   STRST
2C37 C3492C    1690         JMP    GETS5
2C3A 3EE0      1695 GETS3:  MVI    A,0E0H
2C3C BE        1700         CMP    M
2C3D C2462C    1705         JNZ    GETS4
2C40 CD6A2C    1710         CALL   STRAR
2C43 C3492C    1715         JMP    GETS5
2C46 CD4A2C    1720 GETS4:  CALL   STRSY
2C49 C9        1725 GETS5:  RET
               1730 ;
               1735 ; SETS SYMBOL LIST TOP POINTER AND CALLS
               1740 ; SEARCH AND INSERT NODE ROUTINES.
               1745 ;
2C4A 2AA632    1750 STRSY:  LHLD   SYTOP
2C4D 22B032    1755         SHLD   TOP
2C50 CD7A2C    1760         CALL   SERCH
2C53 2AB032    1765         LHLD   TOP
2C56 22A632    1770         SHLD   SYTOP
2C59 C9        1775         RET
               1780 ;
               1785 ; SETS TOP POINTER TO STRING LIST TOP AND
               1790 ; CALLS SEARCH AND INSERT NODE ROUTINES.
               1795 ;
2C5A 2AA832    1800 STRST:  LHLD   STTOP
2C5D 22B032    1805         SHLD   TOP
2C60 CD7A2C    1810         CALL   SERCH
2C63 2AB032    1815         LHLD   TOP
2C66 22A832    1820         SHLD   STTOP
2C69 C9        1825         RET
               1830 ;
               1835 ; SETS TOP POINTER TO ARRAY LIST TOP AND
               1840 ; CALLS SEARCH AND INSERT NODE ROUTINES.
               1845 ;
2C6A 2AAA32    1850 STRAR:  LHLD   ARTOP
2C6D 22B032    1855         SHLD   TOP
2C70 CD7A2C    1860         CALL   SERCH
2C73 2AB032    1865         LHLD   TOP
2C76 22AA32    1870         SHLD   ARTOP
2C79 C9        1875         RET
               1880 ;
               1885 ; SEARCH AND STORE ROUTINE
               1890 ;
2C7A 229C32    1895 SERCH:  SHLD   NEXT
2C7D EB        1900         XCHG
2C7E 229E32    1905         SHLD   POINT
2C81 3EBB      1910         MVI    A,0BBH
2C83 329832    1915         STA    LAST
2C86 329932    1920         STA    LAST+1
2C89 2AAE32    1925         LHLD   AVAIL
2C8C 229A32    1930         SHLD   PRSNT
2C8F 2B        1935         DCX    H
2C90 3AA432    1940         LDA    VALHI
2C93 77        1945         MOV    M,A
2C94 2B        1950         DCX    H
2C95 3AA532    1955         LDA    VALLO
2C98 77        1960         MOV    M,A
2C99 23        1965         INX    H
2C9A 23        1970         INX    H
2C9B 3EBB      1975         MVI    A,0BBH
2C9D 77        1980         MOV    M,A
2C9E 23        1985         INX    H
2C9F 77        1990         MOV    M,A
2CA0 23        1995         INX    H
2CA1 23        2000         INX    H
2CA2 23        2005         INX    H
2CA3 7E        2010         MOV    A,M
2CA4 32AE32    2015         STA    AVAIL
2CA7 3ECC      2020         MVI    A,0CCH
2CA9 77        2025         MOV    M,A
2CAA 23        2030         INX    H
2CAB 7E        2035         MOV    A,M
2CAC 32AF32    2040         STA    AVAIL+1
2CAF 3ECC      2045         MVI    A,0CCH
2CB1 77        2050         MOV    M,A
2CB2 2A9C32    2055         LHLD   NEXT
2CB5 3EBB      2060         MVI    A,0BBH
2CB7 BC        2065         CMP    H
2CB8 C2C92C    2070         JNZ    SERC2
2CBB BD        2075         CMP    L
2CBC C2C92C    2080         JNZ    SERC2
2CBF 2A9A32    2085 SERC1:  LHLD   PRSNT
2CC2 22B032    2090         SHLD   TOP
2CC5 CD752D    2095         CALL   LNNUM
2CC8 C9        2100         RET
2CC9 2B        2105 SERC2:  DCX    H
2CCA 56        2110         MOV    D,M
2CCB 3AA432    2115         LDA    VALHI
2CCE BA        2120         CMP    D
2CCF CA322D    2125         JZ     DEQUA
2CD2 D2FB2C    2130         JNC    DLESA
               2135 ;
               2140 ; SETS THE LINKS IN THE APPROPRIATE LIST
               2145 ; TO REFLECT THE PROPER SEQUENCE FOR THE
               2150 ; CURRENT ITEM.
               2155 ;
2CD5 2A9A32    2160 DGRTA:  LHLD   PRSNT
2CD8 3A9C32    2165         LDA    NEXT
2CDB 77        2170         MOV    M,A
2CDC 23        2175         INX    H
2CDD 3A9D32    2180         LDA    NEXT+1
2CE0 77        2185         MOV    M,A
2CE1 2A9832    2190         LHLD   LAST
2CE4 3EBB      2195         MVI    A,0BBH
2CE6 BC        2200         CMP    H
2CE7 C2EE2C    2205         JNZ    DGRT1
2CEA BD        2210         CMP    L
2CEB CABF2C    2215         JZ     SERC1
2CEE 3A9A32    2220 DGRT1:  LDA    PRSNT
2CF1 77        2225         MOV    M,A
2CF2 23        2230         INX    H
2CF3 3A9B32    2235         LDA    PRSNT+1
2CF6 77        2240         MOV    M,A
2CF7 CD752D    2245         CALL   LNNUM
2CFA C9        2250         RET
               2255 ;
               2260 ; CONTINUES TO SEARCH THE APPROPRIATE
               2265 ; LIST FOR THE PROPER PLACE IN WHICH
               2270 ; TO INSERT THE CURRENT ITEM.
               2275 ;
2CFB 3EBB      2280 DLESA:  MVI    A,0BBH
2CFD 2A9C32    2285         LHLD   NEXT
2D00 BC        2290         CMP    H
2D01 C2182D    2295         JNZ    DLES2
2D04 BD        2300         CMP    L
2D05 C2182D    2305         JNZ    DLES2
2D08 2A9832    2310 DLES1:  LHLD   LAST
2D0B 3A9A32    2315         LDA    PRSNT
2D0E 77        2320         MOV    M,A
2D0F 23        2325         INX    H
2D10 3A9B32    2330         LDA    PRSNT+1
2D13 77        2335         MOV    M,A
2D14 CD752D    2240         CALL   LNNUM
2D17 C9        2345         RET
2D18 2A9C32    2350 DLES2:  LHLD   NEXT
2D1B 229832    2355         SHLD   LAST
2D1E 5E        2360         MOV    E,M
2D1F 23        2365         INX    H
2D20 56        2370         MOV    D,M
2D21 EB        2375         XCHG
2D22 229C32    2380         SHLD   NEXT
2D25 3EBB      2385         MVI    A,0BBH
2D27 BC        2390         CMP    H
2D28 C2C92C    2395         JNZ    SERC2
2D2B BD        2400         CMP    L
2D2C C2C92C    2405         JNZ    SERC2
2D2F C3082D    2410         JMP    DLES1
               2415 ;
               2420 ; CHECKS TO SEE IF THE CURRENT ITEM IS
               2425 ; IS ALREADY IN THE APPROPRIATE LIST.
               2430 ;
2D32 2B        2435 DEQUA:  DCX    H
2D33 56        2440         MOV    D,M
2D34 3AA532    2445         LDA    VALLO
2D37 BE        2450         CMP    M
2D38 CA412D    2455         JZ     DEQU1
2D3B D2FB2C    2460         JNC    DLESA
2D3E C3D52C    2465         JMP    DGRTA
2D41 2AAE32    2470 DEQU1:  LHLD   AVAIL
2D44 2B        2475         DCX    H
2D45 2B        2480         DCX    H
2D46 2B        2485         DCX    H
2D47 3AAF32    2490         LDA    AVAIL+1
2D4A 77        2495         MOV    M,A
2D4B 2B        2500         DCX    H
2D4C 3AAE32    2505         LDA    AVAIL
2D4F 77        2510         MOV    M,A
2D50 2B        2515         DCX    H
2D51 2B        2520         DCX    H
2D52 2B        2525         DCX    H
2D53 EB        2530         XCHG
2D54 2AAE32    2535         LHLD   AVAIL
2D57 2B        2540         DCX    H
2D58 2B        2545         DCX    H
2D59 2B        2550         DCX    H
2D5A 2B        2555         DCX    H
2D5B 22AE32    2560         SHLD   AVAIL
2D5E EB        2565         XCHG
2D5F 3AAF32    2570         LDA    AVAIL+1
2D62 77        2575         MOV    M,A
2D63 2B        2580         DCX    H
2D64 3AAE32    2585         LDA    AVAIL
2D67 77        2590         MOV    M,A
2D68 22AE32    2595         SHLD   AVAIL
2D6B 2A9C32    2600         LHLD   NEXT
2D6E 229A32    2605         SHLD   PRSNT
2D71 CD752D    2610         CALL   LNNUM
2D74 C9        2615         RET
               2620 ;
               2625 ; INSERTS THE LINE NUMBER FOR THE
               2630 ; CURRENT ITEM IN THE LINE NUMBER
               2635 ; LIST FOR THAT ITEM.  THE NEW
               2640 ; LINE NUMBER IS ADDED AT THE END
               2645 ; OF THE LIST SINCE THEY ARE ENCOUNTERED
               2650 ; IN ORDER.
               2655 ;
2D75 2A9A32    2660 LNNUM:  LHLD   PRSNT
2D78 23        2665         INX    H
2D79 23        2670         INX    H
2D7A 23        2675         INX    H
2D7B 23        2680         INX    H
2D7C 3ECC      2685 LNNU1:  MVI    A,0CCH
2D7E BE        2690         CMP    M
2D7F C2B52D    2695         JNZ    LNNU3
2D82 23        2700         INX    H
2D83 BE        2705         CMP    M
2D84 C2B42D    2710         JNZ    LNNU2
2D87 2B        2715         DCX    H
2D88 3AAE32    2720         LDA    AVAIL
2D8B 77        2725         MOV    M,A
2D8C 23        2730         INX    H
2D8D 3AAF32    2735         LDA    AVAIL+1
2D90 77        2740         MOV    M,A
2D91 2AAE32    2745         LHLD   AVAIL
2D94 2B        2750         DCX    H
2D95 2B        2755         DCX    H
2D96 3AA232    2760         LDA    LOLIN
2D99 77        2765         MOV    M,A
2D9A 23        2770         INX    H
2D9B 3AA332    2775         LDA    HILIN
```

```
2D9E 77        2780          MOV   M,A
2D9F 23        2785          INX   H
2DA0 7E        2790          MOV   A,M
2DA1 32AE32    2795          STA   AVAIL
2DA4 3ECC      2800          MVI   A,0CCH
2DA6 77        2805          MOV   M,A
2DA7 23        2810          INX   H
2DA8 7E        2815          MOV   A,M
2DA9 32AF32    2820          STA   AVAIL+1
2DAC 3ECC      2825          MVI   A,0CCH
2DAE 77        2830          MOV   M,A
2DAF 2A9E32    2835          LHLD  POINT
2DB2 EB        2840          XCHG
2DB3 C9        2845          RET
2DB4 2B        2850 LNNU2:   DCX   H
2DB5 5E        2855 LNNU3:   MOV   E,M
2DB6 23        2860          INX   H
2DB7 56        2865          MOV   D,M
2DB8 EB        2870          XCHG
2DB9 C37C2D    2875          JMP   LNNU1
               2880 ;
               2885 ; PRINTS THE NECESSARY HEADERS AND
               2890 ; LOADS THE APPROPRIATE LIST TOP POINTERS
               2895 ; FOR USE BY THE SEARCH AND PRINT ROUTINES.
               2900 ;
2DBC 0E12      2905 PRTRN:   MVI   C,18
2DBE 214131    2910          LXI   H,HDR11
2DC1 CD6830    2915          CALL  CRLF
2DC4 CD5E30    2920          CALL  PRINT
2DC7 CD6830    2925          CALL  CRLF
2DCA 2AA632    2930          LHLD  SYTOP
2DCD 22B032    2935          SHLD  TOP
2DD0 3E01      2940          MVI   A,1
2DD2 329732    2945          STA   MARK
2DD5 CD2F2E    2950          CALL  FNDSY
2DD8 CD6830    2955          CALL  CRLF
2DDB 0E10      2960          MVI   C,16
2DDD 215331    2965          LXI   H,HDR12
2DE0 CD5E30    2970          CALL  PRINT
2DE3 CD6830    2975          CALL  CRLF
2DE6 2AA832    2980          LHLD  STTOP
2DE9 22B032    2985          SHLD  TOP
2DEC 3E02      2990          MVI   A,2
2DEE 329732    2995          STA   MARK
2DF1 CD2F2E    3000          CALL  FNDSY
2DF4 CD6830    3005          CALL  CRLF
2DF7 0E0F      3010          MVI   C,15
2DF9 216331    3015          LXI   H,HDR13
2DFC CD5E30    3020          CALL  PRINT
2DFF CD6830    3025          CALL  CRLF
2E02 2AAA32    3030          LHLD  ARTOP
2E05 22B032    3035          SHLD  TOP
2E08 3E03      3040          MVI   A,3
2E0A 329732    3045          STA   MARK
2E0D CD2F2E    3050          CALL  FNDSY
2E10 CD6830    3055          CALL  CRLF
2E13 0E0C      3060          MVI   C,12
2E15 217231    3065          LXI   H,HDR14
2E18 CD5E30    3070          CALL  PRINT
2E1B CD6830    3075          CALL  CRLF
2E1E 2AAC32    3080          LHLD  NMTOP
2E21 22B032    3085          SHLD  TOP
2E24 3E04      3090          MVI   A,4
2E26 329732    3095          STA   MARK
2E29 CD2F2E    3000          CALL  FNDSY
2E2C C32820    3105          JMP   EXIT
               3110 ;
               3115 ; THIS SECTION USES THE CENTENTS OF 'TOP'
               3120 ; AND FOLLOWS THE LINKS FOR THE APPROPRIATE
               3125 ; LIST. AT EACH ELEMENT IT LOADS 'HOLDT'
               3130 ; WITH THE LINE FOR THE LINE NUMBER
               3135 ; REFERENCES AND FOLLOWS THAT LIST, PRINTING
               3140 ; THE NUMBERS AS IT GOES.
               3145 ;
2E2F 2AB032    3150 FNDSY:   LHLD  TOP
2E32 3EBB      3155          MVI   A,0BBH
2E34 BC        3160          CMP   H
2E35 C24B2E    3165          JNZ   FNDS1
2E38 BD        3170          CMP   L
2E39 C24B2E    3175          JNZ   FNDS1
2E3C 0E09      3180          MVI   C,9
2E3E 217E31    3185          LXI   H,HDR15
2E41 CD6830    3190          CALL  CRLF
2E44 CD5E30    3195          CALL  PRINT
2E47 CD6830    3200          CALL  CRLF
2E4A C9        3205          RET
2E4B CD2130    3210 FNDS1:   CALL  BLKLN
2E4E 2AB032    3215          LHLD  TOP
2E51 5E        3220          MOV   E,M
2E52 23        3225          INX   H
2E53 56        3230          MOV   D,M
2E54 2B        3235          DCX   H
2E55 EB        3240          XCHG
2E56 229C32    3245          SHLD  NEXT
2E59 3A9732    3250          LDA   MARK
2E5C FE04      3255          CPI   4
2E5E C28B2E    3260          JNZ   FNDS3
2E61 1B        3265          DCX   D
2E62 1A        3270          LDAX  D
2E63 32A432    3275          STA   VALHI
2E66 1B        3280          DCX   D
2E67 1A        3285          LDAX  D
2E68 32A532    3290          STA   VALLO
2E6B CD842F    3295          CALL  CONVR
2E6E 214533    3300          LXI   H,PLINE
2E71 EB        3305          XCHG
2E72 21B432    3310          LXI   H,NUMBR
2E75 0E05      3315          MVI   C,5
2E77 7E        3320 FNDS2:   MOV   A,M
2E78 12        3325          STAX  D
2E79 23        3330          INX   H
2E7A 13        3335          INX   D
2E7B 0D        3340          DCR   C
```

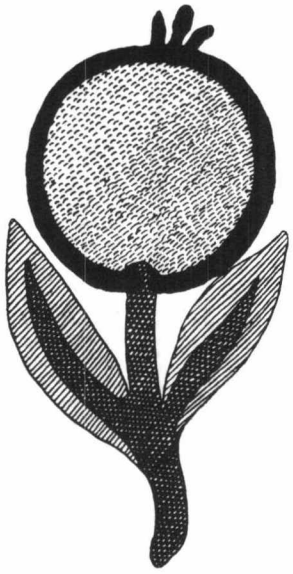
```
2E7C C2772E    3345          JNZ   FNDS2
2E7F 2AB032    3350          LHLD  TOP
2E82 EB        3355          XCHG
2E83 1B        3360          DCX   D
2E84 1B        3365          DCX   D
2E85 CDDB2E    3370          CALL  LSTNM
2E88 C3C12E    3375          JMP   FNDS7
2E8B 214533    3380 FNDS3:   LXI   H,PLINE
2E8E 1B        3385          DCX   D
2E8F 1A        3390          LDAX  D
2E90 77        3395          MOV   M,A
2E91 23        3400          INX   H
2E92 1B        3405          DCX   D
2E93 1A        3410          LDAX  D
2E94 FE20      3415          CPI   20H
2E96 CA9B2E    3420          JZ    FNDS4
2E99 77        3425          MOV   M,A
2E9A 23        3430          INX   H
2E9B 3A9732    3435 FNDS4:   LDA   MARK
2E9E FE02      3440          CPI   2
2EA0 C2AC2E    3445          JNZ   FNDS5
2EA3 3E24      3450          MVI   A,24H
2EA5 77        3455          MOV   M,A
2EA6 CDDB2E    3460          CALL  LSTNM
2EA9 C3C12E    3465          JMP   FNDS7
2EAC FE03      3470 FNDS5:   CPI   3
2EAE C2BE2E    3475          JNZ   FNDS6
2EB1 3E28      3480          MVI   A,28H
2EB3 77        3485          MOV   M,A
2EB4 23        3490          INX   H
2EB5 3E29      3495          MVI   A,29H
2EB7 77        3500          MOV   M,A
2EB8 CDDB2E    3505          CALL  LSTNM
2EBB C3C12E    3510          JMP   FNDS7
2EBE CDDB2E    3515 FNDS6:   CALL  LSTNM
2EC1 2A9C32    3520 FNDS7:   LHLD  NEXT
2EC4 3EBB      3525          MVI   A,0BBH
2EC6 BC        3530          CMP   H
2EC7 C2CF2E    3535          JNZ   FNDS8
2ECA BD        3540          CMP   L
2ECB C2CF2E    3545          JNZ   FNDS8
2ECE C9        3550          RET
2ECF CD6830    3555 FNDS8:   CALL  CRLF
2ED2 2A9C32    3560          LHLD  NEXT
2ED5 22B032    3565          SHLD  TOP
2ED8 C34B2E    3570          JMP   FNDS1
               3575 ;
               3580 ; FOLLOWS EACH LINE NUMBER LIST AND PRINTS
               3585 ; THE FORMATED NUMBERS.
               3590 ;
2EDB 13        3595 LSTNM:   INX   D
2EDC 13        3600          INX   D
2EDD 13        3605          INX   D
2EDE 13        3600          INX   D
2EDF 13        3615          INX   D
2EE0 13        3620          INX   D
2EE1 13        3625          INX   D
2EE2 13        3630          INX   D
2EE3 13        3635          INX   D
2EE4 13        3640          INX   D
2EE5 AF        3645 LSTN1:   XRA   A
2EE6 329432    3650          STA   TAB
2EE9 3E08      3655          MVI   A,8
2EEB 329232    3660          STA   LINCT
2EEE 3E0C      3665          MVI   A,12
2EF0 329332    3670          STA   CHRCT
2EF3 215133    3675          LXI   H,PLINE+12
2EF6 22B232    3680          SHLD  HOLDT
2EF9 1B        3690 LSTN2:   DCX   D
2EFA 1A        3695          LDAX  D
2EFB 32A432    3700          STA   VALHI
2EFE 1B        3705          DCX   D
2EFF 1A        3710          LDAX  D
2F00 32A532    3715          STA   VALLO
2F03 13        3720          INX   D
2F04 13        3725          INX   D
2F05 1A        3730          LDAX  D
2F06 329832    3735          STA   LAST
2F09 13        3740          INX   D
2F0A 1A        3745          LDAX  D
2F0B 329932    3750          STA   LAST+1
2F0E CD842F    3755          CALL  CONVR
2F11 2AB232    3760          LHLD  HOLDT
2F14 EB        3765          XCHG
2F15 21B432    3770          LXI   H,NUMBR
2F18 0E05      3775          MVI   C,5
2F1A 7E        3780 LSTN3:   MOV   A,M
2F1B 12        3785          STAX  D
2F1C 23        3790          INX   H
2F1D 13        3795          INX   D
2F1E 0D        3800          DCR   C
2F1F C21A2F    3805          JNZ   LSTN3
2F22 13        3810          INX   D
2F23 13        3815          INX   D
2F24 EB        3820          XCHG
2F25 22B232    3825          SHLD  HOLDT
2F28 219332    3830          LXI   H,CHRCT
2F2B 3E07      3835          MVI   A,7
2F2D 86        3840          ADD   M
2F2E 329332    3845          STA   CHRCT
2F31 3A9232    3850          LDA   LINCT
2F34 3D        3855          DCR   A
2F35 CA4E2F    3860          JZ    LSTN4
2F38 329232    3865          STA   LINCT
2F3B 2A9832    3870          LHLD  LAST
2F3E EB        3875          XCHG
2F3F 3ECC      3880          MVI   A,0CCH
2F41 BA        3885          CMP   D
2F42 C2F92E    3890          JNZ   LSTN2
2F45 BB        3895          CMP   E
2F46 C2F92E    3900          JNZ   LSTN2
2F49 3E01      3905          MVI   A,1
2F4B 329432    3910          STA   TAB
```

```
2F4E 3A9332      3915 LSTN4: LDA   CHRCT
2F51 4F          3920        MOV   C,A
2F52 214533      3925        LXI   H,PLINE
2F55 CD5E30      3930        CALL  PRINT
2F58 CD6830      3935        CALL  CRLF
2F5B 3A9432      3940        LDA   TAB
2F5E FE01        3945        CPI   1
2F60 C26A2F      3950        JNZ   LSTN6
2F63 2A9C32      3955 LSTN5: LHLD  NEXT
2F66 22B032      3960        SHLD  TOP
2F69 C9          3965        RET
2F6A 2A9832      3970 LSTN6: LHLD  LAST
2F6D 3ECC        3975        MVI   A,0CCH
2F6F BC          3880        CMP   H
2F70 C27A2F      3985        JNZ   LSTN7
2F73 BD          3990        CMP   L
2F74 C27A2F      3995        JNZ   LSTN7
2F77 C3632F      4000        JMP   LSTN5
2F7A CD2130      4005 LSTN7: CALL  BLKLN
2F7D 2A9832      4010        LHLD  LAST
2F80 EB          4015        XCHG
2F81 C3E52E      4020        JMP   LSTN1
                 4025 ;
                 4030 ; CONVERTS EACH NUMBER IN THE LINE NUMBER
                 4035 ; LISTS FROM HEX TO DECIMAL (ASCII).  THIS IS
                 4040 ; DONE BY INSPECTING EACH BIT OF THE STORED
                 4045 ; LINE NUMBER BEGINNING WITH THE HIGH ORDER
                 4050 ; BIT.  IF THE BIT IS ON, THE APPROPRIATE VALUE
                 4055 ; (5 DIGITS) FROM THE DATA CALLED 'BASE:' ARE
                 4060 ; ADDED DECIMALLY, ONE DIGIT AT A TIME, INTO
                 4065 ; AN ACCUMULATOR AREA CALLED 'NUMBR:'.
                 4070 ;
2F84 0E05        4075 CONVR: MVI   C,5
2F86 AF          4080        XRA   A
2F87 21B432      4085        LXI   H,NUMBR
2F8A 77          4090 CONV1: MOV   M,A
2F8B 23          4095        INX   H
2F8C 0D          4100        DCR   C
2F8D C28A2F      4105        JNZ   CONV1
2F90 AF          4110        XRA   A
2F91 329432      4115        STA   TAB
2F94 210833      4120        LXI   H,BASE+79
2F97 EB          4125        XCHG
2F98 3AA432      4130        LDA   VALHI
2F9B 47          4135 CONV2: MOV   B,A
2F9C 0E08        4140        MVI   C,8
2F9E 78          4145 CONV3: MOV   A,B
2F9F E680        4150        ANI   80H
2FA1 C2AC2F      4155        JNZ   CONV4
2FA4 1B          4160        DCX   D
2FA5 1B          4165        DCX   D
2FA6 1B          4170        DCX   D
2FA7 1B          4175        DCX   D
2FA8 1B          4180        DCX   D
2FA9 C3DD2F      4185        JMP   CONV8
2FAC 21B832      4190 CONV4: LXI   H,NUMBR+4
2FAF AF          4195        XRA   A
2FB0 329532      4200        STA   CARRY
2FB3 3E05        4205        MVI   A,5
2FB5 329632      4210 CONV5: STA   COUNT
2FB8 1A          4215        LDAX  D
2FB9 86          4220        ADD   M
2FBA 77          4225        MOV   M,A
2FBB 3A9532      4230        LDA   CARRY
2FBE 86          4235        ADD   M
2FBF FE0A        4240        CPI   0AH
2FC1 CACC2F      4245        JZ    CONV6
2FC4 D2CC2F      4250        JNC   CONV6
2FC7 77          4255        MOV   M,A
2FC8 AF          4260        XRA   A
2FC9 C3D12F      4265        JMP   CONV7
2FCC D60A        4270 CONV6: SUI   0AH
2FCE 77          4275        MOV   M,A
2FCF 3E01        4280        MVI   A,1
2FD1 329532      4285 CONV7: STA   CARRY
2FD4 2B          4290        DCX   H
2FD5 1B          4295        DCX   D
2FD6 3A9632      4300        LDA   COUNT
2FD9 3D          4305        DCR   A
2FDA C2B52F      4310        JNZ   CONV5
2FDD 0D          4315 CONV8: DCR   C
2FDE CAE72F      4320        JZ    CONV9
2FE1 78          4325        MOV   A,B
2FE2 07          4330        RLC
2FE3 47          4335        MOV   B,A
2FE4 C39E2F      4340        JMP   CONV3
2FE7 3A9432      4345 CONV9: LDA   TAB
2FEA FE01        4350        CPI   1
2FEC CAFA2F      4355        JZ    CON10
2FEF 3E01        4360        MVI   A,1
2FF1 329432      4365        STA   TAB
2FF4 3AA532      4370        LDA   VALLO
2FF7 C39B2F      4375        JMP   CONV2
2FFA 0E05        4380 CON10: MVI   C,5
2FFC 21B432      4385        LXI   H,NUMBR
2FFF 7E          4390 CON11: MOV   A,M
3000 C630        4395        ADI   30H
3002 77          4400        MOV   M,A
3003 3A9432      4405        LDA   TAB
3006 FE01        4410        CPI   1
3008 C21B30      4415        JNZ   CON13
300B 3E30        4420        MVI   A,30H
300D BE          4425        CMP   M
300E C21730      4430        JNZ   CON12
3011 3E20        4435        MVI   A,20H
3013 77          4440        MOV   M,A
3014 C31B30      4445        JMP   CON13
3017 AF          4450 CON12: XRA   A
3018 329432      4455        STA   TAB
301B 23          4460 CON13: INX   H
301C 0D          4465        DCR   C
301D C2FF2F      4470        JNZ   CON11
3020 C9          4475        RET
                 4480 ;
                 4485 ; FILLS THE PRINT LINE WITH SPACES (20H).
                 4490 ;
3021 0E48        4495 BLKLN: MVI   C,72
3023 214533      4500        LXI   H,PLINE
3026 3E20        4505        MVI   A,20H
3028 77          4510 BLKL1: MOV   M,A
3029 23          4515        INX   H
302A 0D          4520        DCR   C
302B C22830      4525        JNZ   BLKL1
302E C9          4530        RET
                 4535 ;
                 4540 ; THIS ROUTINE READS ONE 256 BYTE BLOCK
                 4545 ; OF DATA FROM THE DISK INTO THE AREA
                 4550 ; CALLED 'DATA'.
                 4555 ;
302F 3E01        4560 RDBLK: MVI   A,1
3031 0601        4565        MVI   B,1
3033 219131      4570        LXI   H,DRIVE
3036 4E          4575        MOV   C,M
3037 119231      4580        LXI   D,DATA
303A 2A0933      4585        LHLD  DIR
303D 23          4590        INX   H
303E 220933      4595        SHLD  DIR
3041 2B          4600        DCX   H
3042 CD2220      4605        CALL  RDDSK
3045 D25630      4610        JNC   RDBL1
3048 0E0E        4615        MVI   C,14
304A 213331      4620        LXI   H,HDR10
304D CD5E30      4625        CALL  PRINT
3050 CD6830      4630        CALL  CRLF
3053 C32820      4635        JMP   EXIT
3056 AF          4640 RDBL1: XRA   A
3057 32A032      4645        STA   BLKPT
305A 119231      4650        LXI   D,DATA
305D C9          4655        RET
                 4660 ;
                 4665 ; PRINT STRING AT ADDRESS IN REG H/L FOR
                 4670 ; NUMBER OF CHARACTERS IN REG C.
                 4675 ;
305E 46          4680 PRINT: MOV   B,M
305F CD0D20      4685        CALL  COUT
3062 23          4690        INX   H
3063 0D          4695        DCR   C
3064 C25E30      4700        JNZ   PRINT
3067 C9          4705        RET
                 4710 ;
                 4715 ; INSERT CR AND LF
                 4720 ;
3068 060D        4725 CRLF:  MVI   B,0DH
306A CD0D20      4730        CALL  COUT
306D 060A        4735        MVI   B,0AH
306F CD0D20      4740        CALL  COUT
3072 C9          4745        RET
                 4750 ;
                 4755 ; CHECK BLOCK ROUTINE
                 4760 ;
3073 13          4765 NXTCR: INX   D
3074 21A032      4770        LXI   H,BLKPT
3077 34          4775        INR   M
3078 CC2F30      4780        CZ    RDBLK
307B 62          4785        MOV   H,D
307C 6B          4790        MOV   L,E
307D 3E0D        4795        MVI   A,0DH
307F BE          4800        CMP   M
3080 C28630      4805        JNZ   NXTC1
3083 32A132      4810        STA   LNTAB
3086 C9          4815 NXTC1: RET
                 4820 ;
                 4825 ; HEADERS
                 4830 ;
3087 4E4F5254484835 HDR1:  DB    'NORTHSTAR BASIC PROGRAM SUPPORT'
30A6 4E414D5204840 HDR2:  DB    'NAME OF PROGRAM: '
30B7 4F4E2044524845 HDR3:  DB    'ON DRIVE #: '
30C3 4E4F2050524850 HDR4:  DB    'NO PROGRAM WITH THAT NAME'
30DC 46494C45204855 HDR5:  DB    'FILE IS NOT TYPE 2'
30EE 5441424C454860 HDR6:  DB    'TABLE FULL - PARTIAL LIST SUPPLIED'
3110 4E414D45204865 HDR7:  DB    'NAME IS TOO LONG'
3120 494E56414C4870 HDR8:  DB    'INVALID UNIT'
312C 4E4F20524F4875 HDR9:  DB    'NO ROOM'
3133 415247554D4880 HDR10: DB    'ARGUMENT ERROR'
3141 53594D424F4885 HDR11: DB    'SYMBOLIC VARIABLES'
3153 535452494E4890 HDR12: DB    'STRING VARIABLES'
3163 41525241594895 HDR13: DB    'ARRAY VARIABLES'
3172 4C494E45204900 HDR14: DB    'LINE NUMBERS'
317E 4E554C4C204905 HDR15: DB    'NULL LIST'
3187             4910 PNAME: DS    10
3191             4915 DRIVE: DS    1
3192             4920 DATA:  DS    256
3292             4925 LINCT: DS    1
3293             4930 CHRCT: DS    1
3294             4935 TAB:   DS    1
3295             4940 CARRY: DS    1
3296             4945 COUNT: DS    1
3297             4950 MARK:  DS    1
3298             4955 LAST:  DS    2
329A             4960 PRSNT: DS    2
329C             4965 NEXT:  DS    2
329E             4970 POINT: DS    2
32A0             4975 BLKPT: DS    1
32A1             4980 LNTAB: DS    1
32A2             4985 LOLIN: DS    1
32A3             4990 HILIN: DS    1
32A4             4995 VALHI: DS    1
32A5             5000 VALLO: DS    1
32A6             5005 SYTOP: DS    2
32A8             5010 STTOP: DS    2
32AA             5015 ARTOP: DS    2
32AC             5020 NMTOP: DS    2
32AE             5025 AVAIL: DS    2
32B0             5030 TOP:   DS    2
32B2             5035 HOLDT: DS    2
32B4             5040 NUMBR: DS    5
```

```
32B9 0000000015045 BASE:  DB    0,0,0,0,1,0,0,0,0,2,0,0,0,0,4,0,0,0,0
32CC 0800000015050        DB    8,0,0,0,1,6,0,0,0,3,2,0,0,0,6,4,0,0,1
32DF 0208000025055        DB    2,8,0,0,2,5,6,0,0,5,1,2,0,1,0,2,4,0,2
32F2 0004080045060        DB    0,4,8,0,4,0,9,6,0,8,1,9,2,1,6,3,8,4,3
3305 02070608   5065      DB    2,7,6,8
3309            5070 DIR: DS    8
3311            5075 STACK: DS  50
3343            5080 ADDR: DS   2
3345            5085 PLINE: DS  72
338D            5090 TABLE: DS  12288
638D            5095      END   2A00H


DH 2A00)33FF
      ?
>DH 2A00-33FFF?
>DH 2A00-33FF
2A00  21 42 33 F9  0E 1F 21 87  30 CD 5E 30  CD 68 30 0E
2A10  11 21 A6 30  CD 5E 30 0E  0A 21 87 31  36 20 23 0D
2A20  C2 1C 2A 0E  09 21 87 31  CD 10 20 32  B2 32 FE 0D
2A30  CA 6F 2A FE  5F C2 54 2A  3E 09 B9 CA  28 2A 3A B2
2A40  32 2B 0C 47  CD 0D 20 06  20 CD 0D 20  06 5F CD 0D
2A50  20 C3 28 2A  77 47 CD 0D  20 23 0D C2  28 2A CD 68
2A60  30 0E 10 21  10 31 CD 5E  30 CD 68 30  C3 0F 2A 0E
2A70  0C 21 B7 30  CD 68 30 CD  5E 30 21 91  31 CD 10 20
2A80  47 CD 0D 20  FE 31 CA A4  2A FE 32 CA  A4 2A FE 33
2A90  CA A4 2A CD  68 30 0E 0C  21 20 31 CD  5E 30 CD 68
2AA0  30 C3 6F 2A  E6 0F 77 CD  10 20 47 CD  0D 20 FE 0D
2AB0  C2 93 2A 06  0A CD 0D 20  3A 91 31 21  87 31 37 3F
2AC0  CD 1C 20 D2  D4 2A 0E 19  21 C3 30 CD  5E 30 CD 68
2AD0  30 C3 0F 2A  06 00 0E 04  09 3E 02 BE  CA ED 2A 0E
2AE0  12 21 DC 30  CD 5E 30 CD  68 30 C3 0F  2A 2B 2B 2B
2AF0  2B 0E 08 11  09 33 7E 12  23 13 0D C2  F6 2A 26 00
2B00  2E 00 22 43  33 06 00 0E  04 11 8F 33  2A 43 33 23
2B10  3E 0C BC CA  23 2B 22 43  33 62 6B 09  EB 73 23 72
2B20  C3 0C 2B 3E  FF 23 77 26  BB 2E BB 22  A6 32 22 A8
2B30  32 22 AA 32  22 AC 32 21  8F 33 22 AE  32 CD 2F 30
2B40  AF 32 97 32  3C EB BE CA  BC 2D EB CD  73 30 1A 32
2B50  A2 32 CD 73  30 1A 32 A3  32 AF 32 A1  32 CD 73 30
2B60  62 6B 3E 20  BE CA B6 2B  3E 8F BE C2  73 2B 3E 01
2B70  32 97 32 3E  22 BE C2 90  2B 3A 97 32  FE 02 C2 88
2B80  2B AF 32 97  32 C3 B6 2B  3E 02 32 97  32 C3 B6 2B
2B90  3A 97 32 FE  00 C2 B6 2B  3E 9A BE C2  A4 2B CD C8
2BA0  2B C3 B6 2B  3E 40 BE D2  B6 2B 3E 5A  BE DA B6 2B
2BB0  CD EA 2B C3  B9 2B CD 73  30 3E 0D 21  A1 32 BE C2
2BC0  60 2B CD 73  30 C3 40 2B  CD 73 30 1A  32 A5 32 CD
2BD0  73 30 1A 32  A4 32 AF 32  A1 32 2A AC  32 22 B0 32
2BE0  CD 7A 2C 2A  B0 32 22 AC  32 C9 3E 20  32 A5 32 32
2BF0  A4 32 1A 32  A4 32 CD 73  30 3E 0D 21  A1 32 BE C2
2C00  08 2C CD 4A  2C C3 49 2C  62 6B 3E 2F  BE D2 2C 2C
2C10  3E 39 BE DA  2C 2C 1A 32  A5 32 CD 73  30 3E 0D 21
2C20  A1 32 BE C2  2C 2C CD 4A  2C C3 49 2C  62 6B 3E 24
2C30  BE C2 3A 2C  CD 5A 2C C3  49 2C 3E E0  BE C2 46 2C
2C40  CD 6A 2C C3  49 2C CD 4A  2C C9 2A A6  32 22 B0 32
2C50  CD 7A 2C 2A  B0 32 22 A6  32 C9 2A A8  32 22 B0 32
2C60  CD 7A 2C 2A  B0 32 22 A8  32 C9 2A AA  32 22 B0 32
2C70  CD 7A 2C 2A  B0 32 22 AA  32 C9 22 9C  32 EB 22 9E
2C80  32 3E BB 32  98 32 32 99  32 2A AE 32  22 9A 32 2B
2C90  3A A4 32 77  2B 3A A5 32  77 23 23 3E  BB 77 23 77
2CA0  23 23 23 7E  32 AE 32 3E  CC 77 23 7E  32 AF 32 3E
2CB0  CC 77 2A 9C  32 3E BB BC  C2 C9 2C BD  C2 C9 2C 2A
2CC0  9A 32 22 B0  32 CD 75 2D  C9 2B 56 3A  A4 32 BA CA
2CD0  32 2D D2 FB  2C 2A 9A 32  3A 9C 32 77  23 3A 9D 32
2CE0  77 2A 98 32  3E BB BC C2  EE 2C BD CA  BF 2C 3A 9A
2CF0  32 77 23 3A  9B 32 77 CD  75 2D C9 3E  BB 2A 9C 32
2D00  BC C2 18 2D  BD C2 18 2D  2A 98 32 3A  9A 32 77 23
2D10  3A 9B 32 77  CD 75 2D C9  2A 9C 32 22  98 32 5E 23
2D20  56 EB 22 9C  32 3E BB BC  C2 C9 2C BD  C2 C9 2C C3
2D30  08 2D 2B 56  3A A5 32 BE  CA 41 2D D2  FB 2C C3 D5
2D40  2C 2A AE 32  2B 2B 2B 3A  AF 32 77 2B  3A AE 32 77
2D50  2B 2B 2B EB  2A AE 32 2B  2B 2B 2B 22  AE 32 EB 3A
2D60  AF 32 77 2B  3A AE 32 77  22 AE 32 2A  9C 32 22 9A
2D70  32 CD 75 2D  C9 2A 9A 32  23 23 23 23  3E CC BE C2
2D80  B5 2D 23 BE  C2 B4 2D 2B  3A AE 32 77  23 3A AF 32
2D90  77 2A AE 32  2B 2B 3A A2  32 77 23 3A  A3 32 77 23
2DA0  7E 32 AE 32  3E CC 77 23  7E 32 AF 32  3E CC 77 2A
2DB0  9E 32 EB C9  2B 5E 23 56  EB C3 7C 2D  0E 12 21 41
2DC0  31 CD 68 30  CD 5E 30 CD  68 30 2A A6  32 22 B0 32
2DD0  3E 01 32 97  32 CD 2F 2E  CD 68 30 0E  10 21 53 31
2DE0  CD 5E 30 CD  68 30 2A A8  32 22 B0 32  3E 02 32 97
2DF0  32 CD 2F 2E  CD 68 30 0E  0F 21 63 31  CD 5E 30 CD
2E00  68 30 2A AA  32 22 B0 32  3E 03 32 97  32 CD 2F 2E
2E10  CD 68 30 0E  0C 21 72 31  CD 5E 30 CD  68 30 2A AC
2E20  32 22 B0 32  3E 04 32 97  32 CD 2F 2E  C3 28 20 2A
2E30  B0 32 3E BB  BC C2 4B 2E  BD C2 4B 2E  0E 09 21 7E
2E40  31 CD 68 30  CD 5E 30 CD  68 30 C9 CD  21 30 2A B0
2E50  32 5E 23 56  2B EB 22 9C  32 3A 97 32  FE 04 C2 8B
2E60  2E 1B 1A 32  A4 32 1B 1A  32 A5 32 CD  84 2F 21 45
2E70  33 EB 21 B4  32 0E 05 7E  12 23 13 0D  C2 77 2E 2A
2E80  B0 32 EB 1B  1B CD DB 2E  C3 C1 2E 21  45 33 1B 1A
2E90  77 23 1B 1A  FE 20 CA 9B  2E 77 23 3A  97 32 FE 02
2EA0  C2 AC 2E 3E  24 77 CD DB  2E C3 C1 2E  FE 03 C2 BE
2EB0  2E 3E 28 77  23 3E 29 77  CD DB 2E C3  C1 2E CD DB
2EC0  2E 2A 9C 32  3E BB BC C2  CF 2E BD C2  CF 2E C9 CD
2ED0  68 30 2A 9C  32 22 B0 32  C3 4B 2E 13  13 13 13 13
2EE0  13 13 13 13  13 AF 32 94  32 3E 08 32  92 32 3E 0C
2EF0  32 93 32 21  51 33 22 B2  32 1B 1A 32  A4 32 1B 1A
2F00  32 A5 32 13  13 1A 32 98  32 13 1A 32  99 32 CD 84
2F10  2F 2A B2 32  EB 21 B4 32  0E 05 7E 12  23 13 0D C2
2F20  1A 2F 13 13  EB 22 B2 32  21 93 32 3E  07 86 32 93
2F30  32 3A 92 32  3D CA 4E 2F  32 92 32 2A  98 32 EB 3E
2F40  CC BA C2 F9  2E BB C2 F9  2E 3E 01 32  94 32 3A 93
2F50  32 4F 21 45  33 CD 5E 30  CD 68 30 3A  94 32 FE 01
2F60  C2 6A 2F 2A  9C 32 22 B0  32 C9 2A 98  32 3E CC BC
2F70  C2 7A 2F BD  C2 7A 2F C3  63 2F CD 21  30 2A 98 32
2F80  EB C3 E5 2E  0E 05 AF 21  B4 32 77 23  0D C2 8A 2F
2F90  AF 32 94 32  21 08 33 EB  3A A4 32 47  0E 08 78 E6
2FA0  80 C2 AC 2F  1B 1B 1B 1B  1B C3 DD 2F  21 B8 32 AF
2FB0  32 95 32 3E  05 32 96 32  1A 86 77 3A  95 32 86 FE
2FC0  0A CA CC 2F  D2 CC 2F 77  AF C3 D1 2F  D6 0A 77 3E
2FD0  01 32 95 32  2B 1B 3A 96  32 3D C2 B5  2F 0D CA E7
2FE0  2F 78 07 47  C3 9E 2F 3A  94 32 FE 01  CA FA 2F 3E
2FF0  01 32 94 32  3A A5 32 C3  9B 2F 0E 05  21 B4 32 7E
3000  C6 30 77 3A  94 32 FE 01  C2 1B 30 3E  30 BE C2 17
3010  30 3E 20 77  C3 1B 30 AF  32 94 32 23  0D C2 FF 2F
3020  C9 0E 48 21  45 33 3E 20  77 23 0D C2  28 30 C9 3E
3030  01 06 01 21  91 31 4E 11  92 31 2A 09  33 23 22 09
3040  33 2B CD 22  20 D2 56 30  0E 0E 21 33  31 CD 5E 30
3050  CD 68 30 C3  28 20 AF 32  A0 32 11 92  31 C9 46 CD
3060  0D 20 23 0D  C2 5E 30 C9  06 0D CD 0D  20 06 0A CD
3070  0D 20 C9 13  21 A0 32 34  CC 2F 30 62  6B 3E 0D BE
3080  C2 86 30 32  A1 32 C9 4E  4F 52 54 48  53 54 41 52
3090  20 42 41 53  49 43 20 50  52 4F 47 52  41 4D 20 53
30A0  55 50 50 4F  52 54 4E 41  4D 45 20 4F  46 20 50 52
30B0  4F 47 52 41  4D 3A 20 4F  4E 20 44 52  49 56 45 20
30C0  23 3A 20 4E  4F 20 50 52  4F 47 52 41  4D 20 57 49
30D0  54 48 20 54  48 41 54 20  4E 41 4D 45  46 49 4C 45
30E0  20 49 53 20  4E 4F 54 20  54 59 50 45  20 32 54 41
30F0  42 4C 45 20  46 55 4C 4C  20 2D 20 50  41 52 54 49
3100  41 4C 20 4C  49 53 54 20  53 55 50 50  4C 49 45 44
3110  4E 41 4D 45  20 49 53 20  54 4F 4F 20  4C 4F 4E 47
3120  49 4E 56 41  4C 49 44 20  55 4E 49 54  4E 4F 20 52
3130  4F 4F 4D 41  52 47 55 4D  45 4E 54 20  45 52 52 4F
3140  52 53 59 4D  42 4F 4C 49  43 20 56 41  52 49 41 42
3150  4C 45 53 53  54 52 49 4E  47 20 56 41  52 49 41 42
3160  4C 45 53 41  52 52 41 59  20 56 41 52  49 41 42 4C
3170  45 53 4C 49  4E 45 20 4E  55 4D 42 45  52 53 4E 55
3180  4C 4C 20 4C  49 53 54 54  41 42 4C 45  2B 32 0D 0A
3190  31 37 32 30  20 53 4C 49  4E 4B 3A 20  4C 48 4C 44
31A0  20 20 41 44  44 52 0D 0A  31 37 33 30  20 20 20 20
31B0  20 20 20 20  49 4E 58 20  20 20 48 0D  0A 31 37 33
31C0  32 20 20 20  20 20 20 20  20 4D 56 49  20 20 20 41
31D0  2C 30 43 48  0D 0A 31 37  33 34 20 20  20 20 20 20
31E0  20 20 43 4D  50 20 20 20  48 0D 0A 31  37 33 36 20
31F0  20 20 20 20  20 20 20 4A  5A 20 20 20  20 53 54 4F
3200  50 53 0D 0A  31 37 33 37  20 20 20 20  20 20 20 20
3210  53 48 4C 44  20 20 41 44  44 52 0D 0A  31 37 33 38
3220  20 20 20 20  20 20 20 20  4D 4F 56 20  20 20 48 2C
3230  44 0D 0A 31  37 34 30 20  20 20 20 20  20 20 20 4D
3240  4F 56 20 20  20 4C 2C 45  0D 0A 31 37  35 30 20 20
3250  20 20 20 20  20 20 44 41  44 20 20 20  42 0D 0A 31
3260  37 36 30 20  20 20 20 20  20 20 20 58  43 48 47 0D
3270  0A 31 37 36  32 20 20 20  20 20 20 20  20 4D 4F 56
3280  20 20 20 4D  2C 45 0D 0A  31 37 36 34  20 20 20 20
3290  20 20 20 20  49 4E 58 20  20 20 48 0D  0A 31 37 36
32A0  36 20 20 20  20 20 20 20  20 4D 4F 56  20 20 20 4D
32B0  2C 44 0D 0A  31 37 36 38  20 00 00 00  00 01 00 00
32C0  00 00 02 00  00 00 00 04  00 00 00 00  08 00 00 00
32D0  01 06 00 00  00 03 02 00  00 00 06 04  00 00 01 02
32E0  08 00 00 02  05 06 00 00  05 01 02 00  01 00 02 04
32F0  00 02 00 04  08 00 04 00  09 06 00 08  01 09 02 01
3300  06 03 08 04  03 02 07 06  08 4F 56 20  20 20 4D 2C
3310  41 0D 0A 31  37 39 30 20  20 20 20 20  20 20 20 4D
3320  56 49 20 20  20 48 2C 30  42 42 48 0D  0A 31 38 30
3330  30 20 20 20  20 20 20 20  20 4D 56 49  20 20 20 4C
3340  2C 30 42 42  48 0D 0A 31  38 31 30 20  20 20 20 20
3350  20 20 20 53  48 4C 44 20  20 53 59 54  4F 50 0D 0A
3360  31 38 32 30  20 20 20 20  20 20 20 20  53 48 4C 44
3370  20 20 53 54  54 4F 50 0D  0A 31 38 33  30 20 20 20
3380  20 20 20 20  20 53 48 4C  44 20 20 41  52 54 4F 50
3390  0D 0A 31 38  34 30 20 20  20 20 20 20  20 20 53 48
33A0  4C 44 20 20  4E 4D 54 4F  50 0D 0A 31  38 35 30 20
33B0  20 20 20 20  20 20 20 4C  58 49 20 20  20 48 2C 54
33C0  41 42 4C 45  2B 32 0D 0A  31 38 36 30  20 20 20 20
33D0  20 20 20 20  53 48 4C 44  20 20 41 56  41 49 4C 0D
33E0  0A 31 39 30  30 20 20 20  20 20 20 20  20 43 41 4C
33F0  4C 20 20 52  44 42 4C 4B  0D 0A 31 39  31 30 20 3B
```

```
SYMBOLIC VARIABLES
A               310   340   370   560   705
B               330   340   360   560   705
C               350   360   370   560   705
D               480   480   530   532   560
E               480   480   480   480   480   530   534   560
F               480   480   480   532   534   560
I               285   286   565   580   600   630
J               286   286   570   580   600   620
P               470   520   590   590   610   610   640   650
                660   680
T               400   410   690   750
X               286   430   440   445   480   480   705   710

STRING VARIABLES
A$               50    80   820   830

ARRAY VARIABLES
Z()             305   560   560   560   560   560   560   580
                580
```

```
LINE NUMBERS
285    80
290   830
330   340
350   360   370
410   460   550
450   440
470   445
540   530   532   534
560   538
620   580   600
680   650
690   670
750   520   640
820   740
*
```



Some Useful Subroutines for a Z-80 Assembler

BY JOHN W. CARTER
Computer Systems Technology
Memphis State University
Memphis, TN 38152

Language Overview

Many new users of microcomputer systems purchase those systems with one predominant software objective: BASIC programming. Such a high-level language is certainly convenient and relatively powerful. However, BASIC and other high-level languages have a distinct disadvantage in terms of execution time and memory usage. Two major types of high-level languages are now in common use: compilers and interpreters.

Compilers are programs which convert an entire high-level source program into object (binary) code, and generate an object program which is later executed without aid of the compiler. In generating the object program, the compiler generates and calls object subroutines when processing is required. Repetition of these subroutines makes the object program inefficient in terms of execution time and memory usage. However, this trade-off can be more than offset by the convenience of the high-level language.

Interpreters are programs which convert a high-level source program into object code one line at a time. That is, each line or source code is compiled and executed individually, requiring recompilation for each execution. No object code is preserved. Therefore, in order to execute such a source program, both the source program and the interpreter must reside in the memory space (usually RAM). This procedure is extremely slow and consumes large quantities of memory space. A good example of this is evident in the long looping operations of BASIC interpreters. For example, the execution time of a bubble sort of more than 1000 elements is formidable. *You can* expect to wait as long as an hour for such a sort to execute – depending, of course, on the data structure.

A solution to the memory and execution time inefficiency of compilers and interpreters can be realized in many applications through the use of assembly language programming. Programs can be written in assembly language which would be inappropriate for use in a high-level language; the aforementioned sort, for example. Also, assembly language programming opens up a whole new world of data structures, program structures and hardware accessibility not previously available. Using the assembler, I/O ports, analog and digital converters, real-time-clocks and other useful hardware and software techniques are easily accessed.

The assembler acts like a compiler, since it is one. The primary difference between this compiler and that which translates a high-level language, is that translation takes place at the machine level. Each source statement is translated to a single operation. No binary subroutines are generated (with the exception of macro assembly), giving the programmer total control over the program operation. The assembler resides in memory only during the compilation process. It generates a very efficient object code which can later be executed without the aid of the assembler.

In order to help open up this area of programming to those interested individuals the following subroutines are offered.

Subroutines

When writing assembly language programs the first task involves communication between the program and the user. In order to do so, the peripheral keyboard and display need to be accessed, conversions between binary and ASCII must be performed, and some simple debugging tools would be useful. The following nine subroutines will help to fill such a need. They are a sample of a program library of over 60 subroutines written by this author for use in the assembly language programming course in Memphis State University's Technology Division. Access to these subroutines can keep the beginning programmer from becoming bogged down in the process of program debugging.

These program segments have been coded in Z-80 assembly language for use on the TRS-80 microcomputer. However, these can be easily translated for use on almost any mini or micro.

The following is a summary of the subroutines given:

Subroutine AINPUT. This subroutine will scan the keyboard of the microcomputer for an ASCII character, and upon the execution of a wait loop, places that character in the accumulator and returns. The subroutine also echoes that character to the video display by calling subroutine APRINT prior to the return statement. This subroutine will require modification for use on any machine other than the TRS-80. In order to do so, design the subroutine structure such that it places the ASCII character in the accumulator prior to the return.

Subroutine APRINT. This subroutine is the converse of AINPUT. It takes an ASCII character in the accumulator and prints it on the video display device. This subroutine will also require modification for use on any machine other than the TRS-80.

Subroutine REGDMP. This subroutine provides a labeled hexadecimal dump of registers A, B, C, D, E, F, H, L, IX and IY. The technique used is by no means the most efficient in terms of memory usage, but it is very simple to implement and interpret. It can be placed within a program at any point (or points) and a dump of the registers will occur at that point in execution without affecting the contents of any of the registers. This subroutine is not personallized to the TRS-80.

Subroutine OUT16H. This subroutine provides a hexadecimal dump of the 16-bit contents of the HL register. Any register pair can be dumped by this subroutine. For example, the DE pair can be dumped by executing the following sequence:

```
PUSH    HL      ; save data in HL from destruction
PUSH    pp      ; pp is the register pair
POP     HL      ; put that pair into the HL pair
CALL    OUT16H  ; dump it
POP     HL      ; restore old HL data
```

Subroutine OUT08H. This subroutine provides a hexadecimal dump of the contents of the C register. The contents of any register can be dumped by executing the following sequence:

```
PUSH    BC      ; save data in C from destruction
LD      C,r     ; r is the register to be dumped
CALL    OUT08H  ; dump it
POP     BC      ; restore old C data
```

Subroutine BINASC. This subroutine converts a four-bit binary value in the accumulator to its ASCII hexadecimal representation, and places that ASCII character back into the accumulator. It is used in all printing and register dumping routines.

Subroutine IN08HX. This subroutine waits for and receives two ASCII characters from the keyboard (using AINPUT) and converts those two characters, assumed to be in hexadecimal representation, into the 8-bit binary equivalent. The binary value is placed in the accumulator prior to returning to the calling sequence.

Subroutine ASCBIN. This subroutine is the converse of BINASC. The subroutine takes as ASCII character in the accumulator and, assuming that it is a hexadecimal value, converts it to the equivalent four-bit binary nybble. The resulting nybble is placed back into the accumulator prior to the return.

Subroutine BMLTO8. This subroutine was included to enable the beginning programmer the ability to multiply 8-bit numbers. This subroutine will multiply two 8-bit signed 2's complement operands and produce a 16-bit signed 2's complement product, using Booth's algorithm. In order to provide an efficient execution time, the registers were assigned as follows: The multiplicand and multiplier are placed in the C and D registers, respectively, prior to execution. After the subroutine is called, the most significant byte of the product is in the accumulator, the least significant byte is placed in the D register. The registers can be reassigned to provide more convenient use at the expense of execution time.

Program BMLTST. This program was written to test each of the subroutines. The input routines are used to accept hex operands, the multiplication subroutine then generates the product which is consequently printed. The registers are dumped prior to and after the multiplication routine. An example of the printed output is as follows:

```
FE*FE=
A  B  C  D  E  F  H  L   IX   IY
3D 00 FE FE 00 FF 00 11 4015 FFFF
A  B  C  D  E  F  H  L   IX   IY
00 00 FE 04 00 FF 00 11 4015 FFFF
0004



44*44=
A  B  C  D  E  F  H  L   IX   IY
3D 00 44 44 00 FF 00 04 4015 FFFF
A  B  C  D  E  F  H  L   IX   IY
12 00 44 10 E0 FF 00 04 4015 FFFF
1210
```

The aforementioned subroutine library is available from the author. For information concerning the remainder of the library, send a self-addressed, stamped envelope to the author at 4932 Shifri Avenue, Memphis TN 38117. The complete source list and cassette tape will be made available for a duplication and handling fee of $10. Indicate memory size in any requests (4K, 16K, 32K).

Mr. Carter is an assistant professor of computer systems technology at Memphis State University and is actively involved with the development of micro and mini software, hardware systems and computer-aided instruction. Memphis State University offers a baccalaureate degree in Computer Systems Technology; a curriculum which emphasizes the hardware aspects of computer systems, as well as programming using compilers, interpreters and assemblers.

```
                        01000 ;********** PROGRAM BMLTST **********
                        01010 ;
                        01020 ;J. W.  CARTER              25 JUNE 1979
                        01030 ;
                        01040 ;THE PURPOSE OF THIS PROGRAM IS TO TEST THE SUBROUTINES:
                        01050 ;
                        01060 ;BMLT08  (BOOTH'S ALGORITHM 8-BIT MULTIPLY)
                        01070 ;AINPUT  (INPUT ASCII CHARCTER FROM TRS-80 KEYBOARD)
                        01080 ;APRINT  (PRINT ASCII CHARACTER ON TRS-80 VIDEO DISPLAY)
                        01090 ;REGDMP  (PRINT A DUMP OF Z-80 REGISTERS ON THE DISPLAY)
                        01100 ;OUT16H  (PRINT HEX REPRESENTATION OF A REGISTER PAIR)
                        01110 ;OUT08H  (PRINT HEX REPRESENTATION OF A REGISTER)
                        01120 ;BINASC  (CONVERT BINARY NYBBLE TO ASCII CHARACTER)
                        01130 ;ASCBIN  (CONVERT ASCII CHARACTER TO BINARY NYBBLE)
                        01140 ;IN08HX  (INPUT TWO HEX CHARACTERS FROM TRS-80 KEYBOARD
                        01150 ;         AND CONVERT TO 8-BIT BINARY)
                        01160 ;
                        01170 ;REGISTER DEFINITIONS:
                        01180 ;         A       CHARACTER INPUT/OUTPUT VECTOR
                        01190 ;         HL      PRODUCT OUTPUT VECTOR FROM BMLT08
                        01200 ;         C       MULTIPLICAND INPUT TO BMLT08
                        01210 ;         D       MULTIPLIER INPUT TO BMLT08
                        01220 ;SUBROUTINES USED:
                        01230 ;         IN08HX          APRINT
                        01240 ;         BMLT08          ASCBIN
                        01250 ;         OUT16H          ASCBIN
                        01260 ;
4EA9                    01270           ORG     4EA9H
4EA9 310045             01280 START     LD      SP,4500H
4EAC CDBE4F             01290           CALL    IN08HX          ;A=MULTIPLIER
4EAF 57                 01300           LD      D,A             ;D=MULTIPLIER
4EB0 3E2A               01310           LD      A,2AH           ;ASCII "*"
4EB2 CDE44E             01320           CALL    APRINT          ;PRINT IT ON DISPLAY
4EB5 CDBE4F             01330           CALL    IN08HX          ;A=MULTIPLICAND
4EB8 4F                 01340           LD      C,A             ;C=MULTIPLICAND
4EB9 3E3D               01350           LD      A,3DH           ;ASCII "="
4EBB CDE44E             01360           CALL    APRINT          ;PRINT IT ON DISPLAY
4EBE CD0F4F             01370           CALL    REGDMP          ;DUMP WORKING REGISTERS
4EC1 CDE14F             01380           CALL    BMLT08          ;PRODUCT = A D "PAIR"
4EC4 CD0F4F             01390           CALL    REGDMP          ;DUMP WORKING REGISTERS
4EC7 67                 01400           LD      H,A             ;MS BYTE OF PRODUCT
4EC8 6A                 01410           LD      L,D             ;LS BYTE OF PRODUCT
4EC9 CD924F             01420           CALL    OUT16H          ;PRINT PRODUCT IN HEX
4ECC 3E0D               01430           LD      A,0DH           ;ASCII "CR"
4ECE CDE44E             01440           CALL    APRINT          ;PRINT IT ON DISPLAY
4ED1 C3A94E             01450           JP      START           ;DO ANOTHER CALCULATION
                        01460 ;
                        01470 ;********** SUBROUTINE AINPUT **********
                        01480 ;
                        01490 ;THE PURPOSE OF THIS SUBROUTINE IS TO CALL THE TRS-80
                        01500 ;KEYBOARD STROBE ROUTINE, AND ECHO THE CHARACTER TO THE
                        01510 ;TRS-80 VIDEO DISPLAY, STORING THE CHARACTER IN  REG. A.
                        01520 ;
                        01530 ;REGISTER DEFINITIONS:
                        01540 ;         A       CHARACTER INPUT VECTOR
                        01550 ;SUBROUTINES USED:
                        01560 ;         APRINT
                        01570 ;
4ED4 D5                 01580 AINPUT    PUSH    DE              ;SAVE REGISTERS
4ED5 FDE5               01590           PUSH    IY
4ED7 CD2B00             01600 LPAINP    CALL    2BH             ;STROBE KEYBOARD
4EDA B7                 01610           OR      A               ;TEST CHARACTER
4EDB 28FA               01620           JR      Z,LPAINP        ;IF 0 GO BACK
4EDD CDE44E             01630           CALL    APRINT          ;ECHO CHARACTER
4EE0 FDE1               01640           POP     IY              ;RESTORE REGISTERS
4EE2 D1                 01650           POP     DE
4EE3 C9                 01660           RET
                        01670 ;
                        01680 ;********** SUBROUTINE APRINT **********
                        01690 ;
                        01700 ;THE PURPOSE OF THIS SUBROUTINE IS TO CALL THE TRS-80
                        01710 ;CHARACTER DISPLAY ROUTINE.   THE CONTENTS OF THE ACCUMU-
                        01720 ;LATOR ARE DISPLAYED.
                        01730 ;
                        01740 ;REGISTER DEFINITIONS:
                        01750 ;         NO REGISTERS ARE AFFECTED
                        01760 ;SUBROUTINES USED:
                        01770 ;         NONE
                        01780 ;
4EE4 D5                 01790 APRINT    PUSH    DE              ;SAVE REGISTERS
4EE5 FDE5               01800           PUSH    IY
4EE7 CD3300             01810           CALL    33H             ;PRINT CHARACTER
4EEA FDE1               01820           POP     IY              ;RESTORE REGISTERS
4EEC D1                 01830           POP     DE
4EED C9                 01840           RET
                        01850 ;
                        01860 ;********** SUBROUTINE REGDMP **********
                        01870 ;
                        01880 ;THE PURPOSE OF THIS SSUBROUTINE IS TO PROVIDE A HEX DUMP
                        01890 ;OF THE TEN Z-80 WORKING REGISTERS
                        01900 ;
                        01910 ;REGISTER DEFINITIONS:
                        01920 ;         HL      16-BIT DATA OUTPUT VECTOR
                        01930 ;         C       8-BIT DATA OUTPUT VECTOR
                        01940 ;SUBROUTINES USED:
                        01950 ;         APRINT          OUT08H
                        01960 ;         OUT16H
                        01970 ;
4EEE 41                 01980 REGSTR    DEFM    ' A  F  B  C  D  E  H  L   IX   IY'
4EEF 20
4EF0 20
4EF1 46
4EF2 20
4EF3 20
4EF4 42
4EF5 20
4EF6 20
4EF7 43
4EF8 20
4EF9 20
4EFA 44
4EFB 20
4EFC 20
4EFD 45
4EFE 20
4EFF 20
4F00 48
4F01 20
4F02 20
4F03 4C
4F04 20
4F05 20
4F06 20
4F07 49
4F08 58
4F09 20
4F0A 20
4F0B 20
4F0C 49
4F0D 59
4F0E 0D                 01990           DEFB    0DH
4F0F E5                 02000 REGDMP    PUSH    HL              ;SAVE WORKING REGISTERS
4F10 F5                 02010           PUSH    AF
4F11 C5                 02020           PUSH    BC
4F12 3E0D               02030           LD      A,0DH           ;ASCII "CR"
4F14 CDE44E             02040           CALL    APRINT          ;PRINT IT ON DISPLAY
4F17 0E21               02050           LD      C,33            ;LENGTH OF STRING
4F19 21EE4E             02060           LD      HL,REGSTR       ;ADDRESS OF STRING
```

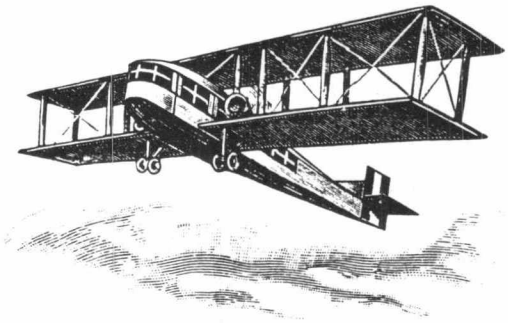
```
4F1C 7E       02070 LPREGD  LD      A,(HL)          ;GET CHARACTER
4F1D CDE44E   02080         CALL    APRINT          ;PRINT IT ON DISPLAY
4F20 23       02090         INC     HL              ;GO TO NEXT ADDRESS
4F21 0D       02100         DEC     C               ;DECREMENT COUNT
4F22 20F8     02110         JR      NZ,LPREGD       ;GO BACK OF NOT DONE
4F24 C1       02120         POP     BC              ;GO GET AF REGISTER PAIR
4F25 E1       02130         POP     HL              ;PLACE IN HL
4F26 E5       02140         PUSH    HL              ;PUT AF BACK IN STACK
4F27 C5       02150         PUSH    BC              ;PUT BC BACK IN STACK
4F28 4C       02160         LD      C,H             ;GET A REGISTER
4F29 CD9D4F   02170         CALL    OUT08H          ;PRINT IT ON DISPLAY
4F2C 3E20     02180         LD      A,20H           ;ASCII "SPACE"
4F2E CDE44E   02190         CALL    APRINT          ;PRINT IT
4F31 4D       02200         LD      C,L             ;GET F REGISTER
4F32 CD9D4F   02210         CALL    OUT08H          ;PRINT IT
4F35 3E20     02220         LD      A,20H           ;ASCII "SPACE"
4F37 CDE44E   02230         CALL    APRINT          ;PRINT IT
4F3A E1       02240         POP     HL              ;GET BC REGISTER PAIR
4F3B E5       02250         PUSH    HL              ;PUT BC BACK IN STACK
4F3C 4C       02260         LD      C,H             ;GET B REGISTER
4F3D CD9D4F   02270         CALL    OUT08H          ;PRINT IT
4F40 3E20     02280         LD      A,20H           ;ASCII "SPACE"
4F42 CDE44E   02290         CALL    APRINT          ;PRINT IT
4F45 4D       02300         LD      C,L             ;GET C REGISTER
4F46 CD9D4F   02310         CALL    OUT08H          ;PRINT IT
4F49 3E20     02320         LD      A,20H           ;ASCII "SPACE"
4F4B CDE44E   02330         CALL    APRINT          ;PRINT IT
4F4E 4A       02340         LD      C,D             ;GET D REGISTER
4F4F CD9D4F   02350         CALL    OUT08H          ;PRINT IT
4F52 3E20     02360         LD      A,20H           ;ASCII "SPACE"
4F54 CDE44E   02370         CALL    APRINT          ;PRINT IT
4F57 4B       02380         LD      C,E             ;GET E REGISTER
4F58 CD9D4F   02390         CALL    OUT08H          ;PRINT IT
4F5B 3E20     02400         LD      A,20H           ;ASCII "SPACE"
4F5D CDE44E   02410         CALL    APRINT          ;PRINT IT
4F60 C1       02420         POP     BC              ;GOING AFTER HL REGISTER
4F61 F1       02430         POP     AF
4F62 E1       02440         POP     HL              ;GOT IT
4F63 E5       02450         PUSH    HL              ;RESTORE STACK
4F64 F5       02460         PUSH    AF
4F65 C5       02470         PUSH    BC
4F66 4C       02480         LD      C,H             ;GET H REGISTER
4F67 CD9D4F   02490         CALL    OUT08H          ;PRINT IT
4F6A 3E20     02500         LD      A,20H           ;ASCII "SPACE"
4F6C CDE44E   02510         CALL    APRINT          ;PRINT IT
4F6F 4D       02520         LD      C,L             ;GET L REGISTER
4F70 CD9D4F   02530         CALL    OUT08H          ;PRINT IT
4F73 3E20     02540         LD      A,20H           ;ASCII "SPACE"
4F75 CDE44E   02550         CALL    APRINT          ;PRINT IT
4F78 DDE5     02560         PUSH    IX              ;GET IX REGISTER
4F7A E1       02570         POP     HL              ;READY TO DISPLAY
4F7B CD924F   02580         CALL    OUT16H          ;PRINT IT
4F7E 3E20     02590         LD      A,20H           ;ASCII "SPACE"
4F80 CDE44E   02600         CALL    APRINT          ;PRINT IT
4F83 FDE5     02610         PUSH    IY              ;GET IY REGISTER
4F85 E1       02620         POP     HL              ;READY TO DISPLAY
4F86 CD924F   02630         CALL    OUT16H          ;PRINT IT
4F89 3E0D     02640         LD      A,0DH           ;ASCII "CR"
4F8B CDE44E   02650         CALL    APRINT          ;PRINT IT
4F8E C1       02660         POP     BC              ;RESTORE REGISTERS
4F8F F1       02670         POP     AF
4F90 E1       02680         POP     HL
4F91 C9       02690         RET
              02700
```

```
              02710 ;********** SUBROUTINE OUT16H **********
              02720 ;
              02730 ;THIS SUBROUTINE WILL TAKE THE 16-BIT BINARY VALUE IN THE
              02740 ;HL REGISTER PAIR AND DISPLAY IT ON THE TRS-80 VIDEO
              02750 ;TERMINAL IN THE FORM OF FOUR HEXADECIMAL CHARACTERS.
              02760 ;
              02770 ;REGISTER DEFINITIONS:
              02780 ;         C       CHARACTER OUTPUT VECTOR
              02790 ;         HL      16-BIT DATA INPUT VECTOR
              02800 ;SUBROUTINES USED:
              02810 ;         OUT08H
              02820 ;
4F92 C5       02830 OUT16H  PUSH    BC              ;SAVE REGISTERS
4F93 4C       02840         LD      C,H             ;GET H REGISTER
4F94 CD9D4F   02850         CALL    OUT08H          ;PRINT IT ON DISPLAY
4F97 4D       02860         LD      C,L             ;GET L REGISTER
4F98 CD9D4F   02870         CALL    OUT08H          ;PRINT IT ON DISPLAY
4F9B C1       02880         POP     BC
4F9C C9       02890         RET
              02900 ;
              02910 ;********** SUBROUTINE OUT08H **********
              02920 ;
              02930 ;THIS SUBROUTINE WILL TAKE THE 8-BIT VALUE IN THE C
              02940 ;REGISTER AND DISPLAY IT ON A TRS-80 VIDEO TERMINAL
              02950 ;IN THE FORM OF TWO HEXADECIMAL CHARACTERS.
              02960 ;
              02970 ;REGISTER DEFINITIONS
              02980 ;         A       CHARACTER OUTPUT VECTOR
              02990 ;         C       DATA INPUT VECTOR
              03000 ;SUBROUTINES USED:
              03010 ;         BINASC          APRINT
              03020 ;
4F9D 3EF0     03030 OUT08H  LD      A,0F0H          ;LS NYBBLE MASK
4F9F A1       03040         AND     C               ;GET LS NYBBLE
4FA0 0F       03050         RRCA                    ;SHIFT TO LS BIT POSITION
4FA1 0F       03060         RRCA
4FA2 0F       03070         RRCA
4FA3 0F       03080         RRCA
4FA4 CDB44F   03090         CALL    BINASC          ;CONVERT TO ASCII
4FA7 CDE44E   03100         CALL    APRINT          ;PRINT IT ON DISPLAY
4FAA 3E0F     03110         LD      A,0FH           ;MS NYBBLE MASK
4FAC A1       03120         AND     C               ;GET MS NYBBLE
4FAD CDB44F   03130         CALL    BINASC          ;CONVERT TO ASCII
4FB0 CDE44E   03140         CALL    APRINT          ;PRINT IT ON DISPLAY
4FB3 C9       03150         RET
              03160 ;
              03170 ;********** SUBROUTINE BINASC **********
              03180 ;
              03190 ;THE PURPOSE OF THIS SUBROUTINE IS TO CONVERT A FOUR-BIT
              03200 ;NYBBLE INTO A SINGLE ASCII CHARACTER
              03210 ;
              03220 ;REGISTER DEFINITIONS:
              03230 ;         A       CHARACTER I/O VECTOR
              03240 ;SUBROUTINES USED:
              03250 ;         NONE
              03260 ;
4FB4 FE0A     03270 BINASC  CP      0AH             ;CHECK IF ASCII "A" - "F"
4FB6 FABB4F   03280         JP      M,LPBINA        ;IF NOT, JUMP
4FB9 C607     03290         ADD     A,7             ;PREPARE CONVERSION
4FBB C630     03300 LPBINA  ADD     A,30H           ;CONVERT TO ASCII
4FBD C9       03310         RET
              03320 ;
```

```
                03330 ;********** SUBROUTINE IN08HX **********
                03340 ;
                03350 ;THE PURPOSE OF THIS SUBROUTINE IS TO GET TWO ASCII
                03360 ;CHARACTERS FROM THE TRS-80 KEYBOARD (HEXADECIMAL)
                03370 ;AND CONVERT THEM TO 8-BIT BINARY AFTER ECHO TO DISPLAY.
                03380 ;
                03390 ;REGISTER DFINITIONS.
                03400 ;         A       CHARACTER INPUT VECTOR
                03410 ;SUBROUTINES USED:
                03420 ;         ASCBIN          APRINT
                03430 ;
4FBE C5         03440 IN08HX  PUSH    BC              ;SAVE REGISTERS
4FBF CDD44E     03450         CALL    AINPUT          ;GET ASCII CHARACTER
4FC2 CDD74F     03460         CALL    ASCBIN          ;CONVERT ASCII TO HEX
4FC5 4F         03470         LD      C,A             ;PLACE IN C REGISTER
4FC6 CB21       03480         SLA     C               ;SHIFT TO PROPER POSITION
4FC8 CB21       03490         SLA     C
4FCA CB21       03500         SLA     C
4FCC CB21       03510         SLA     C
4FCE CDD44E     03520         CALL    AINPUT          ;GET NEXT NYBBLE
4FD1 CDD74F     03530         CALL    ASCBIN          ;CONVERT TO HEX
4FD4 B1         03540         OR      C               ;MERGE NYBBLES
4FD5 C1         03550         POP     BC              ;RESTORE REGISTERS
4FD6 C9         03560         RET
                03570 ;
                03580 ;********** SUBROUTINE ASCBIN **********
                03590 ;
                03600 ;THIS SUBROUTINE CONVERTS AN ASCII CHARACTER IN THE A
                03610 ;REGISTER TO A SINGLE HEXADECIMAL NYBBLE.
                03620 ;
                03630 ;REGISTER DEFINITIONS.
                03640 ;         A       CHARACTER I/O VECTOR
                03650 ;SUBROUTINES USED:
                03660 ;         NONE
                03670 ;
4FD7 D630       03680 ASCBIN  SUB     30H             ;CONVERT TO BINARY
4FD9 FE0A       03690         CP      0AH             ;CHECK IF ASCII "A" - "F"
4FDB FAE04F     03700         JP      M,LPASCB        ;IF NOT, JUMP
4FDE D607       03710         SUB     7               ;CONVERT "A" - "F"
4FE0 C9         03720 LPASCB  RET
                03730 ;
                03740 ;********** SUBROUTINE BMLT08 **********
                03750 ;
                03760 ;THIS SUBROUTINE USES BOOTH'S ALGORITHM TO PROVIDE A FAST
                03770 ;MULTIPLICATION OF TWO SIGNED 8-BIT NUMBERS.  IT PRODUCES
                03780 ;A SIGNED 16-BIT PRODUCT.  (UNSIGNED OPERANDS ALSO WORK.)
                03790 ;
                03800 ;REGISTER DEFINITIONS
                03810 ;         A       MS BYTE OF PRODUCT
                03820 ;         B       BYTE COUNTER
                03830 ;         C       MULTIPLICAND
                03840 ;         D       MULTIPLIER / LS BYTE OF PRODUCT
                03850 ;         E       TEMPORARY MS BYTE OF PRODUCT
                03860 ;
4FE1 AF         03870 BMLT08  XOR     A       ;CLEAR A AND CARRY BIT
4FE2 0608       03880         LD      B,8     ;BYTE COUNTER
4FE4 5F         03890 LPBMLT  LD      E,A     ;SAVE MS BYTE OF PRODUCT
4FE5 7A         03900         LD      A,D     ;GET MULTIPLIER
4FE6 17         03910         RLA             ;PREPARE TO TEST LS BIT AND CARRY
4FE7 E603       03920         AND     3       ;MASK LS BIT AND OLD CARRY
4FE9 FE01       03930         CP      1       ;IF A=1 THEN ADD
4FEB 280C       03940         JR      Z,ADBMLT
4FED FE02       03950         CP      2       ;IF A=2 THEN SUB
4FEF 7B         03960         LD      A,E     ;RESTORE MS BYTE OF PRODUCT
4FF0 280B       03970         JR      Z,SBBMLT
4FF2 CB2F       03980 SHBMLT  SRA     A       ;SHIFT CONCENCATED AD REGISTERS
4FF4 CB1A       03990         RR      D       ;TO THE RIGHT, ONE BIT
4FF6 10EC       04000         DJNZ    LPBMLT  ;REPEAT IF NOT DONE
4FF8 C9         04010         RET             ;BYE BYE
4FF9 7B         04020 ADBMLT  LD      A,E     ;RESTORE MS BYTE OF PRODUCT
4FFA 81         04030         ADD     A,C     ;ADD MCAND TO MS BYTE OF PRODUCT
4FFB 18F5       04040         JR      SHBMLT  ;GO SHIFT
4FFD 91         04050 SBBMLT  SUB     C       ;SUBTRACT MCAND FROM PRODUCT
4FFE 18F2       04060         JR      SHBMLT  ;GO SHIFT
```

```
                04070 ;
                04080 ;
                04090 ;THIS LISTING OF SUBROUTINES IS AN EXAMPLE OF OVER 50
                04100 ;SUBROUTINES IN AN EXTENSIVE TRS-80/Z-80 SUBROUTINE
                04110 ;LIBRARY.  THE LIBRARY PROVIDES EXTENDED AND TABULAR
                04120 ;CALCULATIONS, INPUT AND OUTPUT OF DECIMAL, HEX, OCTAL,
                04130 ;BINARY AND DECIMAL DATA, MEMORY DATA MANIPULATION, AND
                04140 ;MANY OTHER CONVENIENT SUBROUTINES.  FOR INFORMATION,
                04150 ;WRITE TO:
                04160 ;
                04170 ;         SUBPROGRAM LIBRARY
                04180 ;         C/O   J. W. CARTER
                04190 ;         4932 SHIFRI
                04200 ;         MEMPHIS, TN     38117
                04210 ;
                04220 ;
4EA9            04230         END     START
00000 TOTAL ERRORS
SHBMLT  4FF2
SBBMLT  4FFD
ADBMLT  4FF9
LPBMLT  4FE4
LPASCB  4FE0
ASCBIN  4FD7
LPBINA  4FB8
BINASC  4FB4
OUT08H  4F9D
LPREGD  4F1C
REGSTR  4EEE
LPAINP  4ED7
AINPUT  4ED4
OUT16H  4F92
BMLT08  4FE1
REGDMP  4F0F
APRINT  4EE4
IN08HX  4FBE
START   4EA9
```

A Fast Fourier Transform for the 8080



BY ROBERT D. FUSFELD
16767 Marquez Terrace
Pacific Palisades, Cal. 90272

The Fast Fourier Transform (FFT) is a rapid method for digital computer calculation of Discrete Fourier Transforms (DFT). To begin with, the frequency spectrum of a continuous random wave will be a continuous curve with real and imaginary components. The imaginary components arise from analysis of a real signal because of the mathematical necessity of adjusting the phase relationship of the various frequency components of the spectrum to give a true representation of the original signal. However, when doing a digital calculation, the signal to be analysed must be sampled at discrete intervals. When this is done, a modification of the calculation process known as a DFT becomes necessary. In 1964 Cooley and Tukey (Ref. 1) developed the FFT which greatly reduces the calculation time while giving the same numerical results as the DFT. These results are an interval sampled approximation of the continuous frequency spectrum. A good review of the mathematical reasoning behind the FFT calculation is given by Brigham (Ref. 2). Douglass (Ref. 5) also gives some useful comments for users of the FFT.

The importance of Fourier Transforms is that spectral analysis is an extremely valuable tool in many types of engineering and scientific investigations. Applications occur in electrical engineering, vibration analysis, meteorology, speech recording and synthesis, music, etc. Naturally, due to the widespread use, many computer programs to perform the FFT calculation are readily available, particularly in high level languages such as ALGOL and FORTRAN (Ref. 2). When it comes to the microcomputer, BASIC programs are available such as the one by Stanley and Peterson (Ref. 3). However, in the interest of economy of memory and of computation time, implementation of the FFT in machine language is highly desireable. This was done for the 6800 CPU by Lord (Ref. 4). The program listed here is a similar one for the 8080 CPU.

This program is a direct implementation in 8080 machine language of Brigham's flow chart for the FFT calculation (Ref. 2, page 161). The calculation is performed on 256 points of input data and gives the power spectrum at 128 discrete points. The input data, calculations, and results are all done in single byte, two's complement format. This immediately limits the precision to approximately slide rule accuracy with two place decimal numbers. The indexing of data addresses is simplified for the 256 points by using Lord's suggestion of starting all data tables and the sine table on a page boundary. Further simplifications are made by writing the sine table as the sine times 128. The "MULT" routine then does a signed multiply and divide by 128. A "scale test" and "scale" procedure is done before each pass thru the data, during the calculation of each new data point and after calculation of power values. The number of times the real and imaginary data tables have been scaled by dividing by two is given in address 2277_H. However, the power values may have been scaled even further to prevent overflow. The power values at each discrete point are simply the sum of the squares of the real and imaginary data at that point.

To use this FFT program the user must input his 256 points of real data into addresses 2400_H to $24FF_H$ and fill the imaginary data table 2500_H to $25FF_H$ with zeros. I use a Cromemco D+7AI/O board to convert an analog signal into the

correct digital two's complement format. The next user's step is to "CALL FFT". The calculation takes about twenty seconds and is concluded with a "RET" command. The power spectrum will be located in addresses 2600_H to $267F_H$. The real and imaginary frequency spectrum values will be located in 2400_H to $247F_H$ and 2500_H to $257F_H$ respectively. The original input data in 2400_H to $24FF_H$ has been replaced by the real part of the answer. The answers have been unscrambled as required in the FFT algorithm at the end of the calculation. The user can display the power spectrum on an oscilloscope, print out the data, or use it in any way necessary for his study.

What do the points on the power spectrum signify? The first point is the relative power at zero frequency. This is the mean offset of the original data from zero. If the total of 256 data points indicate a time span of T seconds, then the interval between power spectrum points is 1/T Hertz. For example, if the original data was recorded at 10,000 Hertz then the 256 point sample would span .0255 seconds. The interval between points would then be 1/.0255 = 39.21 Hertz and the last point would be the power at 4979 Hertz. As is well known, the sampling rate should be at least twice the frequency of the highest frequency component in the signal.

The FFT program listed here has been assembled at 2000_H. The first 8K bytes of memory have been left for other uses. The program itself occupies 1K bytes and uses the next three pages starting at 2400_H (768 bytes) for the data and power tables. If the FFT program were to be reassembled at a different address, the first seven lines of the source listing would need to be changed. Some precautions will have to be taken. The real and imaginary data tables must start on consecutive page boundaries, *i.e.*, 2400_H and 2500_H. Also, "FIN" must equate to one higher than the high byte of "IMAG", *i.e.*, 26_H. Finally, the location of T1I must be at K+9. The program and sine table could be in ROM but K and the next 13 bytes must be in RAM as well as the data and power tables.

For users who might want to compare this program with Brigham's flow chart (Ref. 2), abbreviations used in the listing generally are those used in his book. However, because many commonly used printers (including mine) do not have lower case characters, there have been a few changes. K is used in place of k, REAL, REAL 1, IMAG, or IMAG 1 is used in place of $x_1(k)$, and REAL 2 or IMAG 2 is used in place of $x_1(k+N/2^1)$.

References

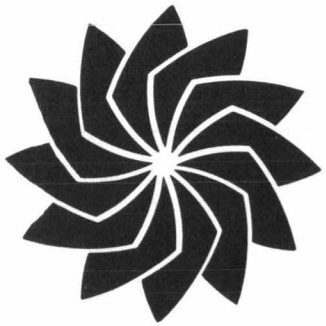
1. Cooley, J.W. and Tukey J.W.: An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics of Computation 19:297-301, 1965. Reprinted in: Rabiner and Roder, Eds.: Digital Signal Processing. IEEE Press, New York, 1972.
2. Brigham, E.O.: The Fast Fourier Transform. Prentice-Hall, New Jersey, 1974.
3. Stanley, W.D. and Peterson, S.J.: Fast Fourier Transforms on Your Home Computer. Byte, Vol. 3, P.14-25, December 1978.
4. Lord, R.H.: Fast Fourier for the 6800. Byte Vol. 4, P.108-119, February 1979.
5. Douglass, K.H.: Fast Fourier Transforms on Your Home Computer. Letter to the editor. Byte, Vol.4, P.225, May 1979.


```
ROBERT D. FUSFELD
16767 MARQUEZ TERRACE
PACIFIC PALISADES, CALIF. 9

A FAST FOURIER TRANSFORM
    FOR THE 8080

ASSM 2000                LIST
2000                0001 K      EQU  2270H
2000                0002 SIN    EQU  2300H
2000                0003 REAL   EQU  2400H
2000                0004 IMAG   EQU  2500H
2000                0005 POW    EQU  2600H
2000                0006 FIN    EQU  26H
2000                0007 T1I    EQU  79H
2000 21 00 01       0008 FFT    LXI  H,0100H   MAIN PROGRAM
2003 22 70 22       0009        SHLD K         INITIALIZE
2006 21 80 07       0010        LXI  H,0780H
2009 22 72 22       0011        SHLD K+2       INITIALIZE
200C 3A 71 22       0012 EFT    LDA  K+1       GET L
200F FE 09          0013        CPI  09H       COMPARE G+1
2011 CA 4F 20       0014        JZ   UNSCR     UNSCR IF L MORE THAN G
2014 3E 01          0015 CNTR   MVI  A,01      SET COUNTER
2016 32 74 22       0016        STA  K+4
2019 CD 70 20       0017 CALP   CALL PCAL      CALCULATE P
201C CD D8 20       0018        CALL XCAL      CALC NEW DATA VALUE
201F 21 70 22       0019        LXI  H,K       POINT TO K
2022 34             0020        INR  M         INCR K
2023 2C             0021        INR  L
2024 2C             0022        INR  L         POINT TO N2
2025 3A 74 22       0023        LDA  K+4       GET I
2028 BE             0024        CMP  M         COMPARE WITH N2
2029 CA 33 20       0025        JZ   STPA      IF EQU GO TO NEXT
202C 3C             0026        INR  A         IF NOT EQ INCR COUMTER
202D 32 74 22       0027        STA  K+4       STORE NEW COUNT
2030 C3 19 20       0028        JMP  CALP      REPEAT
2033 3A 70 22       0029 STPA   LDA  K         GET K
2036 86             0030        ADD  M         ADD N2
2037 32 70 22       0031        STA  K         STORE NEW K
203A D2 14 20       0032        JNC  CNTR      REPEAT
203D 7E             0033        MOV  A,M       GET N2
203E A7             0034        ANA  A         SET CARRY ZERO
203F 1F             0035        RAR            SHIFT
2040 77             0036        MOV  M,A       STORE NEW N2
2041 2C             0037        INR  L         POINT TO MJI
2042 35             0038        DCR  M         DECR IT
2043 2D             0039        DCR  L
2044 2D             0040        DCR  L         POINT TO L
2045 34             0041        INR  M         INCREMENT L
2046 2D             0042        DCR  L         POINT TO K
2047 36 00          0043        MVI  M,00      SET K TO ZERO
2049 CD AE 20       0044        CALL SCLT      SCALE IF OVER RANGE
204C C3 0C 20       0045        JMP  EFT       GO TO NEXT PASS
204F AF             0046 UNSCR  XRA  A         UNSCRAMBLE FFT DATA
2050 47             0047 UNS    MOV  B,A       STORE K IN B
2051 CD 9C 20       0048        CALL BITRV     BIT REVERSE K
2054 4F             0049        MOV  C,A       STORE IT
2055 78             0050        MOV  A,B       GET K BACK
2056 B9             0051        CMP  C         COMP K WITH BITRV K
2057 FA 63 20       0052        JN   REAR      REARRANGE IF LESS
205A FE FF          0053 CKE    CPI  0FFH      IS THIS THE LAST
205C CA 91 21       0054        JZ   POWER     IF LAST CALC POWER
```

```
205F 3C          0055          INR   A
2060 C3 50 20    0056          JMP   UNS       IF NOT LAST REPEAT
2063 21 00 24    0057 REAR     LXI   H,REAL    REARRANGE DATA
2066 11 00 24    0058          LXI   D,REAL
2069 68          0059          MOV   L,B       POINT TO REAL 1
206A 59          0060          MOV   E,C       POINT TO REAL 2
206B 7E          0061          MOV   A,M       GET REAL 1
206C F5          0062          PUSH  PSW       SAVE IT
206D 1A          0063          LDAX  D         GET REAL 2
206E 77          0064          MOV   M,A       MOVE IT
206F F1          0065          POP   PSW       GET REAL 1 BACK
2070 12          0066          STAX  D         MOVE IT
2071 24          0067          INR   H         POINT TO IMAG 1
2072 14          0068          INR   D         POINT TO IMAG 2
2073 7E          0069          MOV   A,M       GET IMAG 1
2074 F5          0070          PUSH  PSW       SAVE IT
2075 1A          0071          LDAX  D         GET IMAG 2
2076 77          0072          MOV   M,A       MOVE IT
2077 F1          0073          POP   PSW       GET IMAG 1 BACK
2078 12          0074          STAX  D         MOVE IT
2079 78          0075          MOV   A,B       GET K BACK
207A C3 5A 20    0076          JMP   CKE       REPEAT
207D 3A 73 22    0077 PCAL     LDA   K+3       CALCULATE P - GET NU1
2080 FE 00       0078          CPI   00
2082 CA 96 20    0079          JZ    MRDY      IF ZERO GO TO MRDY
2085 47          0080          MOV   B,A       PUT NU1 IN B
2086 3A 70 22    0081          LDA   K         GET K
2089 A7          0082 MCAL     ANA   A         CALCULATE M - SET CARRY ZERO
208A 1F          0083          RAR             DIVIDE BY 2
208B 05          0084          DCR   B
208C C2 89 20    0085          JNZ   MCAL      REPEAT IF NOT ZERO
208F CD 9C 20    0086 BR       CALL  BITRV     P IS BTRV OF M
2092 32 75 22    0087          STA   K+5       STORE P
2095 C9          0088          RET
2096 3A 70 22    0089 MRDY     LDA   K         GET K
2099 C3 8F 20    0090          JMP   BR        CALC BITRV OF K
209C C5          0091 BITRV    PUSH  B         CALC BIT REVERSE
209D D5          0092          PUSH  D
209E 16 08       0093          MVI   D,08      SET COUNTER
20A0 87          0094 REV      ADD   A         ROTATE A LEFT
20A1 4F          0095          MOV   C,A       SAVE A
20A2 78          0096          MOV   A,B       GET B
20A3 1F          0097          RAR             MOVE RT WITH CARRY
20A4 47          0098          MOV   B,A       STORE AGAIN IN B
20A5 79          0099          MOV   A,C       GET C
20A6 15          0100          DCR   D         DECREMENT COUNTER
20A7 C2 A0 20    0101          JNZ   REV       REPEAT IF NOT ZERO
20AA 78          0102          MOV   A,B       GET B BACK IF FINISHED
20AB D1          0103          POP   D         GET REGISTERS BACK
20AC C1          0104          POP   B
20AD C9          0105          RET
20AE 21 00 24    0106 SCLT     LXI   H,REAL    SCALE TEST
20B1 7E          0107 SCT      MOV   A,M       GET DATA
20B2 FE 40       0108          CPI   40H       IS IT OVER RANGE
20B4 F2 C4 20    0109          JP    SCALE     SCALE IF OVER
20B7 FE C0       0110          CPI   0C0H      IS IT UNDER RANGE
20B9 FA C4 20    0111          JM    SCALE     SCALE IF UNDER
20BC 23          0112          INX   H         POINT TO NEXT DATA
20BD 7C          0113          MOV   A,H       GET HIGH BYTE OF ADDRESS
20BE FE 26       0114          CPI   FIN       TEST FOR END OF DATA
20C0 C2 B1 20    0115          JNZ   SCT       REPEAT IF NOT END
20C3 C9          0116          RET
20C4 21 00 24    0117 SCALE    LXI   H,REAL    SCALE THE DATA
20C7 7E          0118 SCA      MOV   A,M       GET DATA
20C8 8F          0119          ADC   A         SET CARRY EQU BIT 7
20C9 7E          0120          MOV   A,M       GET DATA BACK
20CA 1F          0121          RAR             DIVIDE BY 2
20CB 77          0122          MOV   M,A       STORE SCALED DATA
20CC 23          0123          INX   H         POINT TO NEXT DATA
20CD 7C          0124          MOV   A,H       GET HIGH BYTE OF ADDRESS
20CE FE 26       0125          CPI   FIN       TEST FOR END OF DATA
20D0 C2 C7 20    0126          JNZ   SCA       IF NOT END REPEAT
20D3 21 77 22    0127          LXI   H,K+7     POINT TO SCALE FACTOR
20D6 34          0128          INR   M         INCR SCALE FACTOR
20D7 C9          0129          RET
20D8 21 70 22    0130 XCAL     LXI   H,K       CALCULATE NEW DATA VALUE
20DB 7E          0131 CALT1    MOV   A,M       CALCULATE T1 - GET DATA
20DC 2C          0132          INR   L
20DD 2C          0133          INR   L         POINT TO N2
20DE 86          0134          ADD   M         CALCULATE K+N2
20DF 32 76 22    0135          STA   K+6       STORE IT
20E2 21 00 24    0136          LXI   H,REAL    POINT TO REAL DATA
20E5 6F          0137          MOV   L,A
20E6 7E          0138          MOV   A,M       GET DATA
20E7 32 7A 22    0139          STA   K+10      SAVE IT
20EA 4E          0140          MOV   C,M       PUT REAL DATA IN C
20EB 3A 75 22    0141          LDA   K+5       GET P
20EE 21 00 23    0142          LXI   H,SIN
20F1 6F          0143          MOV   L,A       POINT TO SIN
20F2 7E          0144          MOV   A,M       GET SIN
20F3 32 7C 22    0145          STA   K+12      STORE IT
20F6 CD 3A 22    0146          CALL  MULT      CALC R*SIN
20F9 2F          0147          CMA
20FA 3C          0148          INR   A         CALC -R*SIN
20FB 32 79 22    0149          STA   K+9       STORE IT
20FE 3A 75 22    0150          LDA   K+5       GET P
2101 C6 40       0151          ADI   40H       CALC ADDR OF COS
2103 21 00 23    0152          LXI   H,SIN
2106 6F          0153          MOV   L,A       POINT TO COS
2107 7E          0154          MOV   A,M       GET COS
2108 32 7D 22    0155          STA   K+13      STORE IT
210B CD 3A 22    0156          CALL  MULT      CALC R*COS
210E 32 78 22    0157          STA   K+8       STORE IT
2111 3A 76 22    0158          LDA   K+6       GET K+N2
2114 21 00 25    0159          LXI   H,IMAG
2117 6F          0160          MOV   L,A       POINT TO IMAG DATA
2118 4E          0161          MOV   C,M       PUT IMAG DATA IN C
2119 3A 7C 22    0162          LDA   K+12      GET SIN
211C CD 3A 22    0163          CALL  MULT      CALC IMAG*SIN
211F 21 78 22    0164          LXI   H,K+8     POINT TO R*COS
2122 86          0165          ADD   M         CALC  IMAG*SIN+R*COS
2123 77          0166          MOV   M,A       STORE IT
2124 3A 7D 22    0167          LDA   K+13      GET COS
2127 CD 3A 22    0168          CALL  MULT      CALC IMAG*COS
212A 21 79 22    0169          LXI   H,K+9     POINT TO -R*SIN
212D 86          0170          ADD   M         CALC  IMAG*COS-R*SIN
212E 77          0171          MOV   M,A       STORE IT
212F 21 7B 22    0172          LXI   H,K+11    POINT TO IMAG DATA BUFFER
2132 71          0173          MOV   M,C       STORE IMAG DATA
2133 21 78 22    0174          LXI   H,K+8     SCALE TEST T1-POINT TO T1
2136 7E          0175 STT1     MOV   A,M       GET T1
2137 FE 40       0176          CPI   40H
2139 F2 4B 21    0177          JP    ST1       SCALE IF OVER RANGE
213C FE C0       0178          CPI   0C0H
213E FA 4B 21    0179          JM    ST1       SCALE IF UNDER RANGE
2141 7D          0180          MOV   A,L
```

```
2142 FE 79      0181          CPI   T1I     TEST FOR REAL AND IMAG PARTS
2144 CA 50 21   0182          JZ    XCAL1   IF SO RETURN TO CALC OF DATA
2147 2C         0183          INR   L
2148 C3 36 21   0184          JMP   STT1    IF NOT REPEAT
214B 21 78 22   0185 ST1      LXI   H,K+8   SCALE T1-POINT TO REAL PART
214E 7E         0186 ST11     MOV   A,M     GET T1 REAL
214F 8F         0187          ADC   A       SET CARRY EQU BIT 7
2150 7E         0188          MOV   A,M     GET T1 REAL BACK
2151 1F         0189          RAR           DIVIDE BY 2
2152 77         0190          MOV   M,A     PUT IT BACK
2153 2C         0191          INR   L       POINT TO NEXT
2154 7D         0192          MOV   A,L
2155 FE 7A      0193          CPI   T1I+1   TEST ADDRESS
2157 C2 4E 21   0194          JNZ   ST11    REPEAT IF NOT FINISHED
215A CD C4 20   0195          CALL  SCALE   SCALE THE REST OF THE DATA
215D 11 78 22   0196 XCAL1    LXI   D,K+8   FINISH NEW DATA CALCULATION
2160 3A 70 22   0197          LDA   K       GET K
2163 21 00 24   0198          LXI   H,REAL
2166 6F         0199          MOV   L,A     POINT TO REAL 1
2167 1A         0200          LDAX  D       GET T1 REAL
2168 86         0201          ADD   M       CALC NEW REAL 1
2169 77         0202          MOV   M,A     STORE IT
216A 1C         0203          INR   E       POINT TO T1 IMAG
2168 24         0204          INR   H       POINT TO IMAG 1
216C 1A         0205          LDAX  D       GET T1 IMAG
216D 86         0206          ADD   M       CALC NEW IMAG 1
216E 77         0207          MOV   M,A     STORE IT
216F 1A         0208          LDAX  D       GET T1 IMAG BACK
2170 2F         0209          CMA
2171 3C         0210          INR   A       CALC -T1 IMAG
2172 86         0211          ADD   M       CALC NEW IMAG 2
2173 4F         0212          MOV   C,A     TEMP STORE IT
2174 3A 72 22   0213          LDA   K+2     GET N2
2177 85         0214          ADD   L       CALC K+N2
2178 6F         0215          MOV   L,A     POINT TO IMAG 2
2179 71         0216          MOV   M,C     STORE NEW IMAG 2
217A 1D         0217          DCR   E       POINT TO T1 REAL
217B 25         0218          DCR   H       POINT TO REAL 2
217C 1A         0219          LDAX  D       GET T1 REAL
217D 2F         0220          CMA
217E 3C         0221          INR   A       CALC -T1 REAL
217F 4F         0222          MOV   C,A     TEMP STORE IT
2180 3A 70 22   0223          LDA   K       GET K
2183 6F         0224          MOV   L,A     POINT TO REAL 1
2184 7E         0225          MOV   A,M     GET IT
2185 81         0226          ADD   C       CALC REAL 2
2186 4F         0227          MOV   C,A     STORE IT
2187 3A 72 22   0228          LDA   K+2     GET N2
218A 85         0229          ADD   L       CALC K+N2
218B 6F         0230          MOV   L,A     POINT TO REAL 2
218C 71         0231          MOV   M,C     STORE NEW REAL 2
218D CD AE 20   0232          CALL  SCLT    TEST AND SCALE IF OVER RANGE
2190 C9         0233          RET
2191 01 00 26   0234 POWER    LXI   B,POW   POWER CALC - POINT TO POW TABLE
2194 21 00 24   0235          LXI   H,REAL  POINT TO REAL DATA TABLE
2197 7E         0236 PC1      MOV   A,M     GET REAL DATA
2198 CD 1C 22   0237          CALL  SQA     SQUARE DATA
219B 7B         0238          MOV   A,E     GET LOW BYTE OF SQR
219C 02         0239          STAX  B       STORE IT
219D 0C         0240          INR   C       POINT TO HIGH BYTE OF SQR
219E 7A         0241          MOV   A,D     GET HIGH BYTE
219F 02         0242          STAX  B       STORE IT
21A0 0D         0243          DCR   C       GO BACK TO POW TABLE
21A1 24         0244          INR   H       POINT TO IMAG TABLE
21A2 7E         0245          MOV   A,M     GET IMAG DATA
21A3 25         0246          DCR   H       POINT TO REAL DATA TABLE
21A4 CD 1C 22   0247          CALL  SQA     SQUARE IMAG DATA
21A7 E5         0248          PUSH  H       SAVE REAL DATA POINTER
21A8 0A         0249          LDAX  B       GET LO BYTE OF REAL SQR
21A9 6F         0250          MOV   L,A     PUT IT IN L
21AA 0C         0251          INR   C       POINT TO HI BYTE OF REAL SQR
21AB 0A         0252          LDAX  B       MOVE IT TO A
21AC 67         0253          MOV   H,A     GET REAL SQR IN HL
21AD 19         0254          DAD   D       CALC REAL SQR + IMAG SQR
21AE 7C         0255          MOV   A,H     GET HIGH BYTE
21AF 02         0256          STAX  B       PUT IT IN POWER TABLE
21B0 0D         0257          DCR   C       POINT TO LOW BYTE
21B1 7D         0258          MOV   A,L     GET IT
21B2 02         0259          STAX  B       PUT IT IN POWER TABLE
21B3 E1         0260          POP   H       GET DATA POINTER BACK
21B4 2C         0261          INR   L       GO TO NEXT DATA
21B5 0C         0262          INR   C
21B6 0C         0263          INR   C       POINT TO NEXT POW TABLE LOC
21B7 C2 97 21   0264          JNZ   PC1     REPEAT IF NOT AT END
21BA 21 00 26   0265          LXI   H,POW   SCALE POWER TO ONE BYTE
21BD 3E 00      0266          MVI   A,00    SEARCH FOR MAX HIGH BYTE
21BF 2C         0267 PS1      INR   L
21C0 46         0268          MOV   B,M
21C1 BE         0269          CMP   M
21C2 F2 C6 21   0270          JP    PS2
21C5 7E         0271          MOV   A,M
21C6 2C         0272 PS2      INR   L
21C7 C2 BF 21   0273          JNZ   PS1
21CA A7         0274          ANA   A       IS MAX HIGH BYTE ZERO
21CB CA FD 21   0275          JZ    PS8     IF SO - GO TO LOW BYTE TEST
21CE 0E 00      0276          MVI   C,00    SET UP COUNTER
21D0 0C         0277 PS3      INR   C       COUNT SHIFTS TO RIGHT
21D1 A7         0278          ANA   A
21D2 1F         0279          RAR
21D3 A7         0280          ANA   A
21D4 C2 D0 21   0281          JNZ   PS3     REPEAT IF NOT COMPLETE
21D7 0C         0282          INR   C
21D8 21 00 26   0283          LXI   H,POW   POINT TO POWER TABLE
21DB 41         0284 PS4      MOV   B,C     SHIFT POWER DATA TO RIGHT
21DC 2C         0285 PS5      INR   L       POINT TO HIGH BYTE
21DD 7E         0286          MOV   A,M     GET HIGH BYTE
21DE A7         0287          ANA   A       CLEAR CARRY
21DF 1F         0288          RAR           SHIFT DATA TO RIGHT
21E0 77         0289          MOV   M,A     PUT IT BACK
21E1 2D         0290          DCR   L       POINT TO LOW BYTE
21E2 7E         0291          MOV   A,M     GET LOW BYTE
21E3 1F         0292          RAR           SHIFT IT RIGHT
21E4 77         0293          MOV   M,A     PUT IT BACK
21E5 05         0294          DCR   B
21E6 C2 DC 21   0295          JNZ   PS5
21E9 2C         0296          INR   L
21EA 2C         0297          INR   L
21EB C2 DB 21   0298          JNZ   PS4
21EE 21 00 26   0299 PS6      LXI   H,POW   INSERT LOW BYTES IN POWER
21F1 11 00 26   0300          LXI   D,POW                         TABLE
21F4 1A         0301 PS7      LDAX  D
21F5 77         0302          MOV   M,A
21F6 2C         0303          INR   L
21F7 1C         0304          INR   E
21F8 1C         0305          INR   E
```

```
21F9 C2 F4 21    0306          JNZ   PS7
21FC C9          0307 END      RET                 END OF POWER SCALING
21FD 21 00 26    0308 PS8      LXI   H,POW         SEARCH FOR OVER RANGE LOW
2200 7E          0309 PS9      MOV   A,M                           BYTE
2201 A7          0310          ANA   A
2202 FA 0D 22    0311          JM    PSA
2205 2C          0312          INR   L
2206 2C          0313          INR   L
2207 C2 00 22    0314          JNZ   PS9
220A C3 EE 21    0315          JMP   PS6
220D 21 00 26    0316 PSA      LXI   H,POW         SHIFT LOW BYTES RIGHT
2210 7E          0317 PSB      MOV   A,M
2211 A7          0318          ANA   A
2212 1F          0319          RAR
2213 77          0320          MOV   M,A
2214 2C          0321          INR   L
2215 2C          0322          INR   L
2216 C2 10 22    0323          JNZ   PSB
2219 C3 EE 21    0324          JMP   PS6
221C C5          0325 SQA      PUSH  B             SQUARE NUMBER IN ACCUM
221D E5          0326          PUSH  H
221E A7          0327          ANA   A             TEST ACCUM
221F F2 24 22    0328          JP    SQA1          IF POSITIVE GO AHEAD
2222 2F          0329          CMA
2223 3C          0330          INR   A             IF NEGATIVE NEGATE AND THEN GO
2224 4F          0331 SQA1     MOV   C,A           SET UP TO MULTIPLY
2225 06 00       0332          MVI   B,00
2227 21 00 00    0333          LXI   H,0000
222A 16 08       0334          MVI   D,08H
222C 29          0335 SQA2     DAD   H             MULTIPLY A*A
222D 07          0336          RLC
```



```
222E D2 32 22    0337          JNC   SQA3
2231 09          0338          DAD   B
2232 15          0339 SQA3     DCR   D
2233 C2 2C 22    0340          JNZ   SQA2
2236 EB          0341          XCHG
2237 E1          0342          POP   H
2238 C1          0343          POP   B
2239 C9          0344          RET
223A 11 00 08    0345 MULT     LXI   D,0800H      SET UP TO DO SIGNED MULTIPLY
223D 21 00 00    0346          LXI   H,0000      AND DIVIDE BY 128
2240 06 00       0347          MVI   B,00
2242 C5          0348          PUSH  B           SAVE MULTIPLICAND IN C
2243 F5          0349          PUSH  PSW         SAVE MULTIPLIER IN A
2244 79          0350          MOV   A,C         GET MULTIPLICAND
2245 A7          0351          ANA   A           SET SIGN FLAG
2246 F2 4C 22    0352          JP    M1          IF POSITIVE GO AHEAD
2249 1C          0353          INR   E           IF NEG SET SIGN FLAG
224A 2F          0354          CMA
224B 3C          0355          INR   A           IF NEG NEGATE MULTIPLICAND
224C 4F          0356 M1       MOV   C,A         PUT MULTIPLICAND IN C
224D F1          0357          POP   PSW         GET MULTIPLIER
224E A7          0358          ANA   A           SET SIGN FLAG
224F F2 55 22    0359          JP    MPY         IF POS GO AHEAD
2252 1C          0360          INR   E           IF NEG CHANGE SIGN FLAG
2253 2F          0361          CMA
2254 3C          0362          INR   A           IF NEG NEGATE MULTIPLIER
2255 29          0363 MPY      DAD   H           SHIFT LEFT
2256 07          0364          RLC               TEST BIT 7
2257 D2 5B 22    0365          JNC   CNT         SKIP IF BIT WAS ZERO
225A 09          0366          DAD   B           ADD MULTIPLICAND
225B 15          0367 CNT      DCR   D           DECR COUNTER
225C C2 55 22    0368          JNZ   MPY         REPEAT IF NOT FINISHED
225F 29          0369          DAD   H           SHIFT ANSWER LEFT
2260 C1          0370          POP   B           GET MULTIPLICAND BACK
2261 7B          0371          MOV   A,E         GET SIGN FLAG
2262 E6 01       0372          ANI   01H         TEST IT
2264 7C          0373          MOV   A,H         GET HIGH BYTE FOR FINAL ANSWER
2265 C8          0374          RZ                FINISHED IF POSITIVE
2266 2F          0375          CMA
2267 3C          0376          INR   A           NEGATE ANS IF NEGATIVE
2268 C9          0377          RET               FINISHED IF NEGATIVE
```

SINE TABLE

```
ASSME 2000
NONE

DUMP 2300 23FF
2300 00 03 06 09 0C 0F 12 15 18 1C 1F 22 25 28 2B 2E
2310 30 33 36 39 3C 3F 41 44 47 49 4C 4E 51 53 55 58
2320 5A 5C 5E 60 62 64 66 68 6A 6C 6D 6F 70 72 73 75
2330 76 77 78 79 7A 7B 7C 7C 7D 7E 7E 7F 7F 7F 7F 7F
2340 7F 7F 7F 7F 7F 7F 7E 7E 7D 7C 7C 7B 7A 79 78 77
2350 76 75 73 72 70 6F 6D 6C 6A 68 66 64 62 60 5E 5C
2360 5A 58 55 53 51 4E 4C 49 47 44 41 3F 3C 39 36 33
2370 30 2E 2B 28 25 22 1F 1C 18 15 12 0F 0C 09 06 03
2380 00 FC F9 F6 F3 F0 ED EA E7 E3 E0 DD DA D7 D4 D1
2390 CF CC C9 C6 C3 C0 BE BB B8 B6 B3 B1 AE AC AA A7
23A0 A5 A3 A1 9F 9D 9B 99 97 95 93 92 90 8F 8D 8C 8A
23B0 89 88 87 86 85 84 83 83 82 81 81 80 80 80 80 80
23C0 80 80 80 80 80 80 81 81 82 83 83 84 85 86 87 88
23D0 89 8A 8C 8D 8F 90 92 93 95 97 99 9B 9D 9F A1 A3
23E0 A5 A7 AA AC AE B1 B3 B6 B8 BB BE C0 C3 C6 C9 CC
23F0 CF D1 D4 D7 DA DD E0 E3 E7 EA ED F0 F3 F6 F9 FC
```

access addend ALGORITHM algebra analog application ai

A MONTHLY ALGORITHM COLUMN

BY DENNIS ALLISON

With this issue of *Dr. Dobb's Journal* we begin a new series concentrating on the building-blocks of systems rather than the systems themselves. One of the failings of most of us is the inability or disinclination to take and use the work of others. Everyone writes his own sort program, his own integration program, and so on *ad infinitum*. Often it would be better and faster to take an existing program and reuse it. Of course that is not always possible, but often one can take the algorithm and adapt it easily.

The intent is for this to be a reader participation column. We will publish submitted algorithms accompanied by adequate, if terse, documentation and (if possible) some measure of their effectiveness. We certainly need a test program drive and the results of a few test runs. Some analysis is important. One needs to know, for example, if a sort program runs in time n*log(n) where the log is base 2 or in time n^2.

From time to time we will publish reference lists of published algorithms. For that reason, if you find a published algorithm to be of use, please send it to Algorithms, *Dr. Dobb's Journal*, PO Box E, Menlo Park, CA 94025 so it can be added to the file. Be sure to include a complete reference; a few comments would also be welcome.

Lastly, if you can improve on an algorithm published here, or if you find a bug, we will publish corrections and improvements. There's a lot to be learned about programming from a study of how algorithms are constructed.

Algorithms may be programmed in any convenient language, including assembler, but are preferred in a high level procedural language such as Pascal or C. On the other hand, some algorithms are particularly efficient and interesting simply because they match the machine structure well. Occasionally we'll just mention a trick, a data structure, a gimmick which seems to be particularly useful.

Merge Sort

The Merge Sort sorts a list of keys in-place in time n*log(n). It is very simple to state:

> Recursively divide the list to be sorted in half until sorted lists are obtained (that is, lists of length one); then successively merge each pair of lists until the entire list is sorted.

RECURSIVE FORM

```
int a[11], link[11];

sort (i,j) int i,j; { int k, ii, jj;
        if(i==j) { link[i] = 0; return(i);}
        k = i+((j-i)>>1);
        ii = sort(i,k); jj = sort(k+1,j);
        return(merge(ii,jj));
}


merge(i,j) int i,j; { int k;
        if (i==0) return (j);
        if (j==0) return (i);
        if(a[i]>a[j]) { k = i; i = link[i];}
                  else { k = j; j = link[j];}
        link[k] = merge(i,j);
        return(k);
}
```

NON-RECURSIVE FORM

```
merge(i,j) int i,j; { int k,m;
        k = 0;
        while( (i!=0) && (j!=0)) {
                if(a[i]<a[j]) { m = i; i = j; j = m;}
                m = link[i];
                link[i] = k;
                k = i; i = m;
        }
        if (i==0) { m = i; i = j; j = m; }
        m = i;
        while( k != 0) {
                j = link[k];
                link[k] = m;
                m = k;
                k = j;
        }
        return(m);
}
```

The sort is obviously n*log(n) since there can be at most log(n) partitioning operations and any merge operation can be, at most, n operations long.

The program given here uses an array of keys and an associated array of links. The index value zero is distinguished as a list terminator. The programming language is C. The routine sort returns the index of the first element of the sorted list, call it j. The a[j] is the first sorted element and link[j] is the index of the next element, and so forth. Sort takes as arguments the first and last indexes to be sorted. If this is a single element (the indexes are equal) it simply initializes the link field and returns as a list of length one is, by definition, sorted. If there is more than one element, sort partitions the list in two and calls itself to sort the two partitioned lists and then merges them returning the head of the merged lists.

The merge routine merges the two sorted lists whose heads are given as arguments and returns the head of the new list. The data are not actually moved, only the link fields are modified. Again a recursive formulation is used. Only the top elements of the two lists need be considered. If either of the lists is empty then the other list is the result. If not, then one finds the index of the largest, removes it from the sublist, and merges the remaining values.

As written, this program is somewhat inefficient since it uses a local variable to save the index for the merge, and it will recur as many times as the lists are long.

The performance can be improved by using an iterative rather than a recursive formulation of the merge. The procedure is similar to the recursive one. The heads of the two lists are examined and the largest removed and placed on a list using its link field to thread the list and the procedure repeated until one of the lists is empty. At that point the entirety of the other list becomes the tail of the merged list. Items from the reversed list are added to the tail until the reversed list is empty. At that point the merge is complete. While this takes more code, it operates in place with a minimum number of procedure calls.

Letters

ADVERGATE . . .

Dear Suzanne;

If you are unable to produce Dr. Dodd's [*sic*] without the advertisements, you very much should resign. In the great muddled world of the small computer, the unbiased product which Dr. Dodd's [*sic*] Journal was prior to your editorship is greatly needed. Another advertiser base journal is sorely unneeded.

An example in point is your advertisement for Lifeboat Associates. This organization represents the greatest obstacle to small computer software development. Almost all worthwhile CP-M products are now sold through Lifeboat. Because of their non-support of the small dealers, these products are not available in the computer stores. Without the support of the computer store, most of these products are worthless.

I am strictly a user of these products and am not in any way a dealer. I have found a great many problems in the CP-M packages which I do use and have spent hundreds of hours trying to work these problems out without any local support whatsoever. Dr. Dodd's [*sic*] Journal has been a great help in circumventing this system. Now with full page ads from Lifeboat Associates, you can count on the fact that you will now produce the glowing reports of these products which are worthless.

Sincerely,
Robert C. Luckey, M.D.

Medical Central Service
471 Williams, Suite 4
Richland, Washington 99352

A symptom of our times, a parable of today: guilty until proven innocent, suspect without cause, wrong even if you're right. Although I'm not quite sure what Dr. Luckey is accusing us of, it seems to be vaguely a general suspicion that, because we now have advertising in DDJ, *we are guilty . . . of* something, *I don't know what.*

I should know, since everything that goes into this magazine filters through me: there has never been a case of altering anything in the magazine to please an advertiser. To be honest, our advertisers report unusually excellent response from advertisements in DDJ, *and since we limit the amount of advertising in our pages, we don't really have to bow down to advertisers. We haven't done it yet, and I hope we never do. I do have a certain way of looking at the world, and it does not include "selling out" or going out of my way to accommodate myself to another's point of view if it does not fit in with what I believe. It's called "ethical." Not too obsolete a word, I hope.*

I think this is really an unfair letter: it is all generalizations and vague assumptions. Any decent graduate school in journalism would give an automatic "F" to Dr. Luckey for all accusations/no hard data. Dr. Luckey thinks another "advertiser base journal is sorely unneeded." Well, that may or may not be true, but frankly, just between us wage-earners, I appreciate getting my very small paycheck twice a month. – SR

TABLE DRIVEN ASSEMBLER IN DDJ # 42

Dear *Dr. Dobb's*:

Due to a phone call from a reader, I have become aware of an oversight concerning the assembler. As I have been developing a number of large programs in ML-80, I long ago modified the compiler to substantially enlarge its internal tables and fix a few bugs. As a result, the assembler can't be compiled by ML-80 as currently distributed by the CPM User's Group (I have sent them the mods, but during their year-long reorganization confusion it may be a while yet) due to its size. The disk I am making available does contain ML-80 with all my patches and will compile it. If I had the source of ML-80 and I had access to a PL/M compiler, I would have made changes that way, but I have neither. One side effect of the changes is that the modified L82 will not run in a 16K system. (It runs fine in 32K, with one small change it would run in 24K, but compile smaller programs.)

Sincerely,
Darrel J. Van Buer

1522 Brockton Ave., Apt. 5
Los Angeles, CA 90025

NORTH STAR PATCHES

Dear *Dr. Dobb's*,

Here are some software patches for the North Star Horizon which might prove of value to readers who use this system.

Two minor but irritating problems with North Star software are the use of the underline character 5FH for backspace and the '@' symbol for line cancel. These are useful functions but why use printing characters? They also implement the rubout or delete character 7FH as a backspace. If you use their personalization routine to change the DELECHO from 5FH to 08H, then you enable the backspace key giving you no less than three different backspace commands. Also, you will note that the backspace 08H will not function in DOS.

Therefore: For Release 4 Software Only

In DOS:	263C holds the backspace character; change 5FH to 08H
	2637 holds the line cancel character (40H); change to your choice. I use 7FH (rubout).
In BASIC:	48A2 holds the backspace; change 5FH to 08H
	48BB holds the line cancel character; change as in DOS.

497B holds the character echo for line cancel; it contains 40H you do not need to change it unless you want to.
48A6-48AA implements a backspace on rubout; if you use the rubout for line cancel as I did, you will need to replace these bytes with 00H.

In MONITOR: This Applies to M6700
6740 holds the backspace character, change to 08H
673B holds the line cancel character, change as in DOS.
686A holds the character (3FH) printed after a line cancel; this is the same character printed after an invalid command. It is probably best to leave it alone.

This should make it possible to have a consistent set of function keys and to regain two 'lost' printing characters.

Sincerely,
Richard M. Johnson

502 W. Kenicott
Carbondale IL 62901

LONG AGO AND FAR AWAY

Dear Suzanne:

One of your readers pointed out a mistake in my article, "A Floating Point Subroutine Package for the 1802" in *DDJ* #37. The subroutine call addresses for floating point subtraction (AC-OP → AC) should be 02E9. It was printed as 029E.

Yours truly,
Paul Wasserman

918 Ray Avenue
Union, NJ 07083

INCREASING Z-80 POWER

Dear Suzanne,

This is something that your readers might be interested in which I stumbled across doing software work for a little firm here in Sacramento. I have since found that it is not at all well known. My favorite micro chip is the Z-80 (In fact, I think the 8080 is completely obsolete and has been since the first Z-80 was fabricated. So why the insistence on "8080 compatible software"? Never mind . . .). The basic statement of what I discovered is that the Z-80 is far more than Zilog has seen fit to tell us. The official tally of Z-80 instructions is 694, according to my Z-80 tech manual. My discovery was that about 410 instructions can be added to that total, genuine useful instructions that any Z-80 will execute. Most involve the twin index registers, and consist of things such as "Load upper half of IX from any 8-bit register" or "Load lower half of IY immediate". In other words, the index registers can be used to a much greater level of efficiency than before. The other members of the new set are things such as "Shift left and set least signficant bit".

The key to the new instructions came when I finally saw something that a lot of people have probably seen: The object code for indexed instructions can be formed by taking the object code for an operation involving HL and tacking a 0FDH or ODDH on at the front end (not quite, there is a problem with the displacement byte, but to a first approximation this is correct). What occurred to me was to try tacking those bytes onto an instruction involving just H. So I did, and it worked!

Some notation for the following examples: HX is the high byte of the IX register, LX the low byte, and similarly for the IY register.

Desired instruction	Object code	Standard Assembler
LD HX,A	DD 67	DEFB 0DDH LD H,A
OR LY	FD B5	DEFB 0FDH OR L
ADD A,HY	FD 84	DEFB 0FDH ADD A,H

You see the pattern. There are some limitations; it is not possible, for instance, to LD HX,LY. On the other hand, LD HX,LX is possible. The rule is that only one index register at a time may be referenced.

The other type of additonal instructions I've found do not involve the index registers. On page 50 of the Z-80 Technical Manual, in the Op-Code column, the object codes for the various rotate and shift instructions are defined in terms of prototypical configurations, with the various codes being differentiated by a 3 bit field. This means that there are 8 possible operation types. NOTE THAT ONLY SEVEN ARE LISTED IN THE MANUAL! The missing one is pattern 6. When it is generated, the results show a "Shift left and set least significant bit"; i.e., the operand is shifted left and the low bit is set rather than reset. The high bit of the operand is shifted into the carry, and sign and parity are affected in the expected manner. This one is moderately useful, also.

Any good macro assembler can generate these extra codes, and in fact, the Microsoft MACRO-80 package allows a single macro to generate all the extra index register instructions. I am sure that other packages will also. Again. ANY Z-80 will execute them. I am on the track of a few more. Once I get them all, I am going to write an assembler that accepts these extended codes. The only question now remaining is: why didn't Zilog tell us?

I hope the above sets someone onto a better way to accomplish a project.

Sincerely,
Daniel R. Lunsford

8725 La Riviera Dr., #68
Sacramento, CA 95826

A SLIGHT CORRECTION

To the Editor:

I would like to correct a slight inaccuracy in the article "CP/M and CDOS Program Compatibility", by Jim Warner, in the January, 1980, issue.

The article states "The address of the jump vector table depends on the amount of memory allocated to CP/M and unfortunately D.R. has not specified how transient programs should locate it." Actually, the D.R. CP/M System Alteration Guide, Rev. Jan., 1978, pages 17 and 21, specifies that location 0, 1, 2 must contain

```
JMP     WBOOT
```

where WBOOT is the address of the second entry in the jump vector table. Contents of location 0, 1, 2 are to be set in the BIOS. See sample CBIOS, System Alteration Guide, page C-5.

Code to load a pointer to the jump vector table origin into HL, usable in any transient program, is as follows:

```
LHLD    1
```

```
DCX   H
DCX   H
DCX   H
```

The article was otherwise quite informative, as was the issue as a whole.

Yours sincerely,
Dr. Stefan M. Silverston

SAVING 8080 REGISTERS IN 47 BYTES

Dear S. R.,

In the last issue of *Doctor Dobb's* the problem was presented of saving all the 8080 registers in 58 bytes or less. This was assuming that the stack pointer may be pointing to ROM at the time the routine was entered. The first time the solution was attempted the code took only 42 bytes. Either I do not understand completely the question or someone's coding is not very tight. (PSW, BC, DE, HL, & the SP saved with all registers restored at the exit)

Yours truly,
L. Barker

2305 W. Winona
Chicago, Ill. 60625

```
                ; As per Dr. Dobb's #40 page 56.
                ; Save all 8080 registers (NOTE 8080 not Z80)
                ; Assuming stack may be pointing to ROM at entry
                ; 5 flags        bit numbers
                ;          7     6     5     4     3     2     1     0
                ; Flags     S     Z     0     AC    0     P     1     CY
                ;
0000 222D00     START:  SHLD    ADDR+6          ; save HL
0003 EB                 XCHG
0004 222900             SHLD    ADDR+4          ; save DE
0007 210000             LXI     H,0
000A 54                 MOV     D,H
000B 5D                 MOV     E,L
000C D21100             JNC     SKIP
000F 1E01               MVI     E,1
0011 39         SKIP:   DAD     SP              ; HL=SP
0012 222B00             SHLD    ADDR+6          ; save pointer
0015 312900             LXI     SP,ADDR+4
0018 F5                 PUSH    PSW
0019 E1                 POP     H               ; PSW in HL
001A 19                 DAD     D               ; with carry fixed
001B C5                 PUSH    B               ; save BC
001C E5                 PUSH    H               ; save PSW
001D F1                 POP     PSW             ; restore
001E C1                 POP     B
001F D1                 POP     D
0020 E1                 POP     H               ; old stack pointer
0021 F9                 SPHL
0022 2A2D00             LHLD    ADDR+6          ; restore HL
                ;*********************************
0025            ADDR:   DS      10
002F            END
*
```

Program = 37 bytes
Storage = 10 bytes
Total = 47 bytes

VARIATION ON SAVING 8080 STATUS

Dear *Dr. Dobb's*,

One of the more popular topics in *DDJ* recently has been how to save the status of an 8080 if no assumptions can be made about what is in any of the registers. The major problem is to get the stack pointer contents without altering the carry flag. A couple of writers in *DDJ* #41 used the current stack to save one or more registers, but this is not allowed by the conditions of the problem, since the contents of the stack pointer might not be valid. (If the stack pointer is pointing at ROM, at memory-mapped I/O devices, or an non-existent memory, then PUSH PSW followed later by POP PSW will not restore the original condition codes or register A).

Also in *DDJ* #41: Jon Bokelman gives a clever, but overly-complicated solution to the problem. (I will take his word for it that his solution correctly deduces the settings of the N and H flags).

In my solution to the problem, one of two DAD SP instructions is executed, depending on the setting of the carry flag. If the carry is reset when the routine is entered, then the jump in the third instruction is not taken, and the following DAD instruction leaves the carry reset. If the carry is set when the routine is entered, the jump in the third instruction is taken to SAVE1, and the carry is set by the STC instruction that follows the DAD. In either case, when the label SAVE2 is reached, the carry flag is the same as it was when the routine was entered. (All other flags are also the same, since DAD affects only the carry flag). At the end of the routine, all registers have been saved and only HL and the stack pointer contain values that are different from their entry values. The original contents of these registers can be easily restored if desired. (Where was the original statement of the problem and what was its motivation?)

Sincerely,
Jim Howell

5472 Playa Del Rey
San Jose, CA 95123

```
;       ROUTINE TO SAVE THE STATUS OF THE 8080
;       NO ASSUMPTIONS ARE MADE ABOUT THE INITIAL CONTENTS
;         OF ANY OF THE REGISTERS
;
SAVE:   SHLD    HLSAVE                  ; SAVE H AND L
        LXI     H,0                     ; PREPARE TO GET STACK POINTER
        JC      SAVE1                   ; JUMP IF CARRY IS SET
        DAD     SP                      ; GET STACK PTR, LEAVE CARRY RESET
        JMP     SAVE2
;
SAVE1:  DAD     SP                      ; GET STACK PTR
        STC                             ; AND RESTORE CARRY TO 'SET'
;
;       HL NOW CONTAINS INITIAL STACK POINTER
;       THE CARRY FLAG (AND ALL OTHER REGISTERS) ARE THE SAME
;         AS THEY WERE ON ENTRY
;
SAVE2:  LXI     SP,LCLSTK               ; LOCAL STACK
        PUSH    H                       ; SAVE INITIAL STACK POINTER
        PUSH    D                       ; AND OTHER REGISTERS
        PUSH    B
        PUSH    PSW

    ... all registers are now saved ...

        ORG     RAM                     ; SOMEWHERE IN RAM
        DS      20                      ; LOCAL STACK (WITH SOME EXTRA SPACE
LCLSTK  EQU     $
HLSAVE  DS      2
```

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 45 | May 1980 | Volume 5, Issue 5

A REFERENCE JOURNAL FOR USERS OF SMALL COMPUTERS

Special Issue!

The C Programming Language

A Small C Compiler for the 8080's 5
Ron Cain

The C Programming Language 20
D.M. Ritchie, S.C. Johnson, M.E. Lesk, B.W. Kernighan

Structured Programming, C and tiny c 30
Tom Gibson and Scott Guthery

Help with OSI's CP/M 36
Lee Barker

MITS DOS — Disk to Disk Copy Routine 40
Andrew Bender

An 8080 to Z80 Translator System 47
Robert W. Dea

A SMALL C COMPILER FOR THE 8080's

BY RON CAIN
811 Eleventh Ave.
Redwood City, CA 94063

I had to have a compiler for my home computer.

There was no doubt about it: after programming nearly every day in BLISS and then reading about C, PASCAL, LISP, and all the other various languages becoming available, it made no sense at all to continue programming in assembly language.

However, the question arose: Which language?

Surprisingly, the decision was not difficult. Since I do mostly system type programming (editors, music-board drivers, modem talkers, faster versions of LIFE, etc.) rather than application programming (accounting programs, solar eclipse predictors, etc.), the choice boiled down to two: C or BLISS. Besides having the most esoteric names of the group, both possess the strengths of system programming languages. Which is to say they both have a slightly better notion of computer architectures than strictly application oriented languages in terms of accessing bytes and words, and neither wastes time doing "garbage collection" or keeping track of legal array subscripting. Both are intended to produce fairly fast (usually inline) code to get the job done with the least amount of overhead.

After looking around the marketplace, I decided C was the appropriate choice if I expected to produce code usable by others. However, another thing also became obvious: the C compilers available either cost a lot of money, ran only on CP/M, or both. Having neither a lot of money nor CP/M (the second by choice), I had to think of an alternate approach.

The first step came by way of the Tiny-C Interpreter offered by Tiny-C Associates. For a mere $40, I was able to buy the source code of a working C interpreter. I intended to use it to run the C programs I would write until something bigger and better came along. I sent off the money, got the huge white notebook in the mail, and after a couple weeks of typing, had it up and running. Let me be another of many praising the product. The documentation, clarity of print, and ease of implementation of the thing were all exceedingly good.

It gave me a few weeks to evaluate the C language as well as the interpreter. It became obvious after very little time that the language was very easy to use and to follow. It also became obvious that the interpreter, though an excellent implementation, was too slow for my needs. Modem to disk programs like to think in terms of microseconds, not milliseconds, and the former was not to be had with an interpreter. Clearly, a third approach was needed.

The solution was obvious. I had to write the compiler myself. I said obvious, not easy.

So, after gathering together a thick stack of paper and getting ready to produce the (hopefully) last vast 8080 programming effort, I wrote a few lines, and, . . . reconsidered.

Hmmm. What better use of a systems programming language could there be than to write a compiler? And what better language could be used than the one the compiler would ultimately support? Ideas began to gel, and pieces began to fit together. I already had the interpreter. Even though it was slow, it might be just the ticket to bootstrap my way onto the machine. And so it began.

Considering the fact I had never even seen a compiler before and had not done any lengthy programs in C, the thing went together in a remarkably short period of time. Chalk that up to the ease of use of the language. The ability to have variables local to individual routines made the recursive-descent expression parser fit together literally in hours. A few weeks of a couple hours a night saw the compiler begin to take shape. A few obstacles appeared along the way; probably the worst being the point where I discovered I didn't have enough memory for the entire compiler to reside in memory along with the interpreter. One demonic session with the editor deleting all comments (gulp), and shortening all names, brought it to size and solved that problem. The compiler's still recovering from that early trauma. Eventually, there came an evening when all such little problems were solved, and the compiler, running under the interpreter, accepted a small C program and produced an 8080 program.

At that point, I refined the compiler, removed subtle errors, and discovered I still didn't have enough memory for the compiler, the interpreter, and the symbol table a program as large as the compiler itself required. So I asked around, eventually got access to a UNIX system, and spent a couple more weeks refining the compiler without regard to memory or interpreters. This exercise helped remove some of the less obvious syntax errors not caught by the interpreter and necessitated modifying the compiler to accept the true C syntax rather than the slightly-modified syntax used by the interpreter. After this upgrade, the compiler was beginning to took like a true C compiler, and there came a time it ran under UNIX and accepted as input itself.

So, finally, I was able to take the equivalent 8080 code for the compiler back to my machine at home to complete it. The generated code was assembled, run, and once again the com-

piler was submitted to itself. The results of this inbreeding highlighted errors in the generated 8080 code which had thus far been hidden. I set those aright, compiled the compiler once again, then used the ouput to compile the compiler again, and then one more time (sound like an infinite loop?) compiled the compiler. When the code generated by this great-grandson was byte-for-byte identical to the code generated by UNIX way back when, and all subsequent offspring produced such twins, I decided the thing could be considered working.

Once it was working, I decided it would save people in a similar situation a lot of parallel effort if I made the thing available. And if I sold it, I would probably get caught up in supporting it on a number of machines, which was not my intention behind writing it. Giving it away for free seemed like an excellent way to make C available to hobbyists who might otherwise not have been able to get it. What better way to add users to the C community? So here it is.

For those of you who are interested in getting a compiler for a structured language on your home machines, I'd feel flattered if you'd consider the one published here. And if you are the sort who is looking for a systems programming language, you are undoubedly the sort who would modify the thing to fit your own needs. Therefore, a few vital statistics are in order.

In a nutshell, the compiler:

1. Is written in C.
2. Accepts as input a text file written in C.
3. Produces as output a text file of 8080 mnemonics.

The syntax it accepts is a subset of the standard C language. Within this subset, it does not depart from the standard syntax, which means the listing shown here will compile and run under UNIX. Although the subset is limited to make the compiler simpler and will therefore not accept just any pre-existing C program, the programs you can write with it are authentic and will be compatible with more complete compilers.

Right out front, so that UNIX freaks can groan and keep their hopes from getting too high, let me tell you the features of standard C this compiler does NOT support:

1. Floating-point data types.
2. Structures (or unions).
3. Multiple dimension arrays.

The aim was not to support the full C language, but rather to support enough of a subset to be able to create C programs which would be compatible with standard C. Then, as the compiler expanded, more and more features could be added to bring it closer to the real thing.

Currently, the allowable data types are:

1. char—8 bit data element.
2. int—16 bit data element.

Obviously, this means the compiler is an integer-only subset of the language. Which means it will not handle floating-point numbers. Actually, all internal arithmetic operations assume 16-bit integers, meaning 8-bit character elements are sign-extended prior to use.

Allowable modifiers of the two basic types are:

1. type *name—declares name to be a pointer to an element of the specified type.
2. type name [ ] —syntactically identical to the above pointer declaration.
3. type name[constant] —declares an array of "constant" size where each array element is of the specified type.

If you've worked with C before, you know just about everything is done with pointers. It lets you exploit the architecture of the cpu by giving access to all addressable memory. Unlike standard C, you can't use more than one modifier per declaration, meaning it will not accept something like "int (*name) [ ]". This does not present a terrible restriction, but it must be mentioned. Since no runtime checking is made on the legality of pointer usage, it is a trivial matter to access any byte in memory, and a not-at-all-trivial problem finding which routine is clobbering some random location. You can see why pointers have been lumped with "goto's" as the bane of the code maintainers and modifiers.

Allowable primary expressions are:

1. **A local or global variable name.** Any name which has been declared previously may be used as an expression. The name can be any length, but only the first 8 characters are used by the compiler. Presently, variable names are not folded to upper-case prior to being used in the output file. Therefore, the variable "x" would expand in the expression "x = 0" to:
   ```
   LXI H,0
   SHLD x
   ```
   If the assembler you intend to use cannot handle lower-case names, do not use lower-case names in your C programs.
2. **Constants.** These can be either:
   a. A decimal number.
   b. A single or pair of ASCII characters enclosed in single quotes, such as 'a' or 'TX'.
   c. A string enclosed in double quotes, such as "this is a string". The value such a constant yields is a pointer to the first character of the string which the compiler stores in memory.
3. **Function calls.** These are defined as any expression followed by an open paren. Thus, a function can be to a named routine, such as "print()", or to the results of some expression, such as "1000()" (which calls location 1000 decimal), or "array[i]()" which calls the location whose value is found in array[i]. Functions always return a value in HL, though it does not have to be used. Arguments to functions appear between the parentheses and are pushed onto the stack in the order they appear. Hence, "print(x,y,z);" would effectively do:
   ```
   PUSH x
   PUSH y
   PUSH z
   CALL print
   ```
   Any number of arguments may be passed.
4. **Subscripted elements.** Either an array name or a pointer may be subscripted to refer to the appropriate element. Subscripts are assumed to start from zero. Therefore, legal

expressions are:

array [0] – the first element in array.
array [x + 31] – the element at the address given by adding x to 31 and then to array.
pointer [i] – the element at the address given by adding i to the contents of pointer.

Only single dimensions are allowed. Subscripting either an integer array or a pointer to an integer will cause the subscript expression to be doubled. Therefore, if you declare "int *ptr", the expression "ptr [3]" refers to the element at ptr+6.

Allowable unary expression operators are:

1. "-" – forms the two's complement of the expression (minus).
2. "*" – refers to the element pointed to by the expression (providing the expression is a pointer).
3. "&" – evaluates the address of the given expression, providing it has one. Hence, &count yields the address of the element "count". &1000 is an error.
4. "++" – increments the expression by one. If this appears before the expression, it increments before using it. If it appears after it, it will increment it after. Only lvalues (expressions which can appear on the left-hand side of an equal sign) are allowed. Hence, assuming "count" contains a 5, ++count would evaluate to a 6, and "count" would contain a 6. Count++ would evaluate to a 5, and count would contain a 6. 1000++ is illegal. If this operator is applied to an integer pointer, it will increment by 2.
5. "--" – decrements the expression by one. This works just like ++ but subtracts one rather than adding.

Allowable binary operators are:

1. "+" – adds the two expressions (i.e. count + total)
2. "-" – subtracts the two expressions.
3. "*" – multiplies the two expressions.
4. "/" – divides the first expression by the second.
5. "%" – yields the remainder after dividing the first expression by the second (modulo).
6. "|" – yields the logical inclusive "or" of the two expressions.
7. "^" – yields the logical exclusive "or".
8. "&" – yields the logical "and".
9. "=" – assigns the value of the expression on the right to the one on the left. Since evaluation is done right to left in this case, syntaxes like:

   x = y = z = 0;

   are legal.

Comparison operators compare two expressions and yield either a zero or a one depending whether the result of the compare is false or true, respectively. These include:

1. "==" – tests for equality.
2. "!=" – tests for inequality.
3. "<" – tests for less than.
4. ">" – tests for greater than.
5. "<=" – tests for less than or equal to.

Comparisons involving a pointer (which is an address) are done as unsigned compares. All other compares are signed.

Program control is accomplished by the following allowable statements:

1. **expression**; An expression, no matter how complex, is considered a simple statement.
2. **if (expression) statement**; If the expression is nonzero, the statement is executed, otherwise it isn't.
3. **if (expression) statement; else statement**; This form of the "if" statement allows the "else" clause. As is the case with most "dangling else" ambiguities, all "else's" pair with the nearest unmatched "if".
4. **while (expression) statement**; The statement is performed until the expression becomes zero. Since the test is made before the statement is executed the first time, it need not be executed at all.
5. **break**; This statement will cause control to be transferred out of the inner-most "while" loop.
6. **continue**; This statement, used within a "while" loop, will transfer control back to the top of the loop.
7. **return**; This statement does an immediate return from the current function. If a function does not end with this statement, one is performed regardless.
8. **return expression**; This statement allows a function to return a value explicitly.
9. ; A semicolon by itself is considered a null statement which does nothing but take the place of a statement. You see this in forms such as: "while (*iptr++ = *jptr++) ;" where the test itself contains all the necessary parts of the statement.
10. { **statement; statement;** . . . ; **statement;**} The use of curly brackets (" { } ") around any group of simple statements is considered a compound statement. A compound statement can be used anywhere a simple statement can. For example:

```
while (1) {x = 3; y = 10; funct(33);}
or
if (x < y)
  { print(x);
   total(x);
   --x;
  }

else
  { type("all done");
  x = y;
  }
```

The compiler also accepts some pseudo-op type statements. These are:

1. #**include filename** Anywhere this statement appears in the program, the indicated filename will be opened and inserted. The "included" file may not contain an "#include" statement.
2. #**define name string** This statement will cause the given name to be replaced by the string throughout the entire program. Normally, it is used to define constants, such as:

   #define tablesize 1000
   #define maxlength 8

   But it can also be used for any sort of text:

   # define jprint 3crs print (12); print(12); print(12);

The replacement is purely on a text level, and error checking will be performed only after the replacement.

3. **#asm ... #endasm** This structure is not supported by standard C, but it was a feature I felt I needed. It may appear anywhere a statement would, but it passes everything between the word "#asm" and the word "#endasm" right through the parser without intervention. It is intended to be used to pass assembly language code through the parsing mechanism. Since it counts as a single statement, allowable (and expected) forms are:

```
if (x < y)
    #asm
    LHLD TOTAL
    CALL ADD
    CNC ERROR
    #endasm
else return;
```

This pseudo-op conceivably allows an entire assembly language program to be passed through the compiler. Its intent is to allow machine dependent features (like the 8080's "IN" and "OUT" instructions to be used without writing separate programs).

In parsing the input, the compiler never tries to be overly clever, which simplifies the compiler quite a bit and makes the generated code quite predictable, if somewhat redundant.

The input file is scanned character-by-character left to right. Only a single 80 character line at a time is buffered. This reduces the amount of memory used by the compiler and makes any kind of look-ahead improbable. The expression parser (namely the routines heir1() through heir11() and primary()) uses the method of recursive-descent to evaluate an expression, which means the routines call themselves when necessary (such as encountering an open paren). All expressions result in a value in the HL register pair. Character values are sign-extended through the H register. This means the line of code:

```
x = 5;
```

generates the code:

```
LXI H,5
SHLD x
```

Notice, HL still contains the 5. The compiler relies upon this convention in the forms:

```
x = y = z = 0;
```

which generates the code:

```
LXI H,0
SHLD x
SHLD y
SHLD z
```

In fact, using an expression as a statement is a good way to load the HL pair using the compiler. The expression:

```
100;
```

puts 100 into the HL pair. Of more use might be the expression:

```
&local;
```

which would load the address of the variable "local" into HL.

Whenever a binary operator is seen, and a second value is needed, the HL pair is pushed onto the stack, and the second operand is loaded into HL. Therefore, the expression:

```
x = x + 5;
```

would generate the code:

```
LHLD x
PUSH H
LXI H,5
POP D
DAD D
SHLD x
```

Obviously, this shows the excessive caution used by the compiler. It need not actually have pushed the HL pair, but could merely have put it into DE with an "XCHG" instruction. Except for the one fact the compiler had no idea how involved the second expression would be. Consider:

```
x = x + (5 * x) / funct(x * 3);
```

Obviously, in such a case, pushing H would be wise. The compiler makes no distinctions, but generates bullet-proof, rather than optimal, code.

Global variables (those declared outside of a function) are assigned storage and can be referenced by name. This means assembly language code can refer to the global variables using the same name as the C program. Local variables, however, like function arguments, occupy a spot on the stack, and are referenced by some offset from the current stack pointer. The simple routine:

```
main(arg)
    char arg;
    {int localword;
    localword = arg;
    }
```

would generate the code:

```
main:
    PUSH B
    LXI H,0
    DAD SP
    PUSH H
    LXI H,4
    DAD SP
    CALL ccgchar
    POP D
    CALL ccpint
    POP B
    RET
```

Notice a few important items. First, the address of "localword" is evaluated as it is encountered. Then it is pushed, since HL must be used for something else. The address of "arg" is evaluated, and an outside routine "ccgchar" is called to read one byte from the specified address and to sign-extend it. Then the address of "localword" is popped, and another routine "ccpint" is called to store a 16-bit value in HL at the address in DE. This means there is a runtime library needed for the C programs produced by this compiler. It is not listed here, since it is as long as the compiler itself (it is in 8080 code) and all the routines seemed fairly obvious, since they are all invoked by the code generating routines at the end of the compiler. If there is enough response, perhaps Dr. Dobbs can print that too one day.

I guess the biggest question remaining is "How can I get it on my system?"

Good question, and I'm afraid you know the answer better than I do.

Obviously, the C listing printed here is not directly usable on your home system. If you had access to a UNIX system,

you could simply type it in as shown, compile it with UNIX's C compiler, and have a running version. At that point, you could run the compiler under UNIX, submit it to itself for compilation, and the output would be a copy of the compiler in 8080 mnemonics. That output is directly usable on any 8080 machine with some mass-storage and an assembler. This is just what I did in bootstrapping the compiler, so naturally I have all the intermediate files on floppy disk.

However, I am not a distributor. I can't handle, nor do I want to handle, requests to put the files onto some other media. I originally released the first version of the compiler some months ago and have not been able to deal with even the moderate number of requests.

It is my hope some of you will either go to the effort yourselves or get some distributor to go to the trouble of getting the files onto the necessary floppies or cassettes. Such group efforts I will assist where possible.

Already, a couple of distributors or private parties have the thing working on other machines or are planning it.

Walt Bilofsky, Software Consultant, 14478 Glorietta Drive, Sherman Oaks, Ca. 91423, has been in on this thing from early on, and has an extremely improved version running on his Heath-kit. His compiler supports much more of the standard C language, and his runtime library sports the ability to do I/O redirection like the UNIX shell. I am told he is considering a CP/M version, and will eventually support others. What is published here is free, if difficult to transport directly to your system. But for a small fee, I'm sure he'll be able to supply you with a working compiler.

Also, The Code Works has obtained a version and was considering making it work on other machines. Their address is Box 550, Goleta, CA 93017.

Which brings up interesting points. If you haven't already made the intuitive leap about the power behind writing the compiler in the same language it supports, it lies in the ability to compile itself. This means a user with extra memory can add additional features to the basic compiler, compile it with the old compiler, and voila! A new and more powerful version will exist.

If you have a working compiler on one cpu and want to bootstrap your way to another kind of processor, you need only change the code generating portions of the compiler (all grouped into the final section of the listing) to make code for the new machine, compile that compiler, and once again, you have a new beasty. Sort of like cloning.

Personally, I've developed the thing about as far as I need to begin using it for the things I originally intended. After all, I did write it for a reason. However, I am still interested in hearing what modifications are found useful, what machines it eventually wanders onto, and any other interesting paths this thing takes.

If you get this thing running on your home system, make mods to it, make it run on another cpu, or are in a position to make copies of your work for others with similar cpu's, I would appreciate hearing about it in Dr. Dobb's.

I thing it's an excellent opportunity to learn how a compiler works and at the same time establish the necessary groundwork for a C community. It's already here, with the Tiny-C interpreter and the other C compilers now available. I hope to see alot more C programs in the future.

PROGRAM

```
/*************************************************/
/*                                               */
/*              small-c compiler                 */
/*                  rev. 1.1                     */
/*                by Ron Cain                    */
/*                                               */
/*************************************************/

/*      Define system dependent parameters       */

/*      Stand-alone definitions                  */

/* #define NULL 0       */
/* #define eol 13       */
/*      UNIX definitions (if not stand-alone)    */

#include <stdio.h>
#define eol 10
/*      Define the symbol table parameters       */

#define symsiz  14
#define symtbsz 5040
#define numglbs 300
#define startglb symtab
#define endglb  startglb+numglbs*symsiz
#define startloc endglb+symsiz
#define endloc  symtab+symtbsz-symsiz

/*      Define symbol table entry format         */

#define name    0
#define ident   9
#define type    10
#define storage 11
#define offset  12

/*      System wide name size (for symbols)      */

#define namesize 9
#define namemax  8

/*      Define possible entries for "ident"      */

#define variable 1
#define array   2
#define pointer 3
#define function 4

/*      Define possible entries for "type"       */
#define cchar   1
#define cint    2

/*      Define possible entries for "storage"    */

#define statik  1
#define stkloc  2

/*      Define the "while" statement queue       */

#define wqtabsz 100
#define wqsiz   4
#define wqmax   wq+wqtabsz-wqsiz

/*      Define entry offsets in while queue      */

#define wqsym   0
#define wqsp    1
#define wqloop  2
#define wqlab   3

/*      Define the literal pool                  */

#define litabsz 2000
#define litmax  litabsz-1

/*      Define the input line                    */

#define linesize 80
#define linemax linesize-1
#define mpmax   linemax

/*      Define the macro (define) pool           */

#define macqsize 1000
#define macmax  macqsize-1

/*      Define statement types (tokens)          */

#define stif    1
#define stwhile 2
#define streturn 3
#define stbreak 4
#define stcont  5
#define stasm   6
#define stexp   7

/*      Now reserve some storage words           */
```

```
char    symtab[symtbsz];        /* symbol table */
char    *glbptr,*locptr;                /* ptrs to next entries */
int     wq[wqtabsz];            /* while queue */
int     *wqptr;                 /* ptr to next entry */

char    litq[litabsz];          /* literal pool */
int     litptr;                 /* ptr to next entry */

char    macq[macqsize];         /* macro string buffer */
int     macptr;                 /* and its index */

char    line[linesize];         /* parsing buffer */
char    mline[linesize];        /* temp macro buffer */
int     lptr,mptr;              /* ptrs into each */

/*      Misc storage    */

int     nxtlab,         /* next avail label # */
        litlab,         /* label # assigned to literal pool */
        sp,             /* compiler relative stk ptr */
        argstk,         /* function arg sp */
        ncmp,           /* # open compound statements */
        errcnt,         /* # errors in compilation */
        eof,            /* set non-zero on final input eof */
        input,          /* iob # for input file */
        output,         /* iob # for output file (if any) */
        input2,         /* iob # for "include" file */
        glbflag,        /* non-zero if internal globals */
        ctext,          /* non-zero to intermix c-source */
        cmode,          /* non-zero while parsing c-code */
                        /* zero when passing assembly code */
        laststs;        /* last executed statement type */

char    quote[2];       /* literal string for "" */
char    *cptr;          /* work ptr to any char buffer */
int     *iptr;          /* work ptr to any int buffer */
/*      >>>>> start cc1 <<<<<<          */
/*                                      */
/*      Compiler begins execution here  */
/*                                      */
main()
        {
        glbptr=startglb;        /* clear global symbols */
        locptr=startloc;        /* clear local symbols */
        wqptr=wq;               /* clear while queue */
        macptr=         /* clear the macro pool */
        litptr=         /* clear literal pool */
        sp =            /* stack ptr (relative) */
        errcnt=         /* no errors */
        eof=            /* not eof yet */
        input=          /* no input file */
        input2=         /* or include file */
        output=         /* no open units */
        ncmp=           /* no open compound states */
        laststs= /* no last statement yet */
        quote[1]=
        0;              /*  ...all set to zero.... */
        quote[0]='"';           /* fake a quote literal */
        cmode=1;        /* enable preprocessing */
        /*                              */
        /*      compiler body           */
        /*                              */
        ask();                  /* get user options */
        openout();              /* get an output file */
        openin();               /* and initial input file */
        header();               /* intro code */
        parse();                /* process ALL input */
        dumplits();             /* then dump literal pool */
        dumpglbs();             /* and all static memory */
        errorsummary();         /* summarize errors */
        trailer();              /* follow-up code */
        closeout();             /* close the output (if any) */
        return;                 /* then exit to system */
        }
/*                                      */
/*      Process all input text          */
/*                                      */
/* At this level, only static declarations, */
/*      defines, includes, and function */
/*      definitions are legal...        */
parse()
        {
        while (eof==0)          /* do until no more input */
                {
                if(amatch("char",4)){declglb(cchar);ns();}
                else if(amatch("int",3)){declglb(cint);ns();}
                else if(match("#asm"))doasm();
                else if(match("#include"))doinclude();
                else if(match("#define"))addmac();
                else newfunc();
                blanks();       /* force eof if pending */
                }
        }
/*                                      */
/*      Dump the literal pool           */
/*                                      */
dumplits()
        {int j,k;
        if (litptr==0) return;  /* if nothing there, exit...*/
        printlabel(litlab);col(); /* print literal label */
        k=0;                    /* init an index... */
        while (k<litptr)        /*      to loop with */
                {defbyte();     /* pseudo-op to define byte */
                j=10;           /* max bytes per line */
                while(j--)
                        {outdec((litq[k++]&127));
                        if ((j==0) | (k>=litptr))
                                {nl();          /* need <cr> */
                                break;
                                }
                        outbyte(',');   /* separate bytes */
                        }
                }
        }
/*                                      */
/*      Dump all static variables       */
/*                                      */
dumpglbs()
        {
        int j;
        if(glbflag==0)return;   /* don't if user said no */
        cptr=startglb;
        while(cptr<glbptr)
                {if(cptr[ident]!=function)
                        /* do if anything but function */
                        {outstr(cptr);col();
                                /* output name as label... */
                        defstorage();   /* define storage */
                        j=((cptr[offset]&255)+
                                ((cptr[offset+1]&255)<<8));
                                        /* calc # bytes */
                        if((cptr[type]==cint)|
                                (cptr[ident]==pointer))
                                j=j+j;
                        outdec(j);      /* need that many */
                        nl();
                        }
                cptr=cptr+symsiz;
                }
        }
/*                                      */
/*      Report errors for user          */
/*                                      */
errorsummary()
        {
        /* see if anything left hanging... */
        if (ncmp) error("missing closing bracket");
                /* open compound statement ... */
        nl();
        comment();
        outdec(errcnt); /* total # errors */
        outstr(" errors in compilation.");
        nl();
        }
/*                                      */
/*      Get options from user           */
/*                                      */
ask()
        {
        int k,num[1];
        kill();                 /* clear input line */
        outbyte(12);            /* clear the screen */
        nl();nl();nl();         /* print banner */
        pl("   * * *  small-c compiler  * * *");
        nl();
        pl("           by Ron Cain");
        nl();nl();
        /* see if user wants to interleave the c-text */
        /*      in form of comments (for clarity) */
        pl("Do you wish the c-text to appear? ");
        gets(line);             /* get answer */
        ctext=0;                /* assume no */
        if((ch()=='Y')|(ch()=='y'))
                ctext=1;        /* user said yes */
        /* see if user wants us to allocate static */
        /*  variables by name in this module    */
        /*      (pseudo external capability)    */
        pl("Do you wish the globals to be defined? ");
        gets(line);
        glbflag=0;
        if((ch()=='Y')|(ch()=='y'))
                glbflag=1;      /* user said yes */
        /* get first allowable number for compiler-generated */
        /*      labels (in case user will append modules) */
        while(1)
                {pl("Starting number for labels? ");
                gets(line);
                if(ch()==0){num[0]=0;break;}
                if(k=number(num))break;
                }
        nxtlab=num[0];
        litlab=getlabel();      /* first label=literal pool */
        kill();                 /* erase line */
        }
/*                                      */
/*      Get output filename             */
/*                                      */
openout()
        {
        kill();                 /* erase line */
        output=0;               /* start with none */
        pl("Output filename? "); /* ask...*/
        gets(line);     /* get a filename */
        if(ch()==0)return;      /* none given... */
        if((output=fopen(line,"w"))==NULL) /* if given, open */
                {output=0;      /* can't open */
                error("Open failure");
```

```
                }
        kill();                 /* erase line */
}
/*                                      */
/*      Get (next) input file           */
/*                                      */
openin()
{
        input=0;                /* none to start with */
        while (input==0)        /* any above 1 allowed */
                {kill();        /* clear line */
                if(eof)break;   /* if user said none */
                pl("Input filename? ");
                gets(line);     /* get a name */
                if(ch()==0)
                        {eof=1;break;} /* none given... */
                if((input=fopen(line,"r"))==NULL)
                        {input=0;       /* can't open it */
                        pl("Open failure");
                        }
                }
        kill();         /* erase line */
        }
/*                                      */
/*      Open an include file            */
/*                                      */
doinclude()
{
        blanks();       /* skip over to name */
        if((input2=fopen(line+lptr,"r"))==NULL)
                {input2=0;
                error("Open failure on include file");
                }
        kill();         /* clear rest of line */
                        /* so next read will come from */
                        /* new file (if open */
}
/*                                      */
/*      Close the output file           */
/*                                      */
closeout()
{       if(output)fclose(output); /* if open, close it */
        output=0;               /* mark as closed */
}
/*                                      */
/*      Declare a static variable       */
/*        (i.e. define for use)         */
/*                                      */
/* makes an entry in the symbol table so subsequent */
/*  references can call symbol by name  */
declglb(typ)            /* typ is cchar or cint */
        int typ;
{       int k,j;char sname[namesize];
        while(1)
                {while(1)
                        {if(endst())return;     /* do line */
                        k=1;            /* assume 1 element */
                        if(match("*"))  /* pointer ? */
                                j=pointer;      /* yes */
                                else j=variable; /* no */
                         if (symname(sname)==0) /* name ok? */
                                illname(); /* no... */
                        if(findglb(sname)) /* already there? */
                                multidef(sname);
                        if (match("["))         /* array? */
                                {k=needsub();   /* get size */
                                if(k)j=array;   /* !0=array */
                                else j=pointer; /* 0=ptr */
                                }
                        addglb(sname,j,typ,k); /* add symbol */
                        break;
                        }
                if (match(",")==0) return; /* more? */
                }
        }
/*                                      */
/*      Declare local variables         */
/*      (i.e. define for use)           */
/*                                      */
/* works just like "declglb" but modifies machine stack */
/*      and adds symbol table entry with appropriate */
/*      stack offset to find it again                */
declloc(typ)            /* typ is cchar or cint */
        int typ;
        {
        int k,j;char sname[namesize];
        while(1)
                {while(1)
                        {if(endst())return;
                        if(match("*"))
                                j=pointer;
                                else j=variable;
                        if (symname(sname)==0)
                                illname();
                        if(findloc(sname))
                                multidef(sname);
                        if (match("["))
                                {k=needsub();
                                if(k)
                                        {j=array;
                                        if(typ==cint)k=k+k;
                                        }
                                else
                                        {j=pointer;
                                        k=2;
                                        }
                                }
                        else
                                if((typ==cchar)
                                        &(j!=pointer))
                                        k=1;else k=2;
                        /* change machine stack */
                        sp=modstk(sp-k);
                        addloc(sname,j,typ,sp);
                        break;
                        }
                if (match(",")==0) return;
                }
        }
/*      >>>>>> start of cc2 <<<<<<<<    */
/*                                      */
/*      Get required array size         */
/*                                      */
/* invoked when declared variable is followed by "[" */
/*      this routine makes subscript the absolute */
/*      size of the array. */
needsub()
        {
        int num[1];
        if(match("]"))return 0; /* null size */
        if (number(num)==0)     /* go after a number */
                {error("must be constant");     /* it isn't */
                num[0]=1;               /* so force one */
                }
        if (num[0]<0)
                {error("negative size illegal");
                num[0]=(-num[0]);
                }
        needbrack("]");         /* force single dimension */
        return num[0];          /* and return size */
        }
/*                                      */
/*      Begin a function                */
/*                                      */
/* Called from "parse" this routine tries to make a function */
/*      out of what follows.    */
newfunc()
        {
        char n[namesize],*ptr;
        if (symname(n)==0)
                {error("illegal function or declaration");
                kill(); /* invalidate line */
                return;
                }
        if(ptr=findglb(n))      /* already in symbol table ? */
                {if(ptr[ident]!=function)multidef(n);
                        /* already variable by that name */
                else if(ptr[offset]==function)multidef(n);
                        /* already function by that name */
                else ptr[offset]=function;
                        /* otherwise we have what was earlier*/
                        /*  assumed to be a function */
                }
        /* if not in table, define as a function now */
        else addglb(n,function,cint,function);
        /* we had better see open paren for args... */
        if(match("(")==0)error("missing open paren");
        outstr(n);col();nl();   /* print function name */
        argstk=0;               /* init arg count */
        while(match(")")==0)    /* then count args */
                /* any legal name bumps arg count */
                {if(symname(n))argstk=argstk+2;
                else{error("illegal argument name");junk();}
                blanks();
                /* if not closing paren, should be comma */
                if(streq(line+lptr,")")==0)
                        {if(match(",")==0)
                        error("expected comma");
                        }
                if(endst())break;
                }
        locptr=startloc;        /* "clear" local symbol table*/
        sp=0;                   /* preset stack ptr */
        while(argstk)
                /* now let user declare what types of things */
                /*      those arguments were */
                {if(amatch("char",4)){getarg(cchar);ns();}
                else if(amatch("int",3)){getarg(cint);ns();}
                else{error("wrong number args");break;}
                }
        if(statement()!=streturn) /* do a statement, but if */
                                  /* it's a return, skip */
                                  /* cleaning up the stack */
                {modstk(0);
                ret();
                }
        sp=0;                   /* reset stack ptr again */
        locptr=startloc;        /* deallocate all locals */
        }
/*                                      */
/*      Declare argument types          */
/*                                      */
/* called from "newfunc" this routine adds an entry in the */
/*      local symbol table for each named argument */
getarg(t)               /* t = cchar or cint */
        int t;
        {
        char n[namesize],c;int j;
```

```
        while(1)
                {if(argstk==0)return;   /* no more args */
                if(match("*"))j=pointer;
                        else j=variable;
                if(symname(n)==0) illname();
                if(findloc(n))multidef(n);
                if(match("["))  /* pointer ? */
                /* it is a pointer, so skip all */
                /* stuff between "[]" */
                        {while(inbyte()!=']')
                                if(endst())break;
                        j=pointer;
                        /* add entry as pointer */
                        }
                addloc(n,j,t,argstk);
                argstk=argstk-2;        /* cnt down */
                if(endst())return;
                if(match(",")==0)error("expected comma");
                }
        }
/*                                      */
/*      Statement parser                */
/*                                      */
/* called whenever syntax requires      */
/*      a statement.                    */
/*  this routine performs that statement */
/*  and returns a number telling which one */
statement()
 {      if ((ch()==0) & (eof)) return;
        else if(amatch("char",4))
                {decl loc(cchar);ns();}
        else if(amatch("int",3))
                {decl loc(cint);ns();}
        else if(match("{"))compound();
        else if(amatch("if",2))
                {doif();laststst=stif;}
        else if(amatch("while",5))
                {dowhile();laststst=stwhile;}
        else if(amatch("return",6))
                {doreturn();ns();laststst=streturn;}
        else if(amatch("break",5))
                {dobreak();ns();laststst=stbreak;}
        else if(amatch("continue",8))
                {docont();ns();laststst=stcont;}
        else if(match(";"));
        else if(match("#asm"))
                {doasm();laststst=stasm;}
        /* if nothing else, assume it's an expression */
        else{expression();ns();laststst=stexp;}
        return laststst;
}
/*                                      */
/*      Semicolon enforcer              */
/*                                      */
/* called whenever syntax requires a semicolon */
ns()    {if(match(";")==0)error("missing semicolon");}
/*                                      */
/*      Compound statement              */
/*                                      */
/* allow any number of statements to fall between "{}" */
compound()
        {
        ++ncmp;         /* new level open */
        while(match("}")==0)
                if(eof) return;
                else statement();
        --ncmp;         /* close current level */
        }
/*                                      */
/*              "if" statement          */
/*                                      */
doif()
        {
        int flev,fsp,flab1,flab2;
        flev=locptr;    /* record current local level */
        fsp=sp;         /* record current stk ptr */
        flab1=getlabel(); /* get label for false branch */
        test(flab1);    /* get expression, and branch false */
        statement();    /* if true, do a statement */
        sp=modstk(fsp); /* then clean up the stack */
        locptr=flev;    /* and deallocate any locals */
        if (amatch("else",4)==0)        /* if...else ? */
                /* simple "if"...print false label */
                {printlabel(flab1);col();nl();
                return;         /* and exit */
                }
        /* an "if...else" statement. */
        jump(flab2=getlabel()); /* jump around false code */
        printlabel(flab1);col();nl();   /* print false label */
        statement();            /* and do "else" clause */
        sp=modstk(fsp);         /* then clean up stk ptr */
        locptr=flev;            /* and deallocate locals */
        printlabel(flab2);col();nl();   /* print true label */
        }
/*                                      */
/*      "while" statement               */
/*                                      */
dowhile()
        {
        int wq[4];              /* allocate local queue */
        wq[wqsym]=locptr;       /* record local level */
        wq[wqsp]=sp;            /* and stk ptr */
        wq[wqloop]=getlabel();  /* and looping label */
        wq[wqlab]=getlabel();   /* and exit label */
        addwhile(wq);           /* add entry to queue */
                                /* (for "break" statement) */
        printlabel(wq[wqloop]);col();nl(); /* loop label */
        test(wq[wqlab]);        /* see if true */
        statement();            /* if so, do a statement */
        jump(wq[wqloop]);       /* loop to label */
        printlabel(wq[wqlab]);col();nl(); /* exit label */
        locptr=wq[wqsym];       /* deallocate locals */
        sp=modstk(wq[wqsp]);    /* clean up stk ptr */
        delwhile();             /* delete queue entry */
        }
/*                                      */
/*      "return" statement              */
/*                                      */
doreturn()
        {
        /* if not end of statement, get an expression */
        if(endst()==0)expression();
        modstk(0);      /* clean up stk */
        ret();          /* and exit function */
        }
/*                                      */
/*      "break" statement               */
/*                                      */
dobreak()
        {
        int *ptr;
        /* see if any "whiles" are open */
        if ((ptr=readwhile())==0) return;       /* no */
        modstk((ptr[wqsp]));    /* else clean up stk ptr */
        jump(ptr[wqlab]);       /* jump to exit label */
        }
/*                                      */
/*      "continue" statement            */
/*                                      */
docont()
        {
        int *ptr;
        /* see if any "whiles" are open */
        if ((ptr=readwhile())==0) return;       /* no */
        modstk((ptr[wqsp]));    /* else clean up stk ptr */
        jump(ptr[wqloop]);      /* jump to loop label */
        }
/*                                      */
/*      "asm" pseudo-statement          */
/*                                      */
/* enters mode where assembly language statement are */
/*      passed intact through parser    */
doasm()
        {
        cmode=0;                /* mark mode as "asm" */
        while (1)
                {inline();      /* get and print lines */
                if (match("#endasm")) break;    /* until... */
                if(eof)break;
                outstr(line);
                nl();
                }
        kill();         /* invalidate line */
        cmode=1;                /* then back to parse level */
        }
/*      >>>>> start of cc3 <<<<<<<<<<   */

/*                                      */
/*      Perform a function call         */
/*                                      */
/* called from heirll, this routine will either call */
/*      the named function, or if the supplied ptr is */
/*      zero, will call the contents of HL              */
callfunction(ptr)
        char *ptr;      /* symbol table entry (or 0) */
{       int nargs;
        nargs=0;
        blanks();       /* already saw open paren */
        if(ptr==0)push();       /* calling HL */
        while(streq(line+lptr,")")==0)
                {if(endst())break;
                expression();   /* get an argument */
                if(ptr==0)swapstk(); /* don't push addr */
                push(); /* push argument */
                nargs=nargs+2;  /* count args*2 */
                if (match(",")==0) break;
                }
        needbrack(")");
        if(ptr)call(ptr);
        else callstk();
        sp=modstk(sp+nargs);    /* clean up arguments */
}
junk()
{       if(an(inbyte()))
                while(an(ch()))gch();
        else while(an(ch())==0)
                {if(ch()==0)break;
                gch();
                }
        blanks();
}
endst()
{       blanks();
        return ((streq(line+lptr,";")|(ch()==0)));
}
illname()
{       error("illegal symbol name");junk();}
multidef(sname)
        char *sname;
{       error("already defined");
```

```
        comment();
        outstr(sname);nl();
}
needbrack(str)
        char *str;
{       if (match(str)==0)
                {error("missing bracket");
                comment();outstr(str);nl();
                }
}
needlval()
{       error("must be lvalue");
}
findglb(sname)
        char *sname;
{       char *ptr;
        ptr=startglb;
        while(ptr!=glbptr)
                {if(astreq(sname,ptr,namemax))return ptr;
                ptr=ptr+symsiz;
                }
        return 0;
}
findloc(sname)
        char *sname;
{       char *ptr;
        ptr=startloc;
        while(ptr!=locptr)
                {if(astreq(sname,ptr,namemax))return ptr;
                ptr=ptr+symsiz;
                }
        return 0;
}
addglb(sname,id,typ,value)
        char *sname,id,typ;
        int value;
{       char *ptr;
        if(cptr=findglb(sname))return cptr;
        if(glbptr>=endglb)
                {error("global symbol table overflow");
                return 0;
                }
        cptr=ptr=glbptr;
        while(an(*ptr++ = *sname++));   /* copy name */
        cptr[ident]=id;
        cptr[type]=typ;
        cptr[storage]=statik;
        cptr[offset]=value;
        cptr[offset+1]=value>>8;
        glbptr=glbptr+symsiz;
        return cptr;
}
addloc(sname,id,typ,value)
        char *sname,id,typ;
        int value;
{       char *ptr;
        if(cptr=findloc(sname))return cptr;
        if(locptr>=endloc)
                {error("local symbol table overflow");
                return 0;
                }
        cptr=ptr=locptr;
        while(an(*ptr++ = *sname++));   /* copy name */
        cptr[ident]=id;
        cptr[type]=typ;
        cptr[storage]=stkloc;
        cptr[offset]=value;
        cptr[offset+1]=value>>8;
        locptr=locptr+symsiz;
        return cptr;
}
/* Test if next input string is legal symbol name */
symname(sname)
        char *sname;
{       int k;char c;
        blanks();
        if(alpha(ch())==0)return 0;
        k=0;
        while(an(ch()))sname[k++]=gch();
        sname[k]=0;
        return 1;
        }
/* Return next avail internal label number */
getlabel()
{       return(++nxtlab);
}
/* Print specified number as label */
printlabel(label)
        int label;
{       outstr("cc");
        outdec(label);
}
/* Test if given character is alpha */
alpha(c)
        char c;
{       c=c&127;
        return(((c>='a')&(c<='z'))|
                ((c>='A')&(c<='Z'))|
                (c=='_'));
}
/* Test if given character is numeric */
numeric(c)
        char c;
{       c=c&127;
        return((c>='0')&(c<='9'));
}
/* Test if given character is alphanumeric */
an(c)
        char c;
{       return((alpha(c))|(numeric(c)));
}
/* Print a carriage return and a string only to console */
pl(str)
        char *str;
{       int k;
        k=0;
        putchar(eol);
        while(str[k])putchar(str[k++]);
}
addwhile(ptr)
        int ptr[];
 {
        int k;
        if (wqptr==wqmax)
                {error("too many active whiles");return;}
        k=0;
        while (k<wqsiz)
                {*wqptr++ = ptr[k++];}
}
delwhile()
        {if(readwhile()) wqptr=wqptr-wqsiz;
        }
readwhile()
 {
        if (wqptr==wq){error("no active whiles");return 0;}
        else return (wqptr-wqsiz);
 }
ch()
{       return(line[lptr]&127);
}
nch()
{       if(ch()==0)return 0;
                else return(line[lptr+1]&127);
}
gch()
{       if(ch()==0)return 0;
                else return(line[lptr++]&127);
}
kill()
{       lptr=0;
        line[lptr]=0;
}
inbyte()
{
        while(ch()==0)
                {if (eof) return 0;
                inline();
                preprocess();
                }
        return gch();
}
inchar()
{
        if(ch()==0)inline();
        if(eof)return 0;
        return(gch());
}
inline()
 {
        int k,unit;
        while(1)
                {if (input==0)openin();
                if(eof)return;
                if((unit=input2)==0)unit=input;
                kill();
                while((k=getc(unit))>0)
                        {if((k==eol)|(lptr>=linemax))break;
                        line[lptr++]=k;
                        }
                line[lptr]=0;   /* append null */
                if(k<=0)
                        {fclose(unit);
                        if(input2)input2=0;
                                else input=0;
                        }
                if(lptr)
                        {if((ctext)&(cmode))
                                {comment();
                                outstr(line);
                                nl();
                                }
                        lptr=0;
                        return;
                        }
                }
}
/*      >>>>>> start of cc4 <<<<<<<     */

keepch(c)
        char c;
{       mline[mptr]=c;
        if(mptr<mpmax)mptr++;
        return c;
}
preprocess()
{       int k;
        char c,sname[namesize];
        if(cmode==0)return;
```

```
        mptr=lptr=0;
        while(ch())
                {if((ch()==' ')|(ch()==9))
                        {keepch(' ');
                        while((ch()==' ')|
                                (ch()==9))
                                gch();
                        }
                else if(ch()=='"')
                        {keepch(ch());
                        gch();
                        while(ch()!='"')
                                {if(ch()==0)
                                  {error("missing quote");
                                  break;
                                  }
                                keepch(gch());
                                }
                        gch();
                        keepch('"');
                        }
                else if(ch()==39)
                        {keepch(39);
                        gch();
                        while(ch()!=39)
                                {if(ch()==0)
                                  {error("missing apostrophe");
                                  break;
                                  }
                                keepch(gch());
                                }
                        gch();
                        keepch(39);
                        }
                else if((ch()=='/')&(nch()=='*'))
                        {inchar();inchar();
                        while(((ch()=='*')&
                                (nch()=='/'))==0)
                                {if(ch()==0)inline();
                                        else inchar();
                                if(eof)break;
                                }
                        inchar();inchar();
                        }
                else if(an(ch()))
                        {k=0;
                        while(an(ch()))
                                {if(k<namemax)sname[k++]=ch();
                                gch();
                                }
                        sname[k]=0;
                        if(k=findmac(sname))
                                while(c=macq[k++])
                                        keepch(c);
                        else
                                {k=0;
                                while(c=sname[k++])
                                        keepch(c);
                                }
                        }
                else keepch(gch());
                }
        keepch(0);
        if(mptr>=mpmax)error("line too long");
        lptr=mptr=0;
        while(line[lptr++]=mline[mptr++]);
        lptr=0;
        }
addmac()
{       char sname[namesize];
        int k;
        if(symname(sname)==0)
                {illname();
                kill();
                return;
                }
        k=0;
        while(putmac(sname[k++]));
        while(ch()==' ' | ch()==9) gch();
        while(putmac(gch()));
        if(macptr>=macmax)error("macro table full");
        }
putmac(c)
        char c;
{       macq[macptr]=c;
        if(macptr<macmax)macptr++;
        return c;
}
findmac(sname)
        char *sname;
{       int k;
        k=0;
        while(k<macptr)
                {if(astreq(sname,macq+k,namemax))
                        {while(macq[k++]);
                        return k;
                        }
                while(macq[k++]);
                while(macq[k++]);
                }
        return 0;
}
outbyte(c)
        char c;
{
        if(c==0)return 0;
        if(output)
                {if((putc(c,output))<=0)
                        {closeout();
                        error("Output file error");
                        }
                }
        else putchar(c);
        return c;
}
outstr(ptr)
        char ptr[];
 {
        int k;
        k=0;
        while(outbyte(ptr[k++]));
 }
 nl()
        {outbyte(eol);}
 tab()
        {outbyte(9);}
 col()
        {outbyte(58);}
 error(ptr)
        char ptr[];
  {
        int k;
        comment();outstr(line);nl();comment();
        k=0;
        while(k<lptr)
                {if(line[k]==9) tab();
                        else outbyte(' ');
                ++k;
                }
        outbyte('^');
        nl();comment();outstr("******  ");
        outstr(ptr);
        outstr("  ******");
        nl();
        ++errcnt;
 }
 ol(ptr)
        char ptr[];
 {
        ot(ptr);
        nl();
 }
 ot(ptr)
        char ptr[];
 {
        tab();
        outstr(ptr);
 }
 streq(str1,str2)
        char str1[],str2[];
  {
        int k;
        k=0;
        while (str2[k])
                {if ((str1[k])!=(str2[k])) return 0;
                k++;
                }
        return k;
 }
 astreq(str1,str2,len)
        char str1[],str2[];int len;
 {
        int k;
        k=0;
        while (k<len)
                {if ((str1[k])!=(str2[k]))break;
                if(str1[k]==0)break;
                if(str2[k]==0)break;
                k++;
                }
        if (an(str1[k]))return 0;
        if (an(str2[k]))return 0;
        return k;
 }
match(lit)
        char *lit;
 {
        int k;
        blanks();
        if (k=streq(line+lptr,lit))
                {lptr=lptr+k;
                return 1;
                }
        return 0;
}
amatch(lit,len)
        char *lit;int len;
 {
        int k;
        blanks();
        if (k=astreq(line+lptr,lit,len))
                {lptr=lptr+k;
                while(an(ch())) inbyte();
                return 1;
                }
        return 0;
 }
blanks()
```

```
        {while(1)
                {while(ch()==0)
                        {inline();
                        preprocess();
                        if(eof)break;
                        }
                if(ch()==' ')gch();
                else if(ch()==9)gch();
                else return;
                }
        }
outdec(number)
        int number;
{
        int k,zs;
        char c;
        zs = 0;
        k=10000;
        if (number<0)
                {number=(-number);
                outbyte('-');
                }
        while (k>=1)
                {
                c=number/k + '0';
                if ((c!='0')|(k==1)|(zs))
                        {zs=1;outbyte(c);}
                number=number%k;
                k=k/10;
                }
}
/*      >>>>>>> start of cc5 <<<<<<<    */

expression()
{
        int lval[2];
        if(heir1(lval))rvalue(lval);
}
heir1(lval)
        int lval[];
{
        int k,lval2[2];
        k=heir2(lval);
        if (match("="))
                {if(k==0){needlval();return 0;}
                if (lval[1])push();
                if(heir1(lval2))rvalue(lval2);
                store(lval);
                return 0;
                }
        else return k;
}
heir2(lval)
        int lval[];
{       int k,lval2[2];
        k=heir3(lval);
        blanks();
        if(ch()!='|')return k;
        if(k)rvalue(lval);
        while(1)
                {if (match("|"))
                        {push();
                        if(heir3(lval2)) rvalue(lval2);
                        pop();
                        or();
                        }
                else return 0;
                }
}
heir3(lval)
        int lval[];
{       int k,lval2[2];
        k=heir4(lval);
        blanks();
        if(ch()!='^')return k;
        if(k)rvalue(lval);
        while(1)
                {if (match("^"))
                        {push();
                        if(heir4(lval2))rvalue(lval2);
                        pop();
                        xor();
                        }
                else return 0;
                }
}
heir4(lval)
        int lval[];
{       int k,lval2[2];
        k=heir5(lval);
        blanks();
        if(ch()!='&')return k;
        if(k)rvalue(lval);
        while(1)
                {if (match("&"))
                        {push();
                        if(heir5(lval2))rvalue(lval2);
                        pop();
                        and();
                        }
                else return 0;
                }
}
heir5(lval)
        int lval[];
{
        int k,lval2[2];
        k=heir6(lval);
        blanks();
        if((streq(line+lptr,"==")==0)&
                (streq(line+lptr,"!=")==0))return k;
        if(k)rvalue(lval);
        while(1)
                {if (match("=="))
                        {push();
                        if(heir6(lval2))rvalue(lval2);
                        pop();
                        eq();
                        }
                else if (match("!="))
                        {push();
                        if(heir6(lval2))rvalue(lval2);
                        pop();
                        ne();
                        }
                else return 0;
                }
}
heir6(lval)
        int lval[];
{
        int k,lval2[2];
        k=heir7(lval);
        blanks();
        if((streq(line+lptr,"<")==0)&
                (streq(line+lptr,">")==0)&
                (streq(line+lptr,"<=")==0)&
                (streq(line+lptr,">=")==0))return k;
                if(streq(line+lptr,">>"))return k;
                if(streq(line+lptr,"<<"))return k;
        if(k)rvalue(lval);
        while(1)
                {if (match("<="))
                        {push();
                        if(heir7(lval2))rvalue(lval2);
                        pop();
                        if(cptr=lval[0])
                                if(cptr[ident]==pointer)
                                {ule();
                                continue;
                                }
                        if(cptr=lval2[0])
                                if(cptr[ident]==pointer)
                                {ule();
                                continue;
                                }
                        le();
                        }
                else if (match(">="))
                        {push();
                        if(heir7(lval2))rvalue(lval2);
                        pop();
                        if(cptr=lval[0])
                                if(cptr[ident]==pointer)
                                {uge();
                                continue;
                                }
                        if(cptr=lval2[0])
                                if(cptr[ident]==pointer)
                                {uge();
                                continue;
                                }
                        ge();
                        }
                else if((streq(line+lptr,"<"))&
                        (streq(line+lptr,"<<")==0))
                        {inbyte();
                        push();
                        if(heir7(lval2))rvalue(lval2);
                        pop();
                        if(cptr=lval[0])
                                if(cptr[ident]==pointer)
                                {ult();
                                continue;
                                }
                        if(cptr=lval2[0])
                                if(cptr[ident]==pointer)
                                {ult();
                                continue;
                                }
                        lt();
                        }
                else if((streq(line+lptr,">"))&
                        (streq(line+lptr,">>")==0))
                        {inbyte();
                        push();
                        if(heir7(lval2))rvalue(lval2);
                        pop();
                        if(cptr=lval[0])
                                if(cptr[ident]==pointer)
                                {ugt();
                                continue;
                                }
                        if(cptr=lval2[0])
                                if(cptr[ident]==pointer)
                                {ugt();
                                continue;
                                }
                        gt();
```

```
                }
        else return 0;
        }
}
/*      >>>>>> start of cc6 <<<<<<      */
heir7(lval)
        int lval[];
{
        int k,lval2[2];
        k=heir8(lval);
        blanks();
        if((streq(line+lptr,">>")==0)&
                (streq(line+lptr,"<<")==0))return k;
        if(k)rvalue(lval);
        while(1)
                {if (match(">>"))
                        {push();
                        if(heir8(lval2))rvalue(lval2);
                        pop();
                        asr();
                        }
                else if (match("<<"))
                        {push();
                        if(heir8(lval2))rvalue(lval2);
                        pop();
                        asl();
                        }
                else return 0;
                }
}
heir8(lval)
        int lval[];
{
        int k,lval2[2];
        k=heir9(lval);
        blanks();
        if((ch()!='+')&(ch()!='-'))return k;
        if(k)rvalue(lval);
        while(1)
                {if (match("+"))
                        {push();
                        if(heir9(lval2))rvalue(lval2);
                        if(cptr=lval[0])
                                if((cptr[ident]==pointer)&
                                (cptr[type]==cint))
                                doublereg();
                        pop();
                        add();
                        }
                else if (match("-"))
                        {push();
                        if(heir9(lval2))rvalue(lval2);
                        if(cptr=lval[0])
                                if((cptr[ident]==pointer)&
                                (cptr[type]==cint))
                                doublereg();
                        pop();
                        sub();
                        }
                else return 0;
                }
}
heir9(lval)
        int lval[];
{
        int k,lval2[2];
        k=heir10(lval);
        blanks();
        if((ch()!='*')&(ch()!='/')&
                (ch()!='%'))return k;
        if(k)rvalue(lval);
        while(1)
                {if (match("*"))
                        {push();
                        if(heir9(lval2))rvalue(lval2);
                        pop();
                        mult();
                        }
                else if (match("/"))
                        {push();
                        if(heir10(lval2))rvalue(lval2);
                        pop();
                        div();
                        }
                else if (match("%"))
                        {push();
                        if(heir10(lval2))rvalue(lval2);
                        pop();
                        mod();
                        }
                else return 0;
                }
}
heir10(lval)
        int lval[];
{
        int k;
        char *ptr;
        if(match("++"))
                {if((k=heir10(lval))==0)
                        {needlval();
                        return 0;
                        }
                if(lval[1])push();
                rvalue(lval);
                inc();
                ptr=lval[0];
                if((ptr[ident]==pointer)&
                        (ptr[type]==cint))
                                inc();
                store(lval);
                return 0;
                }
        else if(match("--"))
                {if((k=heir10(lval))==0)
                        {needlval();
                        return 0;
                        }
                if(lval[1])push();
                rvalue(lval);
                dec();
                ptr=lval[0];
                if((ptr[ident]==pointer)&
                        (ptr[type]==cint))
                                dec();
                store(lval);
                return 0;
                }
        else if (match("-"))
                {k=heir10(lval);
                if (k) rvalue(lval);
                neg();
                return 0;
                }
        else if(match("*"))
                {k=heir10(lval);
                if(k)rvalue(lval);
                lval[1]=cint;
                if(ptr=lval[0])lval[1]=ptr[type];
                lval[0]=0;
                return 1;
                }
        else if(match("&"))
                {k=heir10(lval);
                if(k==0)
                        {error("illegal address");
                        return 0;
                        }
                else if(lval[1])return 0;
                else
                        {immed();
                        outstr(ptr=lval[0]);
                        nl();
                        lval[1]=ptr[type];
                        return 0;
                        }
                }
        else
                {k=heir11(lval);
                if(match("++"))
                        {if(k==0)
                                {needlval();
                                return 0;
                                }
                        if(lval[1])push();
                        rvalue(lval);
                        inc();
                        ptr=lval[0];
                        if((ptr[ident]==pointer)&
                                (ptr[type]==cint))
                                        inc();
                        store(lval);
                        dec();
                        if((ptr[ident]==pointer)&
                                (ptr[type]==cint))
                                dec();
                        return 0;
                        }
                else if(match("--"))
                        {if(k==0)
                                {needlval();
                                return 0;
                                }
                        if(lval[1])push();
                        rvalue(lval);
                        dec();
                        ptr=lval[0];
                        if((ptr[ident]==pointer)&
                                (ptr[type]==cint))
                                        dec();
                        store(lval);
                        inc();
                        if((ptr[ident]==pointer)&
                                (ptr[type]==cint))
                                inc();
                        return 0;
                        }
                else return k;
                }
        }
/*      >>>>>> start of cc7 <<<<<<      */
heir11(lval)
        int *lval;
{       int k;char *ptr;
        k=primary(lval);
        ptr=lval[0];
        blanks();
        if((ch()=='[')|(ch()=='('))
        while(1)
```

```
        {if(match("["))
                {if(ptr==0)
                        {error("can't subscript");
                        junk();
                        needbrack("]");
                        return 0;
                        }
                else if(ptr[ident]==pointer)rvalue(lval);
                else if(ptr[ident]!=array)
                        {error("can't subscript");
                        k=0;
                        }
                push();
                expression();
                needbrack("]");
                if(ptr[type]==cint)doublereg();
                pop();
                add();
                lval[0]=0;
                lval[1]=ptr[type];
                k=1;
                }
        else if(match("("))
                {if(ptr==0)
                        {callfunction(0);
                        }
                else if(ptr[ident]!=function)
                        {rvalue(lval);
                        callfunction(0);
                        }
                else callfunction(ptr);
                k=lval[0]=0;
                }
        else return k;
        }
if(ptr==0)return k;
if(ptr[ident]==function)
        {immed();
        outstr(ptr);
        nl();
        return 0;
        }
return k;
}
primary(lval)
        int *lval;
{       char *ptr,sname[namesize];int num[1];
        int k;
        if(match("("))
                {k=heir1(lval);
                needbrack(")");
                return k;
                }
        if(symname(sname))
                {if(ptr=findloc(sname))
                        {getloc(ptr);
                        lval[0]=ptr;
                        lval[1]=ptr[type];
                        if(ptr[ident]==pointer)lval[1]=cint;
                        if(ptr[ident]==array)return 0;
                                else return 1;
                        }
                if(ptr=findglb(sname))
                        if(ptr[ident]!=function)
                        {lval[0]=ptr;
                        lval[1]=0;
                        if(ptr[ident]!=array)return 1;
                        immed();
                        outstr(ptr);nl();
                        lval[1]=ptr[type];
                        return 0;
                        }
                ptr=addglb(sname,function,cint,0);
                lval[0]=ptr;
                lval[1]=0;
                return 0;
                }
        if(constant(num))
                return(lval[0]=lval[1]=0);
        else
                {error("invalid expression");
                immed();outdec(0);nl();
                junk();
                return 0;
                }
        }
store(lval)
        int *lval;
{       if (lval[1]==0)putmem(lval[0]);
        else putstk(lval[1]);
}
rvalue(lval)
        int *lval;
{       if((lval[0] != 0) & (lval[1] == 0))
                getmem(lval[0]);
                else indirect(lval[1]);
}
test(label)
        int label;
{
        needbrack("(");
        expression();
        needbrack(")");
        testjump(label);
}
constant(val)
        int val[];
{       if (number(val))
                immed();
        else if (pstr(val))
                immed();
        else if (qstr(val))
                {immed();printlabel(litlab);outbyte('+ );}
        else return 0;
        outdec(val[0]);
        nl();
        return 1;
}
number(val)
        int val[];
{       int k,minus;char c;
        k=minus=1;
        while(k)
                {k=0;
                if (match("+")) k=1;
                if (match("-")) {minus=(-minus);k=1;}
                }
        if(numeric(ch())==0)return 0;
        while (numeric(ch()))
                {c=inbyte();
                k=k*10+(c-'0');
                }
        if (minus<0) k=(-k);
        val[0]=k;
        return 1;
}
pstr(val)
        int val[];
{       int k;char c;
        k=0;
        if (match("'")==0) return 0;
        while((c=gch())!=39)
                k=(k&255)*256 + (c&127);
        val[0]=k;
        return 1;
}
qstr(val)
        int val[];
{       char c;
        if (match(quote)==0) return 0;
        val[0]=litptr;
        while (ch()!='"')
                {if(ch()==0)break;
                if(litptr>=litmax)
                        {error("string space exhausted");
                        while(match(quote)==0)
                                if(gch()==0)break;
                        return 1;
                        }
                litq[litptr++]=gch();
                }
        gch();
        litq[litptr++]=0;
        return 1;
}
/*      >>>>>> start of cc8 <<<<<<<     */

/* Begin a comment line for the assembler */
comment()
{       outbyte(';');
}
/* Print all assembler info before any code is generated */
header()
{       comment();
        outstr("small-c compiler rev 1.1");
        nl();
}
/* Print any assembler stuff needed after all code */
trailer()
{       /* ol("END"); */
}
/* Fetch a static memory cell into the primary register */
getmem(sym)
        char *sym;
{       if((sym[ident]!=pointer)&(sym[type]==cchar))
                {ot("LDA ");
                outstr(sym+name);
                nl();
                call("ccsxt");
                }
        else
                {ot("LHLD ");
                outstr(sym+name);
                nl();
                }
        }
/* Fetch the address of the specified symbol */
/*      into the primary register */
getloc(sym)
        char *sym;
{       immed();
        outdec(((sym[offset]&255)+
                ((sym[offset+1]&255)<<8))-
                sp);
        nl();
        ol("DAD SP");
        }
/* Store the primary register into the specified */
```

```
/*      static memory cell */
putmem(sym)
        char *sym;
{       if((sym[ident]!=pointer)&(sym[type]==cchar))
                {ol("MOV A,L");
                ot("STA ");
                }
        else ot("SHLD ");
        outstr(sym+name);
        nl();
        }
/* Store the specified object type in the primary register */
/*      at the address on the top of the stack */
putstk(typeobj)
        char typeobj;
{       pop();
        if(typeobj==cchar)call("ccpchar");
                else call("ccpint");
        }
/* Fetch the specified object type indirect through the */
/*      primary register into the primary register */
indirect(typeobj)
        char typeobj;
{       if(typeobj==cchar)call("ccgchar");
                else call("ccgint");
}
/* Swap the primary and secondary registers */
swap()
{       ol("XCHG");
}
/* Print partial instruction to get an immediate value */
/*      into the primary register */
immed()
{       ot("LXI H,");
}
/* Push the primary register onto the stack */
push()
{       ol("PUSH H");
        sp=sp-2;
}
/* Pop the top of the stack into the secondary register */
pop()
{       ol("POP D");
        sp=sp+2;
}
/* Swap the primary register and the top of the stack */
swapstk()
{       ol("XTHL");
}
/* Call the specified subroutine name */
call(sname)
        char *sname;
{       ot("CALL ");
        outstr(sname);
        nl();
}
/* Return from subroutine */
ret()
{       ol("RET");
}
/* Perform subroutine call to value on top of stack */
callstk()
{       immed();
        outstr("$+5");
        nl();
        swapstk();
        ol("PCHL");
        sp=sp+2;
        }
/* Jump to specified internal label number */
jump(label)
        int label;
{       ot("JMP ");
        printlabel(label);
        nl();
        }
/* Test the primary register and jump if false to label */
testjump(label)
        int label;
{       ol("MOV A,H");
        ol("ORA L");
        ot("JZ ");
        printlabel(label);
        nl();
        }
/* Print pseudo-op to define a byte */
defbyte()
{       ot("DB ");
}
/*Print pseudo-op to define storage */
defstorage()
{       ot("DS ");
}
/* Print pseudo-op to define a word */
defword()
{       ot("DW ");
}
/* Modify the stack pointer to the new value indicated */
modstk(newsp)
        int newsp;
 {      int k;
        k=newsp-sp;
        if(k==0)return newsp;
        if(k>=0)
                {if(k<7)
                        {if(k&1)
                                {ol("INX SP");
                                k--;
                                }
                        while(k)
                                {ol("POP B");
                                k=k-2;
                                }
                        return newsp;
                        }
                }
        if(k<0)
                {if(k>-7)
                        {if(k&1)
                                {ol("DCX SP");
                                k++;
                                }
                        while(k)
                                {ol("PUSH B");
                                k=k+2;
                                }
                        return newsp;
                        }
                }
        swap();
        immed();outdec(k);nl();
        ol("DAD SP");
        ol("SPHL");
        swap();
        return newsp;
}
/* Double the primary register */
doublereg()
{       ol("DAD H");
}
/* Add the primary and secondary registers */
/*      (results in primary) */
add()
{       ol("DAD D");
}
/* Subtract the primary register from the secondary */
/*      (results in primary) */
sub()
{       call("ccsub");
}
/* Multiply the primary and secondary registers */
/*      (results in primary */
mult()
{       call("ccmult");
}
/* Divide the secondary register by the primary */
/*      (quotient in primary, remainder in secondary) */
div()
{       call("ccdiv");
}
/* Compute remainder (mod) of secondary register divided */
/*      by the primary */
/*      (remainder in primary, quotient in secondary) */
mod()
{       div();
        swap();
        }
/* Inclusive 'or' the primary and the secondary registers */
/*      (results in primary) */
or()
        {call("ccor");}
/* Exclusive 'or' the primary and seconday registers */
/*      (results in primary) */
xor()
        {call("ccxor");}
/* 'And' the primary and secondary registers */
/*      (results in primary) */
and()
        {call("ccand");}
/* Arithmetic shift right the secondary register number of */
/*      times in primary (results in primary) */
asr()
        {call("ccasr");}
/* Arithmetic left shift the secondary register number of */
/*      times in primary (results in primary) */
asl()
        {call("ccasl");}
/* Form two's complement of primary register */
neg()
        {call("ccneg");}
/* Form one's complement of primary register */
com()
        {call("cccom");}
/* Increment the primary register by one */
inc()
        {ol("INX H");}
/* Decrement the primary register by one */
dec()
        {ol("DCX H");}

/* Following are the conditional operators */
/* They compare the secondary register against the primary */
/* and put a literal 1 in the primary if the condition is */
/* true, otherwise they clear the primary register */

/* Test for equal */
eq()
        {call("cceq");}
```

```
/* Test for not equal */
ne()
        {call("ccne");}
/* Test for less than (signed) */
lt()
        {call("cclt");}
/* Test for less than or equal to (signed) */
le()
        {call("ccle");}
/* Test for greater than (signed) */
gt()
        {call("ccgt");}
/* Test for greater than or equal to (signed) */
ge()
        {call("ccge");}
/* Test for less than (unsigned) */
ult()
        {call("ccult");}
/* Test for less than or equal to (unsigned) */
ule()
        {call("ccule");}
/* Test for greater than (unsigned) */
ugt()
        {call("ccugt");}
/* Test for greater than or equal to (unsigned) */
uge()
        {call("ccuge");}

/*      <<<<<  End of compiler  >>>>>   */
```

UNIX Time-Sharing System:

The C Programming Language

BY D. M. RITCHIE, S. C. JOHNSON, M. E. LESK, AND B. W. KERNIGHAN
Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

C is a general-purpose programming language that has proven useful for a wide variety of applications. It is the primary language of the UNIX system, and is also available in several other environments. This paper provides an overview of the syntax and semantics of C and a discussion of its strengths and weaknesses.*

C is a general-purpose programming language featuring economy of expression, modern control flow and data structure capabilities, and a rich set of operators and data types.

C is not a "very high-level" language nor a big one and is not specialized to any particular area of application. Its generality and an absence of restrictions make it more convenient and effective for many tasks than supposedly more powerful languages. C has been used for a wide variety of programs, including the UNIX operating system, the C compiler itself, and essentially all UNIX applications software. The language is sufficiently expressive and efficient to have completely displaced assembly language programming on UNIX.

*UNIX is a trademark of Bell Laboratories.

© 1978, American Telephone and Telegraph Company; Reprinted by permission.

C was originally written for the PDP-11 under UNIX, but the language is not tied to any particular hardware or operating system. C compilers run on a wide variety of machines, including the Honeywell 6000, the IBM System/370, and the Interdata 8/32.

I. THE LINGUISTIC HISTORY OF C

The C language in use today[1] is the product of several years of evolution. Many of its most important ideas stem from the considerably older, but still quite vital, language BCPL[2] developed by Martin Richards. The influence of BCPL on C proceeded indirectly through the language B,[3] which was written by Ken Thompson in 1970 for the first UNIX system on the PDP-11.

Although neither B nor C could really be considered dialects of BCPL, both share several characteristic features with it:

(*i*) All are able to express the fundamental flow-control constructions required for well-structured programs: statement grouping, decision-making (if), looping (while) with the termination test either at the top or the bottom of the loop, and branching out to a sequence of possible cases (switch). It is interesting that BCPL provided these constructions in 1967, well before the current vogue for "structured programming."

(*ii*) All three languages include the concept of "pointer" and provide the ability to do address arithmetic.

(*iii*) In all three languages, the arguments to functions are passed by copying the value of the argument, and it is impossible for the function to change the actual argument. When it is desired to achieve "call by reference," a pointer may be passed explicitly, and the function may change the object to which the pointer points. Any function is allowed to be recursive, and its local variables are typically "automatic" or specific to each invocation.

(*iv*) All three languages are rather low-level, in that they deal with the same sorts of objects that most computers do. BCPL and B restrict their attention almost completely to machine words, while C widens its horizons somewhat to characters and (possibly multi-word) integers and floating-point numbers. None deals directly with composite objects such as character strings, sets, lists, or arrays considered as a whole. The languages themselves do not define any storage allocation facility beside static definition and the stack discipline provided by the local variables of functions; likewise, I/O is not part of any of these languages. All these higher mechanisms must be provided by explicitly called routines from libraries.

B and BCPL differ mainly in their syntax, and many differences stemmed from the very small size of the first B compiler (fewer than 4K 18-bit words on the PDP-7). Several constructions in BCPL encourage a compiler to maintain a representation of the entire program in memory. In BCPL, for example,

```
valof $(
        ...
        resultis expression
        ...
$)
```

is syntactically an expression. It provides a way of packaging a block of many statements into a sort of unnamed internal procedure yielding a single result (delivered by the resultis statement). The valof construction can occur in the middle of any expression, and can be arbitrarily large. The B language avoided the difficulties caused by this and some other

constructions by rigorously simplifying (and in some cases adjusting to personal taste) the syntax of BCPL.

In spite of many syntactic changes, B remained very close to BCPL semantically. The most characteristic feature of both languages is their nearly identical treatment of addresses (pointers). They support a model of the storage of the machine consisting of a sequence of equal-sized cells, into which values can be placed; in typical implementations, these cells will be machine words. Each identifier in a program corresponds to a cell, and a cell may contain a variety of values. Most often the value is an integer, or perhaps a respresentation of a character. All the cells, however, are numbered; the address of a cell is just the integer giving its ordinal position. BCPL has a unary operator lv (in some versions, and also in B and C, shortened to &) that, when applied to a name, yields the address of the cell corresponding to the name. The inverse operator rv (later *) yields the value in the cell pointed to its argument. Thus the statement

```
px = &x;
```

of B assigns to px the number that can be interpreted as the address of x; the statements

```
y = *px + 2;
*px = 5;
```

first use the value in the cell pointed to by px (which is the same cell as x) and then assign 5 to this cell.

Arrays in BCPL and B are intimately tied up with pointers. An array declaration, which might in BCPL be written

```
let Array = vec 10
```

and in B

```
auto Array [10];
```

creates a single cell named Array and initializes it with the address of the first of a sequence of 10 unnamed cells containing the array itself. Since the quantity stored in Array is just the address of the cell of the first element of the array, the expression

```
Array + i
```

is the address of the *ith* element, counting from zero. likewise, applying the indirection operator,

```
* (Array + i)
```

refers to the value of the *ith* member of the array. This operation is so frequent that special syntax was invented to express it:

```
Array [i]
```

Thus, despite its asymmetric appearance, subscripting is a commutative operation; the above example could equally well be written

```
i[Array]
```

In BCPL and B there is only one type of object, the machine word, so when the same language operator is applied to two operands, the calculation actually carried out must always be the same. Thus, for example, if one wishes to provide the ability to do floating-point arithmetic, the "+" operator notation cannot be used, since it implies an integer addition. Instead (in a version of BCPL for the GE 635), a "." was placed in front of each operator that had floating-point operands. As may be appreciated, this was a frequent source of errors.

The machine model implied by the definitions of BCPL and B is simple and self-consistent. It is, however, inadequate for many purposes, and on many machines it causes inefficiencies when implemented. The problems became evident to us after B began to be used heavily on the first PDP-11 version of UNIX. The first followed from the fact that the PDP-11, like a number of machines (including, for example, the IBM System/370), is byte addressed; a machine address refers to any of several bytes (characters) in a word, not the word alone. Most obviously, the word orientation of B cut us off from any convenient ability to access individual bytes. Equally important was the fact that, before any address could be used, it had to be shifted left by one place. The reason for this is simple: there are two byte per PDP-11 word. On the one hand, the language guranteed that if 1 was added to an address quantity, it would point to the next word; on the other, the machine architecture required that word addresses be even and equal to the byte number of the first byte in the word. Since, finally, there was no way to distinguish cells containing ordinary integers from those containing pointers, the only solution visible was to represent pointers as word numbers and then, at the point of use, convert to the byte representation by multiplication by 2.

Yet another problem was introduced by the desire to provide for floating-point arithmetic. The PDP-11 supports two floating-point formats, one of which requires two words, the other four. In neither case was it satisfactory to use the trick used on the GE 635 (operator like ".+") because there was no way to represent the requirement for a single data item occupying four or eight bytes. This problem did not arise on the 635 because integers and single-precision floating-point both require only one word.

Thus the problems evidenced by B led us to design a new language that (after a brief period under the name NB) was dubbed C. The major advance provided by C is its typing structure, which completely solved the difficulties mentioned above. Each declaration in a C program specifies (sometimes implicitly) a *type*, which determines how much storage the object requires and how it is to be interpreted. The original fundamental types provided were single character (byte), integer, single-precision floating-point, and double-precision floating-point. (Others discussed below were added later.) Thus in the program

```
double a, b;
   ...
a = b + 3;
```

the compiler is able to determine from the declarations of *a* and *b* the fact that they require four words of storage each, that the "+" means a double-precision floating add, and that "3" must be converted to floating.

Of course, the idea of typing variables is in no way original with C; in fact, it was the general rule among the most widely used and influential languages, including ALGOL, FORTRAN, and PL/I. Nevertheless, the introduction of types marked an important change in our own thinking. The typeless nature of BCPL and B had seemed to promise a great simplification in the implementation, understanding, and use of these languages. By the time that C was created (circa 1972), advocates of languages like ALGOL 68 and PASCAL recommended a strongly enforced type structure on psychological grounds; but even disregarding their arguments, the typeless nature of BCPL and B seemed inappropriate, for purely technological reasons, to be available hardware.

II. THE TYPE STRUCTURE OF C

The introduction of types in C, although a major departure from the tradition of BCPL and B, was done in such a way that many of the characteristic usages of the earlier languages survived. To some extent, this continuity was an attempt to preserve as much as possible of the considerable corpus of exisitng software written in B, but even more important, especially in retrospect, was the desire to minimize the intellectual distance beween the past and the future ways of expression.

2.1 Pointers, arrays and address arithmetic

One clear example of the similarity of C to the earlier languages is its treatment of pointers and arrays. In C an array of 10 integers might be declared

```
int Array [10];
```

which is identical to the corresponding declaration in B. (Arrays begin at zero; the elements of *Array* are *Array[0]*, . . . , *Array[9]*. As discussed above, the B implementation caused a cell named *Array* to be allocated and initialized with a pointer to 10 otherwise unnamed cells to hold the array. In C, the effect is bit different; 10 integers are allocated, and the first is associated with the name *Array*. But C also includes a general rule that, whenever the name of an array appears, in an expression, it is converted to a pointer to the first member of the array. Strictly speaking, we should say, for this example, it is converted to an *integer pointer* since all C pointers are associated with a particular type to which they point. In most usages, the actual effects of the slightly different meanings of *Array* are indistinguishable. Thus in the C expression

```
Array + i
```

The identifier *Array* is converted to a pointer to the first element of the array; *i* is scaled (if required) before it is added to the pointer. For a byte-addressed machine, the scale factor is the number of bytes in an integer; for a word-addressed machine the scale factor is unity. In any event, the result is a pointer to the *ith* member of the array. Likewise identical in effect to the interpretation of B,

```
* (Array + i)
```

is the *ith* member itself, and

```
Array [i]
```

is another notation for the same thing. In all these cases, of course, should Array be an array of, or pointer to, some objects other than integers, the scale factor is adjusted appropriately. The pointer arithmetic, as written, is independent of the type of object to which the pointer points and indeed of the internal representation of the pointer.

2.2 Derived types

As mentioned above, the basic types in C were originally int, which represents an integer in the basic size provided by the machine architecture; *char*, which represents a single byte; *float*, a single-precision floating-point number; and *double*, double-precision floating-point. Over the years, *long, short,* and *unsigned* integers have been added. In current C implementations, *long* is at least 32 bits; *short* is usually 16 bits; and *int* remains the "natural" size for the machine at hand. *Unsigned* integers exist mainly to squeeze an extra bit out of the machine, since the sign bit need not be represented.

In addition to these basic types, C provides a conceptually infinite hierarchy of derived types, which are formed by composition of the basic types with pointers, arrays, structures, unions, and functions. Examples of pointer and array declarations have already been exhibited; another is

```
double *vecp, vector[100];
```

which declares a pointer *vecp* to double-precision floating numbers, and an array *vector* of the same kind of objects. The size of an array, when specified, must always be a constant.

A *structure* is an aggregate of one or more objects, usually of various types, which can be treated as a unit. C structures are essentially the same as records in languages like PASCAL, and semantically, though not syntactically, like PL/I and Cobol structures. Thus,

```
struct tag {
        int     i;
        float   f;
        char    c[3];
};
```

defines a template, called *tag*, for a structure containing three *members*: an integer *i*, a floating point number *f*, and a three-character array *c*. The declaration

```
struct tag x, y[10], *p;
```

declares a structure *x* of this type, an array *y* of 10 such structures, and a pointer *p* to this kind of structure. The hierarchical nature of derived types is clearly evident here: *y* is an array of structures whose members include an array of characters. References to individual members of structures use the . operator.

```
x.i
x.f
y[i].c[0]
(*p).c[1]
```

Parentheses in the last line are necessary because the . binds more tightly than *. It turns out that pointers to structures are so common that special syntax is called for to express structure access through a point.

```
p->c[1]
p->i
```

This soon becomes more natural than the equivalent

```
(*p).c[1]
(*p).i
```

A union is capable of holding, at different times, objects of different types, with the compiler keeping track of size and alignment requirements. Unions provide a way to manipulate different kinds of data in a single part of storage, without embedding machine-dependent information (like the relative sizes of *int* and *float*) in a program. For example, the union *u*, declared

```
union {
        int   i;
        float f;

} u;
```

can hold either an *int* (written u.i) or a *float* (written u.f.). Regardless of the machine it is compiled on, it will be large enough to hold either one of these quantitites. A union is syntactically identical to a structure; it may be considered as a structure in which all the members begin at the same offset.

Unions in C are more analogous to PL/I's CALL than to the unions of ALGOL 68 or the variant records of PASCAL, because it is the responsibility of the programmer to avoid referring to a union that does not currently contain an object of the implied type.

A *function* is a subprogram that returns an object of a given type:

```
unsigned unsf( );
```

declares a function that returns *unsigned*. The type of a function ignores the number and types of its arguments, although in general the call and the definition must agree.

2.3 Type composition

The syntax of declarations borrows from that of expressions. The key idea is that a declaration, say

```
int    ... ;
```

contains a part " . . . " that, if it appeared in an expression would be of type *int*. The constructions seen so far, for example,

```
int    *lptr;
       int    lfunc();
       int    iarr[10];
```

exhibit this approach, but more complicated declarations are common. For example,

```
int    *funcptr();
int    (*ptrfunc)();
```

declare respectively a function that returns a pointer to an integer, and a pointer to a function that returns an integer. The extra parentheses in the second are needed to make the * apply directly to ptrfunc, since the implicit function-call operator () binds more tightly than *. Functions are not variables, so array or structures of functions are not permitted. However, a pointer to a function, like ptrfunc, may be stored, copied, passed as an argument, returned by a function, and so on, just as any other pointer.

Arrays of pointers are frequently used instead of multi-dimensional arrays. The usage of a and b when declared

```
int    a[10][10];
int    *b[10];
```

may be similar, in that a[5][5] and b[5][5] are both legal references to a single int, but a is a true array: all 100 storage cells have been allocated, and the conventional rectangular subscript calculation is done. For b, however, the declaration has only allocated 10 pointers; each must be set to point to an array of integers. Assuming each does point to a 10-element array, then there will be 100 storage cells set aside, plus the 10 cells for the pointers. Thus the array of pointers uses slightly more space and may require an extra initialization step, but has two advantages: it trades an indirection for a subscript multiplication, and it permits the rows of the array to be of different lengths. (That is, each element of b need not point to a 10-element vector; some may point to 2 elements, some to 20). Particularly with strings whose length is not known in advance, an array of pointers is often used instead of a multidimensional array. Every C main program gets access to its invoking command line in this form, for example.

The idea of specifying types by appropriating some of the syntax of expressions seems to be original with C, and for the simpler cases, it works well. Occasionally some rather ornate types are needed, and the declaration may be a bit hard to interpret. For example, a pointer to an array of pointers to functions, each returning an int, would be written

```
int (*(*funnyarray)[])();
```

which is certainly opaque, although understandable enough if read from the inside out. In an expression, funnayarray might appear as

```
i = (*(*funnyarray)[j])(k);
```

The corresponding Algol 68 declaration is

```
ref [] ref proc int funnyarray
```

which read from left to right in correspondence with the informal description of the type if ref is taken to be the equivalent of C's "pointer to." The Algol may be clearer, but both are hard to grasp.

III. STATEMENTS AND CONTROL FLOW

Control flow in C differs from other languages primarily in details of syntax. As in PL/1, semicolons are used to terminate statements, not to separate them. Most statements are just expressions followed by a semicolon; since assignments are expressions, there is no need for a special assignment statement.

Statements are grouped with braces { and } , rather than with words like begin-end or do-od, because the more concise form seems much easier to read and is certainly easier to type. A sequence of statements enclosed in { } is syntactically a single statement.

The if-else statement has the form

```
if (expression)
        statement
else
        statement
```

The *expression* is evaluated; if it is "true" (that is, if *expression* has a non-zero value), the first statement is done. If it is "false" (*expression* is zero) and if there is an else part, the second *statement* is executed instead. The else part is optional; if it is omitted in a sequence of nested if's, the resulting ambiguity is resolved in the usual way by associating the else with the nearest previous else-less if.

The switch statement provides a multi-way branch depending on the value of an integer expression:

```
switch (expression)
        case const:
                code
        case const:
                code
        ...
        default:
                code
}
```

The *expression* is evaluated and compared against the various cases, which are labeled with distinct integer constant values. If any case matches, execution begins at that point. If no case matches but there is a default statement, execution begins there: otherwise, no part of the switch is executed.

The cases are just labels, and so control may flow through one case to the next. Although this permits multiple lables on cases, it also means that in general most cases must be terminated with an explicit exit from the switch (the break statement below).

The switch construction is part of C's legacy from BCPL; it is so useful and so easy to provide that the lack of a corresponding facility of acceptable generality in languages ranging from For-

tran through Algol 68, and even to Pascal (which does not provide for a default), must be considered a real failure of imagination in language designers.

C provides three kinds of loops. The while is simply

```
while (expression)
    statement
```

The *expression* is evaluated; if it is true (non-zero), the *statement* is executed, and then the process repeats. When *expression* becomes false (zero), execution terminates.

The do statement is a test-at-the-bottom loop:

```
do
    statement
while (expression);
```

statement is performed once, then *expression* is evaluated. If it is true, the loop is repeated; otherwise it is terminated.

The for loop is reminiscent of similarly named loops in other languaged, but rather more general. The for statement

```
for (expr1; expr2; expr3)
    statement
```

is equivalent to

```
expr1;
while (expr2)
    {
    statement
    expr3;
    }
```

Grammatically, the three components of a for loop are expressions. Any of the three parts can be omitted, although the semicolons must remain. If *expr1* or *expr3* is left out, it is simply dropped from the expansion. If the test, *expr2*, is not present, it is taken as permanently true, so

```
for (;;)
    ...
```

is an "infinite" loop, to be broken by other means, for example by break, below.

The for statement keeps the loop control components together and visible at the top of the loop, as in the idiomatic

```
for (i = 0; i < N; i = i+1)
```

which processes the first N elements of an array, the analogue of the Fortran or PL/1 DO loop. The for is more general, however. The test is re-evaluated on each pass through the loop, and there is no restriction on changing the variables involved in any of the expressions in the for statement. The controlling variable i retains it value regardless of how the loop terminates. And since the components of a for are arbitrary expressions, for loops are not restricted to arithmetic progressions. For example, the classic walk along a linked list is

```
for (p = top; p != NULL; p = p -> next)
    ...
```

There are two statements for controlling loops. The break statement, as mentioned, causes an immediate exit from the immediately enclosing while, for, do or switch. The continue statement causes the next iteration of the immediately enclosing loop to begin. break and continue are asymmetric, since continue does not apply to switch.

Finally, C provides the oft-maligned goto statement. Empirically, goto's are not much used, at least on our system. The operating system itself, for example, contains 98 in some 8300 lines. The PDP-11 C compiler, in 9660 lines, has 147. Essentially all of these implement some form of branch to the top or bottom of a loop, or to error recovery code.

IV. OPERATORS AND EXPRESSIONS

C has been characterized as having a relatively rich set of operators. Some of these are quite conventional. For example, the basic binary arithmetic operators are +, -, * and /. To these, C adds the modulus operator %: m%n is the remainder when m is divided by n.

Besides the basic logical or bitwise operators & (bitwise AND), and | (bitwise OR), there are also the binary operators ^ (bitwise exclusive OR), >> (right shift), and << (left shift), and the unary operator ~ (ones complement). These operators apply to all integers; C provides no special bit-string type.

The relational operators are the usual >, >=, <, <=, == (equality test), and != (inequality test). They have the value 1 if the stated relation is true, 0 if not.

The unary pointer operators * (for indirection) and & (for taking the address) were described in Section 1. When y is such as to make the expressions &*y or &*y legal, either is just equal to y. Note that & and * are used as both binary and unary operators (with different meanings).

The simplest assignment is written =, and is used conventionally: the value of the expression on the right is stored in the object whose address is on the left. In addition, most binary operators can be combined with assignment by writing

```
a op= b
```

which has the effect of

```
a = a op b
```

except that a is only evaluated once. For example,

```
x += 3
```

is the same as

```
x = x + 3
```

if x is just a variable, but

```
p[i + j + 1] += 3
```

adds 3 to the element selected from the array p, calculating the subscript only once, and, more importantly, requiring it to be written out only once. Compound assignment operators also seem to correspond well to the way we think; "add 3 to x" is said, if not written, much more commonly than "assign x + 3 to x."

Assignment expressions have a value, just like other expressions, and may be used in larger expressions. For example, the multiple assignment

```
i = j = k = 0;
```

is a byproduct of this fact, not a special case. Another very common instance is the nesting of an assignment in the condition part of an if or a loop, as in

```
while (c = getchar()) != EOF) ...
```

which fetches a character with the function getchar, assigns it to c, then tests whether the result is an end of file marker. (Parentheses are needed because the precedence of the assignment = is lower than that of the relational !=.)

C provides two novel operators for incrementing and decrementing variables. The increment operator ++ adds 1 to its

operand; the decrement operator -- subtracts 1. Thus the statement

```
++i;
```

increments i. The unusual aspect is that ++ and -- may be used either as prefix operators (before the variable, as in ++i), or postfix (after the variable i++). In both cases, the effect is to increment i. But the expression ++i increments i *before* using its value, while i++ increments i *after* its value has been used. If i is 5, then

```
x = i++;
```

sets x to 5, but

```
x = ++i;
```

sets x to 6. In both cases, i becomes 6.

For example,

```
stack[i++] = ... ;
```

pushes a value on a stack stored in an array stack indexed by i, while

```
... = stack[--i];
```

retrieves the value and pops the stack. Of course, when the quantity incremented or decremented is a pointer, appropriate scaling is done, just as if the "1" were added explicitly:

```
*stackp++ = ... ;
... = *--stackp;
```

are analogous to the previous example, this time using a stack pointer instead of an index.

Tests may be combined with the logical connectives && (AND), || (OR), and ! (truth value negation). The && and || guarantee left-to-right evaluation, with termination as soon as the truth value is known. For example, in the test

```
if (i <= N && array[i] > 0 ...
```

if i is greater than N, then array[i] (presumably at that point an out-of-bounds reference) will not be accessed. This predictable behavior is especially convenient, and much preferable to the explicitly random order of evaluation promised by most other languages. Most C programs rely heavily on the properties of && and ||.

Finally, the *conditional expression*, written with the ternary operator ?:, provides an analogue of if-else in expressions. In the expression

```
e1 ? e2 : e3
```

the expression *e1* is evaluated first. If it is non-zero (true), then the expression *e2* is evaluated, and that is the value of the conditional expression. Otherwise, *e3* is evaluated, and that is the value. Only one of *e2* and *e3* is evaluated. Thus to set z to the maximum of a and b,

```
z = (a > b) ? a : b; /* z = max(a, b) */
```

We have already discussed how integers are scaled appropriately in pointer arithmetic. C does a number of other automatic conversions between data types, more freely than Pascal, for example, but without the wild abandon of PL/1. In all contexts, char variables and constants are promoted to int. This is particularly handy in code like

```
n = c - '0';
```

which assigns to n the integer value of the character stored in c, by subtracting the value of the character '0'. Generally, in fact, the basic types fall into only two classes, integral and floating-point; char variables, and the various lengths of int's, are taken to be representations of the same kind of thing. They occupy different amounts of storage but are essentially compatible. Boolean values as such do not exist; relational or truth-value expressions have value 1 if true, and 0 if false.

Variables of type int are converted to floating-point when combined with floats or doubles and in fact all floating arithmetic is carried out in double precision, so floats are widened to double in expressions.

Conversions that involve "narrowing" an expression (for example, when a longer value is assigned to a shorter) are also well behaved. Floating point values are converted to integer by truncation; integers convert to shorter integers or characters by dropping high-order bits.

When a conversion is desired, but is not implicit in the context, it is possible to force a conversion by an explicit operator called a *cast*. The expression

```
(type) expression
```

is a new expression whose type is that specified in *type*. for example, the sin routine expects an argument of type double; in the statement

```
x = sin((double) n);
```

the value of n is converted to double before being passed to sin.

V. THE STRUCTURE OF C PROGRAMS

Complete programs consist of one or more files containing function and data declarations. Thus, syntactically, a program is made up of a sequence of declarations; executable code appears inside functions. Conventionally, the run-time system arranges to call a function named main to start execution.

The language distinguishes the notions of *declaration* and *definition*. A declaration merely announces the properties of a variable (like its type); a definition declares a variable and also allocates storage for it or, in the case of a function, supplies the code.

5.1 Functions

The notion of *function* in C includes the subroutines and functions of FORTRAN and the procedures of most other languages. A function call is written

```
name(arglist)
```

where the parentheses are required even if the argument list is empty. All functions may be used recursively.

Arguments are passed by value, so the called function cannot in any way affect the actual argument with which it was called. This permits the called program to use its formal arguments as conveniently initialized local variables. Call by value also eliminates the class of errors, familiar to FORTRAN programmers, in which a constant is passed to a subroutine that tries to alter the corresponding argument. An array name as an actual argument, however, is converted to a pointer to the first array element (as it always is), so the effect is as if arrays were called by reference; given the pointer, the called function can work its will on the individual elements of the array. When a function must return a value through its argument list, an explicit pointer may be passed, and the function references the ultimate target through this pointer. For example, the function *swap(pa, pb)* interchanges two integers pointed to by its arguments:

```
swap(px, py)        /* flip int's
                       pointed to by px
                       and py */.
int *px, *py;

int temp;
{
temp = *px;
*px = *py;
*py = temp;
}
```

This also demonstrates the form of a function definition: the name is followed by an argument list; the arguments are declared, and the body of the function is a block, or compound statement, enclosed in braces. Declarations of local variables may follow the opening brace.

A function returns a value by

return *expression*;

The *expresssion* is automatically coerced to the type that the function returns. By default, functions are assumed to return *int*; if this is not the case, the function must be declared both in the calling routine and when it is defined. For example, a function definition is

```
double sqrt(x)  /* returns square root
                   of x */
double x;
{
    ...
}
```

In the caller, the declaration is

```
double y, sqrt( );

y = sqrt(y);
```

A function argument may be any of the basic types or a pointer, but not an array, structure, union, or function. The same is true of the value returned by a function. (the most recent versions of the language, still not standard everywhere, permit structures and unions as arguments and values of functions and allow them to be assigned.)

5.2 Data

Data declared at the top level (that is, outside the body of any function definition) are static in lifetime, and exist throughout the execution of the program. Variables declared within a function body are by default *automatic*: they come into existence when the function is entered and vanish when it is exited. Automatic variables may be declared to be ***register*** variables; when possible they will be placed in machine registers, which may result in smaller, faster code. The ***register*** declaration is only considered a hint to the compiler; no hardware register names are mentioned, and the hint may be ignored if the compiler wishes.

Static variables exist throughout the execution of a program, and retain their values across function calls. Static variables may be local to a function or (if defined at the top level) common to several functions.

External variables have the same lifetime as static, but they are also accessible to programs from other source files. That is, all references to an identically named external variable are references to the same thing.

The "storage class" of a variable can be explicitly announced in its declaration:

```
static int x;
extern double y[10];
```

More often the defaults for the context are sufficient. Inside a function, the default is auto (for automatic). Outside a function, at the top level, the default is extern. Since automatic and register variables are specific to a particular call of a particular function, they cannot be declared at the top level. Neither top-level variables nor functions explicitly declared static are visible to functions outside the file in which they appear.

5.3 Scope

Declarations may appear either at the top level or at the head of a block (compound statement). Declarations in an inner block temporarily override those of identically named variables outside. The scope of a declaration persists until the end of its block, or until the end of the file, if it was at the top level.

Since function definitions may be given only at the top level (that is, they may not be nested), there are no internal procedures. They have been forbidden not for any philosophical reason, but only to simplify the implementation. It has turned out that the ability to make certain functions and data invisible to programs in other files (by explicitly declaring them static) is sufficient to provide one of their most important uses, namely hiding their names from other functions. (However, it is not possible for one function to access the internal variables of another, as internal procedures could do.) Similarly, the ability to conceal functions and data in one file from access by another satisfies some of the most crucial requirements of modular programming (as in languages like Alphard, CLU, and Euclid), even through it does not satisfy them all.

VI. C PREPROCESSOR

It is well recognized the "magic numbers" in a program are a sign of bad programming. Most languages, therefore, provide a way to define symbolic names for constants, so that the value of a magic number need be specified in only one place, and the rest of the code can refer to the value by some mnemonic name. In C such a mechanism is available, but it is not part of the syntax of the language; instead, symbolic naming is provided by a macro preprocessor automatically invoked as part of every C compilation. For example, given the definitions

```
#define   P1   3.14159
#define   E    2.71284
```

the preprocessor replaces all occurrences of a defined name by the corresponding defining string. (Upper-case names are normally chosen to emphasize that these are not variables.) Thus, when the programmer recognizes that he has written an incorrect value for *e*, only the definition line has to be changed to

```
#define   E    2.71828
```

instead of each instance of the constant in the program.

Providing this service by a macro processor instead of by syntax has some significant advantages. The replacement text is not restricted to being numbers; any string of characters is permitted. Furthermore, the token being replaced need not be a variable, although it must have the form of a name. For example, one can define

```
#define   forever   for (;;)
```

and then writen infinite loops as

```
forever {
    ...
}
```

The macro processor also permits macros to have arguments; this capability is heavily used by some I/O packages.

A second service of the C preprocessor is library file include: a source line of the form

```
#include "name"
```

causes the contents of the file name to be interpolated into the source at that point. (includes may be nested.) This feature is much used, especially in larger programs, for making sure that all the source files of the program are supplied with identical #defines, global data declarations, and the like.

VII. ENVIRONMENTAL CONSIDERATIONS

By intent, the C language confines itself to facilities that can be mapped relatively efficiently and directly into machine instructions. For example, writing matrix operations that look exactly like scalar operations is possible in some programming languages and occasionally misleads programmers into believing that matrix operations are as cheap as scalar operations. More important, restricting the domain of the C compiler to those areas where it knows how to do a relatively efficient job provides the freedom to design subroutine libraries for the remaining tasks without constraining them to fit into some language specification. When the compiler cannot implement some facility without heavy costs in nonportability, complexity, or efficiency, there are many benefits to leaving out such a facility: it simplifies the language and the compiler, frequently without inconveniencing the user (who often rejects a high-cost built-in operation and does it himself anyway).

At present, C is restricted to simple operations on simple data types. As a result, although the C area of operation is comparatively clean and pleasant, the user must know something about the polluting effects of the environment to get most jobs done. A program can always access the raw system calls on each system if very close interaction with the operating system is needed, but standard library routines have been implemented in each C environment that try to encourage portability while retaining speed and flexibility. The basic areas covered by the standard library at present are storage allocation, string handling, and I/O. Additional libraries and utilities are available for such areas as graphics, coroutine sequencing, execution time monitoring, and parsing.

The only automatic storage management service provided by C itself is the stack discipline for automatic variables. Two subroutines exist for more flexible storage handling. The function calloc(n, s) returns a pointer to a properly aligned storage block that will hold n items each of which is s bytes long. Normally s is obtained from the sizeof pseudo-function, a compile-time function that yields the size in bytes of a variable or data type. To return a block obtained from calloc to the free storage pool, cfree(p) may be called, where p is a value returned by a previous call to calloc.

Another set of routines deals with string handling. There is no "string" data type, but an array of characters, with a convention that the end of a string is indicated by a null byte, can be used for the same purpose. The most commonly used string routines perform the functions of copying one string to another, comparing two strings, and computing a string length. More sophisticated string operations can often be performed using the I/O routines, which are described next.

Most of the routines in the standard library deal with input and output. Most C programmers use stream I/O, although there is no reason why record I/O could not be used with the language. There are three default streams: the standard input, the standard output, and the error output. The most elementary routines for dealing with these streams are getchar() which reads a character from the standard input, and putchar(c), which writes the character c on the standard output. In the environments in which C programs run, it is generally possible to redirect these streams to files or other programs; the program itself does not change and is unaware of the redirection.

The most common output function is printf(format, data1, data2, ...), which performs data conversion for formatted output. The string format is copied to the standard output, except that when a conversion specification introduced by a % character is found in format it is replaced by the value of the next data argument, converted according to the specification. For example,

```
printf("n = %d, x = %f", n, x);
```

prints n as a decimal integer and x as a floating point number, as in

```
n = 17, x = 12.34
```

A similar function scanf performs formatted input conversion.

All the routines mentioned have versions that operate on streams other than the standard input or output, and printf and scanf variants may also process a string, to allow for in-memory format conversion. Other routines in the I/O library transmit whole lines between memory and files, and check for error or end-of-file status.

Many other routines and utilities are used with C, somewhat more on UNIX than on other systems. As an example, it is possible to compile and load a C program so that when the program is run, data are collected on the number of times each function is called and how long it executes. This profile pinpoints the parts of the program that dominate the run-time.

VIII. EXPERIENCE WITH C

C compilers exist for the most widely used machines at Bell Laboratories (the IBM S/370, Honeywell 6000, PDP-11) and perhaps 10 others. Several hundred programmers within Bell Laboratories and many outside use C as their primary programming language.

8.1 Favorable experiences

C has completely displaced assembly language in UNIX programs. All applications code, the C compiler itself, and the operating system (except for about 1000 lines of initial bootstrap, etc.) are written in C. Although compilers or interpreters are available under UNIX for Fortran, Pascal, Algol 68, Snobol, APL, and other languages, most programmers make little use of them. Since C is a relatively low-level language, it is adequately efficient to prevent people from resorting to assembler, and yet sufficiently terse and expressive that its users prefer it to PL/1 or other very large languages.

A language that doesn't have everything is actually easier to program in than

some that do. The limitations of C often imply shorter manuals and easier training and adaptation. Language design, especially when done by a committee, often tends toward including all doubtful features, since there is no quick answer to the advocate who insists that the new feature will be useful to some and can be ignored by others. But this results in long manuals and hierarchies of "experts" who know progressively larger subsets of the language. In practice, if a feature is not used often enough to be familiar and does not complete some structure of syntax or semantics, it should probably be left out. Otherwise, the manual and compiler get bulky, the users get surprises, and it becomes harder and harder to maintain and use the language. It is also desirable to avoid language features that cannot be compiled efficiently; programmers like to feel that the cost of a statement is comparable to the difficulty in writing it. C has thus avoided implementing operations in the language that would have to be performed by subroutine call. As compiler technology improves, some extensions (e.g., structure assignment) are being made to C, but always with some principles in mind.

One direction for possible expansion of the language has been explicitly avoided. Although C is much used for writing operating systems and associated software, there are no facilities for multiprogramming, parallel operations, synchronization, or process control. We believe that making these operations primitives of the language is inappropriate, mostly because language design is hard enough in itself without incorporating into it the design of operating systems. Language facilities of this sort tend to make strong assumptions about the underlying operating system that may match very poorly what it actually does.

8.2 Unfavorable experiences

The design and implementation of C can (or could) be criticized on a number of points. Here we discuss some of the more vulnerable aspects of the language.

8.2.1 Language level

Some users complain that C is an insufficiently high-level language; for example, they want string data types and operations, or variable-size multi-dimensional arrays, or generic functions. Sometimes a suggested extension merely involves lifting some restriction. For example, allowing variable-size arrays would actually simplify the language specification, since it would only involve allowing general expressions in place of constants in certain contexts.

Many other extensions are plausible; since the low level of C was praised in the previous section as an advantage of the language, most will not be further discussed. One is worthy of mention, however. The C language provides no facility for I/O, leaving this job to library routines. The following fragment illustrates one difficulty with this approach:

```
printf("%d\n", x);
```

The problem arises because on machines on which int is not the same as long, x may not be long; if it were, the program must be written

```
printf("%D\n", x);
```

so as to tell printf the length of x. Thus, changing the type of x involves changing not only its declaration, but also other parts of the program. If I/O were built into the language, the association between the type of an expression and the format in which it is printed could be reconciled by the compiler.

8.2.2 Type safety

C has traditionally been permissive in checking whether an expression is used in a context appropriate to its type. A complete list of examples would be long, but two of the most important should illustrate sufficiently. The types of formal arguments of functions are in general not known, and in any case are not checked by the compiler against the actual arguments at each call. Thus in the statement

```
s = sin(1);
```

the fact that the sin routine takes a floating-point argument is not noticed until the erroneous result is detected by the programmer.

In the structure reference

```
p->memb
```

p is simply assumed to point to a structure of which memb is a member; p might even be an integer and not a pointer at all.

Much of the explanation, if not justification, for such laxity is the typeless nature of C's predecessor languages. Fortunately, a justification need no longer be attempted, since a program is now available that detects all common type mismatches. This utility, called lint because it picks bits of fluff from programs, examines a set of files and complains about a great many dubious constructions, ranging from unused or uninitialized variables through the type errors mentioned. Programs that pass unscathed through lint enjoy about as complete freedom from type errors as do Algol 68 programs, with a few exceptions: unions are not checked dynamically, and explicit escapes are available that in effect turn off checking.

Some languages, such as Pascal and Euclid, allow the writer to specify that the value of a given variable may assume only a given subrange of the integers. This facility is often connected with the usage of arrays, in that any array index must be a variable or expression whose type specifies a subset of the set given by the bounds of the array. This approach is not without theoretical difficulties, as suggested by Habermann.[4] In itself it does not solve the problems of variables assuming unexpected values or of accessing outside array bounds; such things must (in general) be detected dynamically. Still, the extra information provided by specifying the permissible range for each variable provides valuable information for the compiler and any verifier program. C has no corresponding facility.

One of the characteristic features of C is its rather complete integration of the notion of pointer and of address arithmetic. Some writers, notably Hoare,[5] have argued against the very notion of pointer. We feel, however, that the facilities offered by pointers are too valuable to give up lightly.

8.2.3 Syntax peculiarities

Some people are annoyed by the terseness of expression that is one of the characteristics of the language. We view C's short operators and general lack of noise as a benefit. For example, the use of braces for grouping instead of begin and end seems appropriate in view of the frequency of the operation. The use of braces even fits well into ordinary mathematical notation.

Terseness can lead to code that is hard to read, however. For example,

```
*++*argv
```

where argv has been declared char **argv (pointer to an array of character pointers) means: select the character pointer pointed at by argv (*argv), increment it by one (++*argv), then fetch the character that *that* pointer points at (*++*argv). This is concise and efficient but reminiscent of APL.

An example of a minor problem is the comment convention, which is PL/1's /* ... */. Comments do not nest, so an effort to "comment out" a section of code will fail if that section contains a comment. And a number of us can testify that it is surprisingly hard to recognize when an "end comment" delimiter has been botched, so that the comment silently continues until the next comment is reached, deleting a line or two of code. It would bemore convenient if a single unique character were reserved to introduce a comment, and if comments always terminated at an end of line.

8.2.4 Semantic peculiarities

There are some occasionally surprising operator precedences. For example,

```
a >> 4 + 5
```

shifts right by 9. Perhaps worse,

```
(x & MASK) == 0
```

must be parenthesized to associate the proper way. Users learn quickly to parenthesize such doubtful cases; and when feasible lint warns of suspicious expressions (including both of these).

We have already mentioned the fact that the case actions in a switch flow through unless explicitly broken. In practice, users write so many switch statements that they become familiar with this behavior and some even prefer it.

Some problems arise from machine differences that are reflected, perhaps unnecessarily, into the semantics of C. For example, the PDP-11 does sign extension on byte fetches, so that a character (viewed arithmetically) can have a value ranging from -128 to +127, rather than 0 to +255. Although the reference manual makes it quite clear that the precise range of a char variable is machine dependent, programmers occasionally succumb to the temptation of using the full range that their local machine can represent, forgetting that their programs may not work on another machine. The fundamental problem, of course, is that C permits small numbers, as well as genuine characters, to be stored in char variables. This might not be necessary if, for example, the notion of subranges (mentioned above) were introduced into the language.

8.2.5 Miscellaneous

C was developed and is generally used in a highly responsive interactive environment, and accordingly the compiler provides few of the services usually associated with batch compilers. For example, it prepares no listing of the source program, no cross reference table, and no indication of the nature of the generated code. Such facilities are available, but they are separate programs, not parts of the compiler. Programmers used to batch environments may find it hard to live without giant listings; we would find it hard to use them.

IX. CONCLUSIONS AND FUTURE DIRECTIONS

C has continued to develop in recent years, mostly by upwardly compatible extensions, occasionally by restrictions against manifestly nonportable or illegal programs that happened to be compiled into something useful. The most recent major changes were motivated by the extension of C to other machines, and the resulting emphasis on portability. The advent of union and of casts reflects a desire to be more precise about types when moving to other machines is in prospect. These changes have had relatively little effect on programmers who remained entirely on the UNIX system. Of more importance was a new library, which changed the use of a "portable" library from an option into an effective standard, while simultaneously increasing the efficiency of the library so that users would not object.

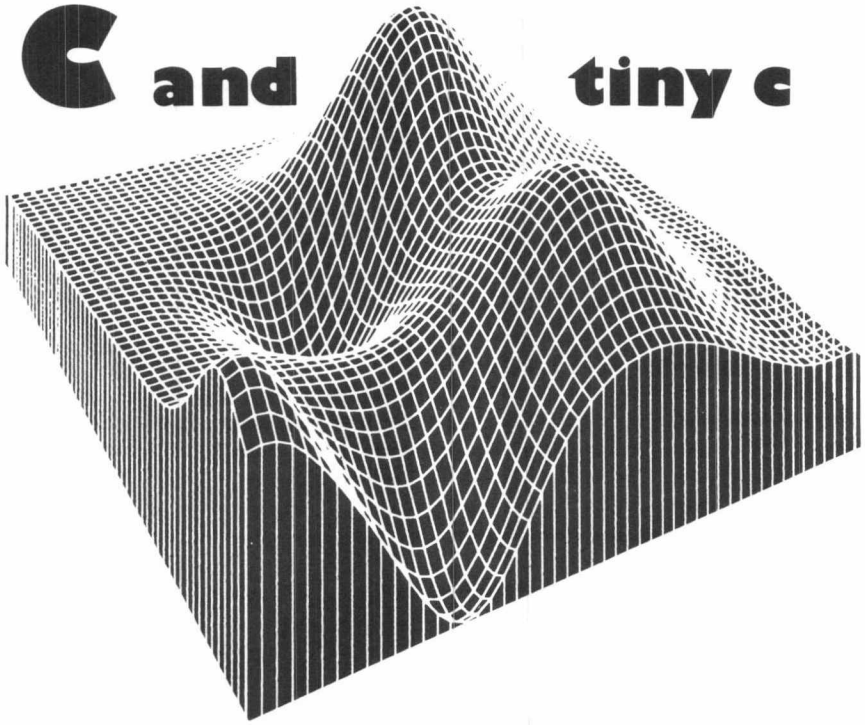
It is more difficult, of course, to speculate about the future. C is now encountering more and more foreign environments, and this is producting many demands for C to adapt itself to the hardware, and particularly to the operating systems, of other machines. Bit fields, for example, are a response to a request to describe externally imposed data layouts. Similarly, the procedures for external storage allocation and referencing have been made tighter to conform to requirements on other systems. Portability of the basic language seems well handled, but interactions with operating systems grow ever more complex. These lead to requests for more sophisticated data descriptions and initializations, and even for assembler windows. Further changes of this sort are likely.

What is not likely is a fundamental change in the level of the language. Realistically, the very acceptance of C has compelled changes to be made only most cautiously and compatibly. Should the pressure for improvements become too strong for the language to accommodate, C would probably have to be left as is, and a totally new language developed. We leave it to the reader to speculate on whether it should be called D or P.

REFERENCES

1. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Englewood Cliffs, N.J.: Prentice-Hall, 1978.
2. M. Richards, "BCPL: A Tool for Compiler Writing and Systems Programming," Proc. AFIPS SJCC, *34* (1969), pp. 557-566.
3. S. C. Johnson and B. W. Kernighan, "The Programming Language B," Comp. Sci. Tech Rep. No. 8, Bell Laboratories (January 1973).
4. A. N. Habermann, "Critical Comments on the Programming Language PASCAL," Acta Informatica, *3* (1973), pp. 47-58.
5. C. A. R. Hoare, "Data Reliability," ACM SIGPLAN Notices, *10* (June 1975), pp. 528-533.

Structured Programming, C and tiny c



BY T. A. GIBSON AND S. B. GUTHERY
tiny-c associates
P.O. Box 269
Holmdel, N.J. 07733

Perhaps you have heard structured programming described as "go-to-less" programming. Or programming with just if-then-else and do-while control statements. Such remarks oversimplify what structured programming is all about. The essence of structured programming is PROGRAM CLARITY. You can write programs in small, modular parts with easy-to-follow program flow. You can use well-chosen, descriptive variable names. This leads to clear, humanly understandable programs. That is the real objective of structured programming.

We discuss here three principle ideas that make program clarity possible. These are: modularity, predictable program flow, and local variables.

MODULARITY in software is just as important as modularity in hardware. It makes it humanly possible to deal with complexity. A module is a brick or atom used for building bigger modules. Seen from within, a module may be very complex but from the outside it is an indivisible whole. Software modularity is achieved through the use of FUNCTIONS.

© *1980, tiny-c associates*

PROGRAM FLOW is predictable if you can point to any statement and easily answer the question "under what conditions is this statement executed?" This is particularly important if the program is 20 or 30 pages long and still has bugs. Scanning the whole program and drawing arrows is no fair. That's not considered an easy way to answer the question. Predictable program flow can be achieved in many ways. In C, it is done with COMPOUND STATEMENTS.

Functions also make it possible to hide variables used in a strictly local context. The variable n is very popular; it's used frequently to count things. Have you ever had a program blow up because you were using n in two places for two purposes? The fix was to change one of them to n1. A better idea is in the concept of LOCAL and GLOBAL variables.

David Gries suggests structured programming be called "simplicity theory," and characterizes it as "an approach to understanding the complete programming process" [Gries 1974]. As a pleasant dividend, structured programming is more enjoyable than monolithic programming. It should certainly, therefore, be a part of personal computing. To begin our look at tiny-c as a structured programming language, let's look at the foundation of functions and predictable program flow – the compound statement.

Compound Statements

When you write a program, you write a list of statements:

```
x = x – 1
a = b + c
b = b * 2 –c
x = b – a
```

The idea behind a compound statement is to make one statement – a molecule – out of a set of statements – some atoms. This is done in tiny-c by

```
[ x = x – 1
  a = b + c
  b = b * 2 – c
  x = b – a ]
```

Anywhere you can write a simple statement you can also write a compound statement. This sounds simple, but the effect is powerful. For example most programming languages have an if statement similar to this:

```
if (logical expression) statement
```

So you can write

```
if (x>0)  x = x – 1
```

But make the statement part compound, and you have this capability:

```
if (x>0) [
   b = b*2–c
   a = b + c
   x = x – 1
]
```

This multiline if is not some special kind of if. It is still:

```
if (logical expression) statement
```

But the statement part is compound. The compound statement is treated as an indivisible unit. It is either all done or all not done depending on the value of the logical expression.

The compound statement also is a natural for LOOPS. There is a big difference among the various programming languages in how you write loops, but they all have one thing in common. A loop has a beginning and an end. A compound statement can be used to express this. The looping statement is:

```
while (logical expression) statement
```

Notice the similarity with the if. Only the keyword has changed. Here's how while works. The logical expression is evaluated. If it is true, then the statement is executed, and then the while is done again. The effect is a repeated if, i.e., a loop. As long as the logical expression remains true, the statement is done again and again. Eventually something in the statement causes the logical expression to become false, and the loop terminates. Of course, the statement can be compound, as in:

```
while (x>0) [
  a = b+c
  b = b*2-c
  x = x-1
  ]
```

The compound statement is a natural way to delimit the beginning and end of loops.

With one simple idea, the compound statement, two things are achieved. The if statement is more powerful than is common in non-structured programming languages. The concept of a loop collapses to a simple repeated if or while statement. In both situations you are stating conditions under which the statement – whether simple or compound – is to be executed.

Nesting Compound Statements

> ANYWHERE YOU CAN WRITE A SIMPLE STATEMENT, YOU CAN WRITE A COMPOUND STATEMENT

That is a fundamental rule. A compound statement contains simple statements. Therefore a compound statment can contain compound statements. Figure 1 illustrates this.

FIGURE 1

```
[  x = x - 1
   a = b+c
   b = b*2 - a
   x = b-a
]
```

a = b+c is a simple statement. The rule says a compound statement can be written here. For example:

```
[  x = x-1
   [   a=b+c
       w=y+2*x+w
       y=17
   ]
   b = b*2-a
   x = b-a
]
```

The substitution of a compound for a simple shown in Figure 1 is certainly allowable, but is of no practical value.

The real utility in nested compounds is in writing nested if and while statements. Figure 2 is therefore a more realistic example of the use of compound statements.

FIGURE 2

```
if (x>0) [
    while (x<limit) [
      if (case==1) [
        y = 0; w = 99
      ]
      if (case==2) [
         y = 99; w = 0
      ]
      nextaction
      x = x+1
    ]
]
```

In Figure 2 if you remove everything except the brackets, you have this:

```
[ [ [ ] [ ] ] ]
```

This is what is meant by nested compound statements. Brackets are used to form program units the same way parentheses are used to create arithmetic statements. The main difference is that a pair of brackets is preceded by a function name, or a logical expression. In the first case you're naming the contents of the brackets and in the second you're stating the conditions under which the contents are to be executed.

Readable Program Flow

In Figure 2, look at the "y = 0" in the fourth line. How can it be reached? Only if case is 1, and x is less than limit. No go-to can lead here, either accidentally or on purpose.

How can "nextaction" be reached? Only if x is less than limit, and then only after possible changes to y and w. This program has simple, predictable flow. The only way a statement other than a while can be reached is from directly above. Whiles can also be reached from their matching] below.

Indenting and the Placement of Brackets in Compound Statements

The brackets alone define the "structure" of a program. Indenting means nothing. But one of the purposes of structured programming is to make programs more readable and, hence, more understandable. A good choice of indenting style is very important to program readability. There are several styles to choose from. The actual choice is not too important. But once you choose a style, stick to it. Consistency IS important.

One easily explained style is to align matching brackets vertically. This looks like:

```
if (x>0)
[ statement
  statement
      "
      "
      "
]
```

A problem with this is that when editing the first statement, care must be taken to keep the [intact. So some use this style:

```
if (x>0)
[
 statement
 statement
      "
      "
      "
]
```

This takes an extra line. Also there is a visual break between the if and its statements. So some take the left bracket and move it to the end of the preceding line:

```
if (x>0) [
    statement
    statement
        "
        "
        "
]
```

The right bracket is now vertically aligned with the if or while that preceded the compound statement.

You may pick one of these, or invent a style of your own. But, we repeat, whatever you decide to do, do it consistently.

Functions

A large software project can usually be broken into natural parts, and each part programmed and debugged as a separate unit. Each of these units then becomes a reliable building block for the construction of still larger parts of the project. Sometimes units can be designed to be useful in many projects.

In various programming languages these building blocks are called subprograms, subroutines, or, as in C, FUNC-

TIONS. Here is a tiny-c function for any computer versus human game:

```
game [
    getready
    while (stillplaying ( ) ) [
        humanturn
        if (stillplaying ( ) ) computerturn
    ]
    gameover
]
```

The name of the function is "game". The compound statement that follows is called the body of the function. Each [can be read as "do all of this", and its matching] read as "end of this". Game divides the design of a game program into five parts:

> getready (which initializes things, and prints instructions if requested),
> stillplaying (which determines if the game is still going, and returns true if it is, otherwise false),
> humanturn (which conducts the human's turn),
> computerturn (which conducts the computer's turn),
> gameover (which computes and prints scores, makes remarks about the human's skill, promotes the human, or whatever).

The game function is the first step in divide-and-conquer or top-down program development. Let's carry this development one step further. The getready function can be expanded this way:

```
getready [
    ps "Do you want instructions?"
    if (gc ( ) =='y') instructions
]
```

Getready divides the initialization into two parts: instructions, and setupboard. (*Note*: ps prints a character string, gc( ) reads a character, and =='y' tests if the character is a y.)

Notice that both game and getready are universal. They can be used in many game programs. Programming in this fashion eventually leads to a library of useful, general purposes functions these can be pulled off the shelf into a software project. You know they work because they were used before. Your programming becomes more productive, and more pleasant.

The next time you're programming a sizable project, i.e., anything more than a page, try to identify subsets of the logic usable in other projects. Capture these as functions.

> It is gratifying to discover a general purpose function where none was suspected.

Local and Global Variables

A LOCAL VARIABLE is one that is known only inside a function. It can be used and changed only within the body of the function. Even its name is unknown outside the function. In fact, its name can be used in other functions without conflict. This is what makes local variables useful .

Take a look at Figure 3. There are four local variables in these two functions. The variables n and maximum are local to a function. The variables n and total are local to anotherfunction. If either of these functions calls the other, the values of n will not be confused since they only have meaning inside the body of their own functions. It helps to think of local variable names as being preceded by the possessive form of the function to which they are local; for example, afunction's n and anotherfunction's n.

FIGURE 3

```
afunction [
  int n, maximum
  n=0
  while (n<maximum) [
      .
      .
      .
   n=n+1
  ]
]
anotherfunction [
  int n, total
      .
      .
      .
  n = n+2
  total = total+n
      .
      .
      .
]
```

The value of locals is obvious to anyone who has spent a nasty debugging session trying to find out where, in a huge program, some variable is getting changed.

Of course not all variables can be local. Some must be shared by many functions. These are called GLOBALS. They should be used infrequently, as they do cause debugging headaches. Choosing good, descriptive names for globals alleviates the problem. A global named "k"is inviting disaster. Call it "klingonsleft" and you're less likely to accidentally use it for two purposes. Also you've given a reader of your program a pretty good clue to the variable's use.

Summary of Structured Programming

These concepts: functions, locals and globals, and compound statements are probably the main tools of structured programming. It's really that simple. Many variations and improvements are possible. The most important improvement is to the concept of locals and globals. This is a special case of the more general notion of nested scope of variables. But we will not go into that here.

Tiny-c and (BIG) C

The examples in this article all use tiny-c. If you move to BIG C, there are a few differences. Tiny-c is a training language, just like BASIC. Technically, it is a COGNATE of C, not a subset. The syntax differences are:

> In C, all statements must end with a semi colon(;). In tiny-c they end with the end of a line, or a ; or].
> or a ; or].
> In C, all functions must have parentheses around the arguments. In tiny-c they can frequently be omitted.
> In C, compound statements use { } instead of [ ], array references use [ ] instead of ( ).

So figure 2 in C looks like:

```
if (x>0){
  while (x< limit){
    if (case ==1){
      y = 0; w = 99;
    }
    if (case ==2){
      y = 99; w = 0;
    }
    nextaction ( );
    x = x+1;
  }
}
```

Notice that the function call nextaction ( ); must have the parenthesis, even with no arguments. And the semicolons shown must all be there.

The game function is written in C like this:

```
game ( ) ; {
  getready( ) ;
  while (stillplaying ( )) {
    humanturn( ) ;
    if (stillplaying( )) computerturn( ) ;
  }
  gameover( ) ;
}
```

There are also differences in the input/output library functions. These differences are less important than the syntax differences, because in C input/output is done with function calls, and not built into the syntax of the language.

In C, to print an integer requires this:

```
printf ( "%d", x) ;
```

In tiny-c this suffices:

```
pn x
```

Summary of tiny-c and C

In deriving a training cognate of C, we moved to lower case characters ([instead of {), removed punctuation distracting to students, and introduced much simpler input/output functions. The resulting language, tiny-c, is as easy to learn as BASIC, and serves as a good stepping stone into structured programming.

BIBLIOGRAPHY

Gibson, T. and Guthery, S., *The tiny-c Owner's Manual*, tiny c associates, 1978.

Gibson, Paul, "Stopwatch: A tiny-c Program," CREATIVE COMPUTING, Vol. 5, No. 3 (March 1979).

Gibson, Tom, "Caution: Structured Programming Can Be Habit-Forming," CREATIVE COMPUTING, Vol. 5, No. 1 (January 1979).

Gries, David, "On Structured Programming – A Reply to Smoliar," CACM, Vol. 17, No. 11 (November 1974).

Hancock, L., "Growing, Pruning & Climbing Binary Trees with tiny-c," DR. DOBB'S JOURNAL, Vol. 4, Issue 6 (June/July 1979).

Hancock, L., "Dear Doctors, " letter to DR. DOBB'S JOURNAL, Vol. 4, Issue 3 (March 1979), on page 9.

Hancock, L., "Implementing a tiny interpreter with a CP/M flavored C," DR. DOBB'S JOURNAL, Vol. 5, Issue 1, (January 1980).

Kern, Christopher O., "A User's Look at tiny-c," BYTE, Vol. 4, No. 12 (December 1979).

Kernighan, Brian W. and Ritchie, Dennis M., THE C PROGRAMMING LANGUAGE, Prentice-Hall Software Series, 1978.

Madden, J. Gregory, "C: A Language for Microprocessors?," BYTE, Vol. 2, No. 10 (October 1977).

Newberry, Steve, "A Great Way to Learn Structured Programming, tiny–c," part of the article "An Overview of CP/M Compatible Software," DR. DOBB'S JOURNAL, Vol. 5, Issue 1 (January 1980).

Nussbaum, Frank, "Reader Cs Pascal Alternative," letter to BYTE, page 142, Vol. 3, No. 11 (November 1978).

Shapin, Ted, "Review of tiny-c Owner's Manual," DR. DOBB'S JOURNAL, Vol. 4, Issue 2 (February 1979).

Zahn, C. T., *C NOTES: A Guide to the C Programming Language*, Yourdon Press, 1979.

HELP WITH OSI'S CPM

BY LEE BARKER
Chicago, Il.

OSI's Fortran, Basic, and Cobol CP/M package was purchased a year ago and a number of problems encountered. The most serious of these being that Cobol wouldn't work. Last month the updated version was obtained with some of the old problems fixed, and a few new ones added. Enclosed is a modified listing of the OSI's 10 routines that corrects a few of these problems.

Not covered here is the most serious problem with the new operating system. There is a routine in 6502 code, 81 bytes into track 0 to initialize the 16 ports starting at 0CF00H. CP/M in loading overwrites these ports. If these ports are used before being re-initialized the system will lock.

Other problems include—control S works perhaps 20% of the time, but not at all in basic. Microsoft's Basic was designed for the 8080 using the IN, OUT instructions. Since the OSI uses ACIAs Basic's IN, OUT, WAIT do not work. Also if the control character 9 is needed for cursor addressing or the printer, a string of CHR$(32)s will be obtained instead. (TAB) As an alternative, the CURSOR.ASM listing is a method of associating a string directly with an address on the screen. This simplifies building screens in that only one variable is needed to describe a string and its location rather than the usual four.

I hope these listings will provide some help to others who may have had similar problems.

P.S. To Re-init ports

Fix:	LXI	H,0CF2O
	DCR	L
	DCR	L
	MVJ	M,3
	MVJ	M,17
	JNZ	Fix + 3

```
1:
2:                      ; title   OSIBIOS.ASM
3:                      ; Rewritten 1-O routines for the OSI with CP/M
4:                      ;
5:                      ; Set up for CRT at OFCOOH
6:                      ;           LPT at OCFOOH
7:                      ;           MODEM at OCF02H
8:                      ;
9:                      ; Problems with system as delivered:
10:                     ; 1.    ^S usually didn't work with CP/M
11:                     ; 2.    ^S didn't work at all in Basic
12:                     ; 3.    CHR$(9) prints as a string of spaces
13:                     ; 4.    OSI uses ACIA's so IN OUT & WAIT don't work
14:                     ; 5.    Program's must be rewritten to redirect output
15:                     ; 6.    Track 0 initialized the 16 ports at OCFOOH
16:                     ;         which CP/M in loading would clobber.
17:                     ;         If any of them were used, the system would lock.
18:                     ;
19:                     ; Corrections:
20:                     ; are for #1 to 5.  #6 is handled seperately.
21:
22: BE00                ORG     OBE00H                  ; jump table
23: FC00 =              CRT     EQU     OFC00H          ; Main terminal port
24: CF00 =              LPT     EQU     OCF00H          ; Line printer
25:
26:                     ;
27:                     ; CP/M jump table
28: BE00 CD2DBE         START:  CALL    CONT            ; Cold
29: BE03 CD2DBE                 CALL    CONT            ; Warm
30: BE06 C300BF                 JMP     READY
31: BE09 C309BF                 JMP     TTYIN
32: BE0C C316BF                 JMP     TTYOUT
33: BE0F C328BF                 JMP     PRINT           ; Line printer
34: BE12 C345BF                 JMP     POUT            ; Peripneral
35: BE15 C338BF                 JMP     PIN
36: BE18 CD2DBE                 CALL    CONT
37: BE1B CD2DBE                 CALL    CONT
38: BE1E CD2DBE                 CALL    CONT
39: BE21 CD2DBE                 CALL    CONT
40: BE24 CD2DBE                 CALL    CONT
41: BE27 CD2DBE                 CALL    CONT
42: BE2A CD2DBE                 CALL    CONT
43:                     CONT:
44:                     ;
45: BF00                ORG     OBF00H          ; CP/M OSI 1-O
46:                     ;
47:                     ; Was a key pressed on the CRT?
48: BF00 3A00FC         READY:  LDA     CRT
49: BF03 1F                     RAR
50: BF04 3E00                   MVI     A,0
51: BF06 D0                     RNC
52: BF07 3D                     DCR     A
53: BF08 C9                     RET
54:                     ;
55:                     ; Input a character from the CRT.
56: BF09 3A00FC         TTYIN:  LDA     CRT
57: BF0C 1F                     RAR
58: BF0D D209BF                 JNC     TTYIN
59: BF10 3A01FC                 LDA     CRT+1
60: BF13 E67F                   ANI     127
61: BF15 C9                     RET
62:                     ;
63:                     ; Output a character to the CRT.
64: BF16 F5             TTYOUT: PUSH    PSW
65: BF17 3A00FC                 LDA     CRT
66: BF1A 1F                     RAR
67: BF1B DA57BF                 JC      TEST
68: BF1E 1F                     RAR
69: BF1F D217BF                 JNC     TTYOUT+1
70: BF22 79                     MOV     A,C
71: BF23 3201FC                 STA     CRT+1
72: BF26 F1                     POP     PSW
73: BF27 C9                     RET
74:                     ;
75:                     ; Output a character to the line printer.
76: BF28 E5             PRINT:  PUSH    H
77: BF29 F5                     PUSH    PSW
78: BF2A 2A53BF                 LHLD    ADDR
79: BF2D 7E             PLOOP:  MOV     A,M
80: BF2E 1F                     RAR
81: BF2F 1F                     RAR
82: BF30 D22DBF                 JNC     PLOOP
83: BF33 23                     INX     H
84: BF34 71                     MOV     M,C
85: BF35 F1                     POP     PSW
86: BF36 E1                     POP     H
87: BF37 C9                     RET
88:                     ;
89:                     ; Input a chr. from the modem (ADDR+2)
90: BF38 2A55BF         PIN:    LHLD    ADDR+2
91: BF3B 7E                     MOV     A,M
92: BF3C 1F                     RAR
93: BF3D D23BBF                 JNC     PIN+3
94: BF40 23                     INX     H
95: BF41 7E                     MOV     A,M
96: BF42 E67F                   ANI     127
97: BF44 C9                     RET
98:                     ;
99:                     ; Output a chr. to the modem.
100: BF45 2A55BF        POUT:   LHLD    ADDR+2
```

```
BF48 F5             PUSH    PSW
BF49 7E             MOV     A,M
BF4A 1F             RAR
BF4B 1F             RAR
BF4C D249BF         JNC     POUT+4
BF4F 23             INX     H
BF50 71             MOV     M,C
BF51 F1             POP     PSW
BF52 C9             RET
                    ;
BF53 00CF00CF ADDR: DB      00,207,00,207
                    ;
                    ; Test for what key was entered.
BF57 3A01FC   TEST: LDA     CRT+1
BF5A E67F           ANI     127
BF5C FE13           CPI     19              ; ^S
BF5E C217BF         JNZ     TTYOUT+1
BF61 C357BF         JMP     TEST
                    ;
                    ; Single entry point for the following routines.
BF64 3D       USR:  DCR     A
BF65 3D             DCR     A       ; If passed integer
BF66 CA79BF         JZ      INTEG   ; Print CHR$(integer mod 256)
BF69 3D             DCR     A
BF6A CA7DBF         JZ      STRING  ; Print exact string to LPT
                    ;
                    ; Input a single character.
BF6D CD09BF   CHAR: CALL    TTYIN
BF70 6F             MOV     L,A
BF71 2600           MVI     H,0
BF73 E5             PUSH    H
BF74 2A0501         LHLD    0105H
BF77 E3             XTHL
BF78 C9             RET
                    ;
                    ; Print the CHR passed as an integer 0-255.
BF79 4E       INTEG: MOV    C,M
BF7A C316BF         JMP     TTYOUT
                    ;
                    ; Print a string to the line printer (inc CHR$(9)).
BF7D EB       STRING: XCHG
BF7E 46             MOV     B,M
BF7F 23             INX     H
BF80 5E             MOV     E,M
BF81 23             INX     H
BF82 56             MOV     D,M
BF83 EB             XCHG
BF84 04             INR     B
BF85 05       SLOOP: DCR    B
BF86 C8             RZ
BF87 4E             MOV     C,M
BF88 23             INX     H
BF89 CD00CF         CALL    LPT
BF8C C385BF         JMP     SLOOP
                    ;
BF8F                END
```

```
1:
2:
3:                  ; title         CURSOR.ASM
4:                  ; USR call to print string with direct cursor addressing.
5:                  ; String is in the form "#yyxx........"
6:
7:  BF16 =          CRT     EQU     0BF16H  ; prints to CRT reg. C
8:  BF09 =          CRTIN   EQU     0BF09H  ; Returns entered key in reg A
9:
10: BF64            ORG     0BF64H
11:                 ;
12: BF64 3D         START:  DCR     A
13: BF65 3D                 DCR     A       ; Integer passed?
14: BF66 CA79BF             JZ      INTEG   ; Yup
15: BF69 3D                 DCR     A
16: BF6A CA7DBF             JZ      STRING
17: BF6D CD09BF             CALL    CRTIN
18: BF70 6F                 MOV     L,A
19: BF71 2600               MVI     H,0
20: BF73 E5                 PUSH    H
21: BF74 2A0501             LHLD    0105H   ; MAKINT
22: BF77 E3                 XTHL
23: BF78 C9                 RET
24:                 ;
25: BF79 4E         INTEG:  MOV     C,M
26: BF7A C316BF             JMP     CRT
27:                 ;
28: BF7D EB         STRING: XCHG
29: BF7E 7E                 MOV     A,M
30: BF7F 46                 MOV     B,M
31: BF80 04                 INR     B
32: BF81 23                 INX     H
33: BF82 5E                 MOV     E,M
34: BF83 23                 INX     H
35: BF84 56                 MOV     D,M     ; DE points to string
36: BF85 EB                 XCHG
37: BF86 FE05               CPI     5       ; A = len.
38: BF88 DABEBF             JC      SKIP
39: BF8B 7E                 MOV     A,M
40: BF8C FE23               CPI     35      ; "#"
41: BF8E C2BEBF             JNZ     SKIP
```

```
42: BF91 E5         PUSH    H       ; save in case plain string
43: BF92 CDC8BF     CALL    TEN
44: BF95 D2D9BF     JNC     FAIL1   ; Y wasn't numeric
45: BF98 83         ADD     E
46: BF99 FE18       CPI     24      ; Y
47: BF9B D2D9BF     JNC     FAIL1   ; Y > 0 to 23
48: BF9E 57         MOV     D,A
49: BF9F CDC8BF     CALL    TEN
50: BFA2 D2D9BF     JNC     FAIL1   ; X wasn't numeric
51: BFA5 83         ADD     E
52: BFA6 FE50       CPI     80
53: BFA8 D2D9BF     JNC     FAIL1   ; X > 0 to 79
54: BFAB 0E14       MVI     C,20    ; Cursor address code
55: BFAD CD16BF     CALL    CRT
56: BFB0 4A         MOV     C,D
57: BFB1 CD16BF     CALL    CRT
58: BFB4 4F         MOV     C,A
59: BFB5 CD16BF     CALL    CRT
60: BFB8 23         INX     H       ; first byte of string
61: BFB9 D1         POP     D       ; didn't need it
62: BFBA 78         MOV     A,B
63: BFBB D605       SUI     5
64: BFBD 47         MOV     B,A
65: BFBE 05   SKIP: DCR     B
66: BFBF C8         RZ              ; done
67: BFC0 4E         MOV     C,M
68: BFC1 23         INX     H
69: BFC2 CD16BF     CALL    CRT
70: BFC5 C3BEBF     JMP     SKIP
71: BFC8 CDD2BF TEN: CALL   TEST
72: BFCB D0         RNC
73: BFCC 07         RLC
74: BFCD 5F         MOV     E,A
75: BFCE 07         RLC
76: BFCF 07         RLC
77: BFD0 83         ADD     E       ; times 10
78: BFD1 5F         MOV     E,A
79: BFD2 23   TEST: INX     H
80: BFD3 7E         MOV     A,M
81: BFD4 D630       SUI     48
82: BFD6 FE0A       CPI     10
83: BFD8 C9         RET
84:             ;
85: BFD9 E1   FAIL1: POP    H
86: BFDA C3BEBF     JMP     SKIP    ; print string
87:
88: BFDD            END
```

access addend ALGORITHM algebra analog application ai

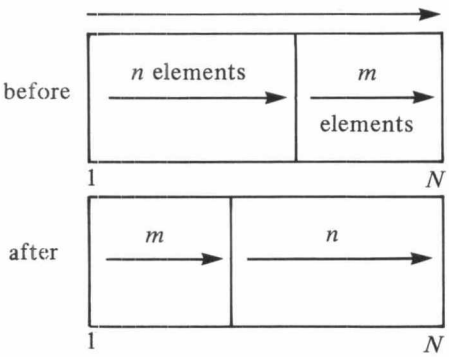
BY DENNIS ALLISON

ALGORITHMS

Algorithms are published in many places. For many years there was an algorithms column in the *Communications of the ACM* (it's now published as part of ACM's *Transactions on Mathematical Software*). I learned a lot reading those algorithms and trying to understand how they work); they are still worth reading and understanding.

BLOCK INTERCHANGE

Suppose we have an array of N objects which we partition into two parts, one with m objects in it and the other with $n = N - m$. The problem is to interchange the two parts with a minimum of data moves and no significant temporary storage. Symbolically we can represent the task as



The program below is a modification of an algorithm published by Walter Fletcher [ACM Algorithm 284, CACM 9:5, May 1966] based upon some results of group theory.

A group is a mathematical system consisting of a non-empty set of elements and a single binary operation, ∘, which obeys the axioms:

Associativity: $a \circ (b \circ c) = (a \circ b) \circ c$ for all a, b, c in the group.
Identity Element: there is an element i in the group such that $a \circ i = i \circ a = a$ for any a in the group.
Inverse Element: for any a in the group there exists an element a' in the group (the inverse of a) such that $a \circ a' = i$

If one also has the axiom

Commutativity: $a \circ b = b \circ a$ for all a and b in the group.

then the group is an Abelian group and a number of useful and interesting properties follow immediately.

Consider the additive group of integers modulo N. The elements of the group are the integers 0, 1, . . . , N−1. The operation is ordinary addition except that results greater than or equal to N are reduced by N.

If we partition N into two elements so that N = m + n, then the elements 0, n, 2n, 3n, . . . form a cyclic subgroup. A subgroup is simply a subset of the elements of a group which themselves form a group. A cyclic group is one which for all elements in the group

$$a \circ a \circ a \circ \ldots \circ a = a$$

for some number of repeated operations. The number of operations required is called the order of the group. For our group this is given by $N/gcd(m,n)$ where $gcd(m,n)$ is the greatest-common-divisor. Each of the elements 1, 2, 3, . . . belongs to a distinct coset of the group and form a disjoint covering. There are, of course, gcd(m,n) cosets.

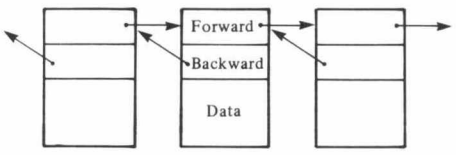
Fletcher's published algorithm computed the gcd(m,n) and then ran two loops: an outer loop which selected one of the cosets and an inner loop which selected an element, interchanged it with the next generated element of the subgroup n locations forward (modulo N) and continued until the entire coset was moved.

My version of the algorithm, programmed below in C, does not compute the gcd. Again, there are two loops: an outer loop which simply counts the total number of interchanges which must, eventually, equal N. The inner loop simply looks for the periodic return to the initial value and then selects the next coset. All in all, an elegant solution with only N interchanges.

```
interchange( a, m, N) int a[], m, N; {
        int i,j,n,t,tt;
        i = 0; n = N-m;
        while(N) {
                j = ++i;
                t = a[i];
                do {
                        j = (j<=m)?(j+n):(j-m);
                        tt = a[j]; a[j] = t; t = tt;
                        N--;
                } while (j!=i);
        }
}
```

SAVING THE POINTER IN DOUBLY LINKED LISTS

A doubly linked list consists of nodes which contain two pointers and the associated data. One pointer points forward to the next list node; the other points backward to the previous node:



The flexibility of a doubly linked list is often offset by the additional storage required for the second pointer. For many algorithms which traverse doubly linked lists one can use a trick and save the storage space required for one of the pointers.

The trick relies upon a property of the exclusive-or operation which is available on most computers. For any numbers *a* and *b* one has

$$a = (a \wedge b) \wedge b ,$$

where ^ is the exclusive- or operator. The trick is to store the exclusive-or of the FWD and BKWD pointers at each node. Traversal of the list then requires three pieces of information: the old direction, the address of the current node (to decode the next node), and the address of the last node visited.

To see how this works, suppose we are at node N_5 and we've just come from node N_4 so both addresses are available. Using N_i to indicate the *address* of the node, the contents of the link fields would be

$$N_4 : \boxed{\begin{array}{c} N_3 \wedge N_5 \\ \hline \text{Data} \end{array}} \quad N_5 : \boxed{\begin{array}{c} N_4 \wedge N_6 \\ \hline \text{Data} \end{array}} \quad N_6 : \boxed{\begin{array}{c} N_5 \wedge N_7 \\ \hline \text{Data} \end{array}}$$

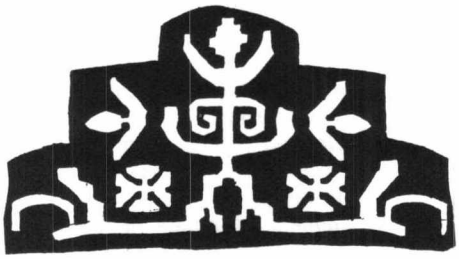
and we could get a forward move by

$$N_6 = N_4 \wedge (N_4 \wedge N_6)$$

and a backwards move directly from the stored address. Of course one would have to update the window information to allow successive moves in any direction.

Notice that the program must know something of the list structure in order to interpret the pointer field. Considered just as values they are inscrutable!

I don't know much about the history of this trick. Charles Wetherell (author of "Programming Pastimes and Pleasures" in this journal) showed it to me with considerable glee some years ago; he'd found it, if memory serves, in block memory allocation and recovery support system in an operating system.



MITS DOS Disk to Disk Copy Routine

BY ANDREW L. BENDER, M. D.
Neurological Services Inc.
Westwood, NJ 07675

Often there is a need to prepare floppy disks, or removable packs, with sets of "standard" predetermined files. One example is the construction of "library disks;" another is the preparation of distribution copies of programs. Being in the position of needing both library disks and distribution disks I had used the "COP" routine supplied with MITS DOS. This routine is quite fast and very flexible; however, it is difficult to use. It must be invoked each time you want to copy a single file; six parameters must be given for each file copied. If anything goes wrong during the copy or assignment of files, the routine unforgivingly aborts with little or no information as to why it failed.

In order to remedy the above faults and ease the burden of preparing libraries, I wrote a disk to disk copy program which I call "CPY." CPY will accept a list of files to be copied in a text file which you, the user, prepare with the text editor "EDIT" system program. The list of files gives the name and disk number of each file to be copied. The CPY program is given the number of an output disk and the files in the list are copied one by one to that disk. Once the copy has started there is no need for the user to intervene. The files must retain the same name on the output disk as on the input disk.

During the copy several things are checked and certain actions are taken. If a file is specified as being on the output disk in the list, no copy will occur. In the case of a file being specified to be copied onto the output disk when it already resides on the output disk, no copy will occur. If a file cannot be found to be copied, the directive is just ignored as if it were not present. An I/O error will abort the copy.

The philosophy in structuring the error checking and handling was that the routine was going to be used only by people familiar with files, file structures and data processing. It was *not* written to be used as an application program by a general user. Whenever an error occurs during processing the error code number is printed along with the file number on which the error occurred.

Use of CPY

To call the program:
. CPY textfilename, textfile disk, outputdisk
(. supplied by DOS)

Where textfilename is the name of the list of files prepared with the text editor program. Note that the leading ampersand (&) in the file name should *not* be included as CPY will supply this internally. Textfile disk is the disk unit on which the file list can be found. Outputdisk is the number of the disk onto which the files will be copied. For example if you wish to copy the files specified in RLIB$ on disk zero to disk one the call would be: . CPY RLIB$,0,1

Let's carry the example along: Suppose the file directory of disk zero looked like this: *HEXOUT *OCTOCT *OPEN$ *HAZ2000 #LINK #ASM #EDIT #CPY #FDMP #ABSDMP #LDR #MAP &RLIB$ &MAPSYM &BLANG &CP Let's further assume the directory of disk one looked like: #INIT #LINK #DEBUG #EDIT #CNS #COP #ASM Let's further assume that RLIB$ on disk zero looked like:

```
00100  *HEXOUT, 0
00110  *DECOUT, 0
00120  *OCTOCT, 0
00130  #CPY, 0
00140  #ASM, 0
```

Now let's trace the operation of the routine: Processing the first file in the list (*HEXOUT, 0) the CPY program will copy its text from disk zero to disk one. The next file *DECOUT will not be found on disk zero, so a message that the requested file wasn't found on the input disk will be printed. The program will then examine the requests on lines 120 and 130 copying the text of both files from disk zero to disk one. At line 140 the file #ASM is already on disk one so it will not do any copy. The CPY program will then exit to DOS since line 140 was the last line in the file.

DISK CONTROL

Just prior to initiation of copy, the program will ask if the output disk is to be unloaded. At this time, if you want to put a disk on the output drive you should change the disk *before* typing in the response which can be Y or N. If you state Y, the disk will be dismounted, then an immediate mount will be issued. It is this action that makes it necessary to physically change disks *before* answering the message. You can, of course, rewrite that portion of the program if that annoys you, but it seems to be the fastest way to handle it. Keying in "N" assumes that you want to copy onto the current output disk.

PROGRAM DOCUMENTATION

The program listing documents what is happening here. I used several undocumented MITS DOS entry points. They are documented in the appendix. Two text messages in DOS were also used: 010A6H is the start of "ENTER FILE NAME" and 0101DH is the start of "ENTER DRIVE". Two parameters used in DOS were also used: 01321H which is a byte containing the highest disk number and 01322H is a byte which says how many sequential files are permitted. I would recommend putting all of these entries into your SYSENT element. I called the number of disks ND$$$ and the number of files NF$$$.

PROGRAM TESTING

I tested this program for about a month on a two drive system. I never tested it on drive numbers higher than 1. I think it will work without problems because I used the conversion subroutines for larger numbers. It is a real pain getting a routine up and running only to find that you have become part of the debugging. I tried to avoid putting the user in this position. There doesn't seem to be any problem, but if you find one please let me know. One note of caution concerns attempts to copy the list of files with this program. Remember that during program operation the list of files (FILE #3) is open for input. This condition will not allow you to open this file as an input file (FILE #1) for copying. You can handle this situation in two ways: Use COP to copy the file or prepare a second identical file on the input disk. I don't feel this is a bug, just a limitation imposed by DOS.

AVAILABILITY

I can supply a copy of the program and its accompanying routine on *your* clean, error-free, HARD SECTORED MITS FORMAT 8" disk. You must send the disk in a *reusable* mailing container. A chipboard box will be fine. Please enclose $8.00 to cover postage and handling in money order or cash. We cannot accept stamps, checks or any requests from any foreign countries with "black lists." You are free to distribute this program for any NON-COMMERCIAL use, but you may not modify it or use it for commercial use without expressed permission in writing from Neurological Services Inc. This distribution notice must accompany ALL copies made for distribution.

APPENDIX

This is a narrative description of some non-published entry points in MITS DOS. Not all of these entry points are described.

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 precede 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`

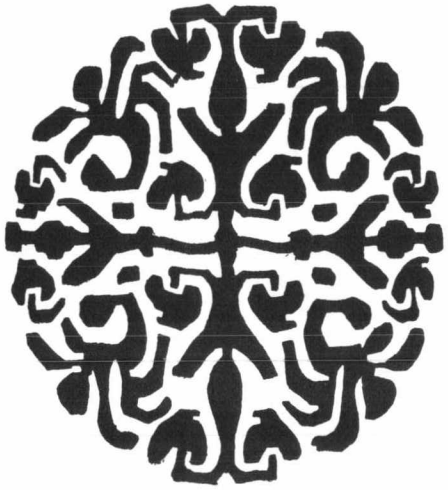
X'0F75'	ASK$$ – This subroutine is used to get the first or next parameter from a processor call. It must be followed by a DW which describes a string to be output in the event that the parameter is missing. On return, DE is pointing to the first byte of the parameter. If ASK$$ finds a C/R code (x'0D') it will output the message described at the DW and wait for a response to be typed in.
X'0FBE'	AANS$ – Converts ASCII characters to 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.

$CPY

```
000000                  000100 ; MAIN PROGRAM COPY
000000                  000110 ;
000000                  000120 ; PROGRAMMED BY A. L. BENDER, M. D.
000000                  000130 ;                WESTWOOD NEUROLOGICAL ASSOCIATES, P.A.
000000                  000140 ;
000000                  000150 ; DATE           JULY 5, 1979
000000                  000160 ;
000000                  000170 ; COPYRIGHT 1979 - NEUROLOGICAL SERVICES INC.
000000                  000180 ; 336 CENTER AVENUE
000000                  000190 ; WESTWOOD NJ 07675
000000                  000200 ;
000000                  000210 ; HOBBY USE AND DISTRIBUTION OF THIS PROGRAM IS PERMITTED
000000                  000220 ; WITHOUT RESTRICTION.  THERE IS NO RESTRICTION ON HOBBY
000000                  000230 ; COPIES - COMMERCIAL USE HOWEVER OR SALE OF THIS PROGRAM
000000                  000240 ; BY ANYONE BUT THE OWNER IS FORBIDDEN BY LAW.
000000                  000250 ;
000000                  000260          ENTRY   CPY
000000                  000270          EXT     ASK$$,IO,OPEN$,OUTEEE,OCTOCT,DECOUT
000000                  000280          EXT     BINCV$,CCELL,CCEL1,AANS$,ABORT,EXIT
000000                  000290          EXT     STACK$,MSG,TASKNM,CLOF$,ATOB,MOVE8
011441                  000300 ND$$$    EQU     X'1321' ; LAST DISK NUMBER
011442                  000310 NF$$$    EQU     X'1322' ; NUMBER OF FILES INITIALIZED
000000                  000320 ; THIS PROGRAM WILL ACCEPT A LIST OF FILES TO BE COPIED FROM ONE
000000                  000330 ; OR MORE DIFFERENT DISKS ONTO AN OUTPUT OR 'TARGET' DISK.  THE
000000                  000340 ; LIST OF FILES IS PREPARED WITH THE TEXT EDITOR AND
000000                  000350 ; THEREFORE IS EXPECTED TO HAVE A FILE NAME BEGINNING WITH AN
000000                  000360 ; AMPERSAND ('&').  THE AMPERSAND SHOULD NOT BE EXPLICITLY
                                                                          DESIGNATED
000000                  000370 ; ON THE PROCESSOR CALL IMAGE AS IT WILL BE INTERNALLY SUPPLIED.
000000                  000380 ;
000000                  000390 ; CALL IS: CPY FILENAME,DISK,OUTPUT
000000                  000400 ;
000000                  000410 ; FILENAME IS THE NAME OF THE FILE WHICH WILL CONTAIN THE
                                                                          FILENAMES
000000                  000420 ; DISK IS THE DISK NUMBER ON WHICH FILENAME RESIDES
000000                  000430 ; OUTPUT IS THE DISK NUMBER ON WHICH THE OUTPUT WILL BE WRITTEN
000000                  000440 ;
000000                  000450 ; SPECIAL CARE IS TAKEN SO AN OUTPUT FILE ISN'T AN INPUT FILE
000000                  000460 ; ON THE SAME UNIT.
000000                  000470 ; NO VERIFICATION IS PERFORMED AFTER THE COPY. - WE ASSUME
000000                  000480 ; SEE DOCUMENTATION FOR FURTHER DETAILS.          ITS OK.
000000                  000490 ;
000000 041 000000*      000500 CPY      LXI     SP,STACK$       ; HIGH MEMORY
000003 041 002075'      000510          LXI     H,CPYNAM        ; COPY NAME
000006 042 000000*      000520          SHLD    TASKNM
000011 072 011442       000530          LDA     NF$$$   ; NUMBER OF FILES INITIALIZED
000014 376 003          000540          CPI     3       ; NUMBER NEEDED FOR CPY
000016 372 000743'      000550          JM      CPERR7  ; EMODE - NOT ENOUGH FILES
000021 315 000000*      000560          CALL    MSG
000024 002077'          000570          DW      NAM     ; PUT OUT NAME
000026 315 000000*      000580 CPY1     CALL    ASK$$   ; GET FILE NAME FROM RESIDENT
000031 010246           000590          DW      MSG1    ; 'ENTER FILE NAME'
000033 041 002154'      000600          LXI     H,FILE3+1       ; FILE NAME TARGET
000036 315 000000*      000610          CALL    MOVE8   ; MOVE FILENAME
000041 315 000000*      000620 CPY2     CALL    ASK$$   ; GET DISK NUMBER
000044 010035           000630          DW      MSG2    ; 'ENTER DRIVE'
000046 315 000000*      000640          CALL    AANS$   ; CONVERT DRIVE TO BINARY
000051 000041'          000650          DW      CPY2    ; REPEAT IF ERROR
000053 062 002150'      000660          STA     DN3     ; SET DISK NUMBER FOR FILE3
000056 315 000000*      000670 CPY3     CALL    ASK$$   ; GET OUTPUT DISK NUMBER
000061 001713'          000680          DW      MSG3
000063 315 000000*      000690          CALL    AANS$   ; CONVERT IT TO BINARY
000066 000056'          000700          DW      CPY3    ; REPEAT IF N.G.
000070 062 002212'      000710          STA     DNO     ; SET UP OUTPUT DISK NUMBER
000073 107              000720          MOV     B,A
000074 072 011441       000730          LDA     ND$$$   ; GET NUMBER OF DISKS IN SYSTEM
000077 270              000740          CMP     B       ; COMPARE WITH DISK TO BE USED
000100 372 000613'      000750          JM      CPERR1  ; ERROR, DISK NUMBER NOT POSSIBLE
000103 315 000000*      000760          CALL    IO      ; TRY TO ASSIGN FILE3
000106 002143'          000770          DW      PKT3O   ; OPEN FILE FOR INPUT.
000110 072 002144'      000780          LDA     PKT3O+1 ; GET STATUS RESPONSE
000113 267              000790          ORA     A       ; SET FLAGS
000114 312 000142'      000800          JZ      CPY5    ; FILE EXISTS IN DIRECTORY
000117 315 000000*      000810          CALL    MSG     ; INDICATE SYSTEM FAILURE
000122 001156'          000820          DW      ERR2    ; 'ERROR CODE: '
000124 072 002144'      000830          LDA     PKT3O+1
000127 315 000000*      000840          CALL    OCTOCT  ; 'XXX'
000132 315 000000*      000850          CALL    MSG     ; ' TRYING TO OPEN FILELIST'
000135 001174'          000860          DW      ERR3    ;
000137 303 000000*      000870          JMP     ABORT   ; TOUGH ONE SWEETIE1
000142 076 012          000880 CPY5     MVI     A,10    ; SET DECIMAL CONVERSION
000144 062 000000*      000890          STA     CCELL   ; SET UP RESIDENT ROUTINES
000147 062 000000*      000900          STA     CCEL1
000152                  000910 ;
000152                  000920 ; START COPY - CHECK EACH FILE IN TURN
000152                  000930 ;
000152 257              000940 CPY6     XRA     A
000153 062 002263'      000950          STA     COUNT
000156 072 002212'      000960          LDA     DNO
000161 365              000970          PUSH    PSW
000162 315 001033'      000980          CALL    UNLOAD  ; ASK ABOUT UNLOADING CURRENT TARGET DISK
000165 361              000990          POP     PSW     ; GET FILE DEVICE
000166 315 001074'      001000          CALL    MOUNT   ; MOUNT CURRENT TARGET DISK
000171                  001010 ; PROCESS FILE NAME AND DISK OF FILE TO BE COPIED
000171 315 000770'      001020 CPY7     CALL    READIN  ; READ FIRST OR NEXT FILE & DISK
000174 041 002164'      001030          LXI     H,INFILE
000177 076 040          001040          MVI     A,' '
000201 006 010          001050          MVI     B,8     ; CLEAR FILE NAME AREA
000203 167              001060 CPY7A    MOV     M,A     ; BLANK OUT TARGET FILE NAME
000204 043              001070          INX     H
000205 005              001080          DCR     B
000206 302 000203'      001090          JNZ     CPY7A
000211 021 002264'      001100          LXI     D,BUF   ; GET START OF FILE
000214 006 006          001110          MVI     B,6     ; DATA LINE
000216 032              001120 CPY7AA   LDAX    D       ; GET LINE NUMBER
000217 346 177          001130          ANI     X'7F'   ; MASK OFF H.O. BIT
000221 315 000000*      001140          CALL    OUTEEE
000224 023              001150          INX     D       ; GET TO NEXT CHARACTER
000225 005              001160          DCR     B
000226 302 000216'      001170          JNZ     CPY7AA  ; PROCESS LINE NUMBER
000231 023              001180          INX     D       ; GET ACROSS TAB
000232 076 040          001190          MVI     A,' '   ; PUT OUT A BLANK
000234 315 000000*      001200          CALL    OUTEEE  ;
000237 041 002164'      001210          LXI     H,INFILE ; SET ADDR OF FILE NAME
000242 006 010          001215          MVI     B,8     ; LIMIT FILENAME TO 8 CHARACTERS **ALB**
000244 032              001220 CPY7B    LDAX    D       ; GET FILENAME FROM INBUF AND COPY TO
000245 376 054          001230          CPI     ','     ; LOOK FOR COMMA                        COMMA
000247 312 000274'      001240          JZ      CPY7C   ; FOUND, EXIT THIS CODE
000252 167              001250          MOV     M,A     ; MOVE FILE NAME CHARACTER
000253 043              001260          INX     H
000254 023              001270          INX     D       ; ADVANCE POINTERS
000255 315 000000*      001280          CALL    OUTEEE  ; DISPLAY FILE NAME ON CONSOLE
000260 005              001290          DCR     B       ; LOOP
000261 302 000244'      001300          JNZ     CPY7B   ; SHOULD NEVER FAIL THIS TEST
000264 315 000000*      001310          CALL    MSG     ; ERROR - FILE NAME>8 CHARACTERS
000267 001224'          001320          DW      ERR4
000271 303 000000*      001330          JMP     ABORT   ; REALLY CAN NEVER BE
000274 315 000000*      001340 CPY7C    CALL    OUTEEE  ; PUT OUT COMMA
000277 023              001350          INX     D       ; GET ACROSS COMMA
000300 325              001360          PUSH    D       ; SAVE ADDRESS IN BUFFER
000301 315 000000*      001370          CALL    ATOB    ; CONVERT DISK NUMBER TO BINARY
000304 062 002202'      001380          STA     DNI     ; INPUT DISK
000307 321              001390          POP     D
000310 032              001400 CPY7D    LDAX    D       ; PUT OUT DISK NUMBER
000311 376 060          001410          CPI     '0'     ; CHECK FOR NUMERIC
000313 372 000332'      001420          JM      CPY7E
000316 376 072          001430          CPI     '9'+1
000320 362 000332'      001440          JP      CPY7E
000323 315 000000*      001450          CALL    OUTEEE  ; PUT OUT CHARACTER
000326 023              001460          INX     D       ; ADVANCE TAKE INDEX
000327 303 000310'      001470          JMP     CPY7D   ; LOOP
000332 076 040          001480 CPY7E    MVI     A,' '   ; PUT OUT BLANK
000334 315 000000*      001490          CALL    OUTEEE
000337 107              001500          MOV     B,A     ; SET UP COMPARE
000340 072 002212'      001510          LDA     DNO     ; DESTINATION DISK NUMBER
000343 270              001520          CMF     B       ; ARE DISKS THE SAME?
000344 312 000623'      001530          JZ      CPERR2  ; YES, THATS NO GOOD FILE WILL BE DELETED
000347 076 201          001540          MVI     A,0201  ; TRY TO ASSIGN FILE FOR INPUT
000351 062 002210'      001550          STA     PKT2+3  ; MODIFY OPEN PACKET
000354 315 000000*      001560          CALL    IO      ; OPEN THRU SUPERVISOR
000357 002205'          001570          DW      PKT2    ; ADDRESS OF OPEN PACKET
000361 072 002206'      001580          LDA     PKT2+1  ; STATUS RESPONSE
000364 267              001590          ORA     A       ; SET FLAGS
000365 312 000633'      001600          JZ      CPERR3  ; ERROR, FILE ALREADY EXISTS ON OUTPUT
000370 376 022          001610          CPI     022     ; FILE NOT FOUND IN DIRECTORY       DISK
000372 302 000655'      001620          JNZ     CPERR4  ; SOME OTHER ERROR, REPORT IT
000375 076 202          001630          MVI     A,0202  ; MAKE OPEN PACKET OPEN FILE FOR OUTPUT
000377 062 002210'      001640          STA     PKT2+3  ;
000402 315 000000*      001650          CALL    IO      ; OPEN IT TO AVOID OPEN$ KICK OFF
000405 002175'          001660          DW      PKT1    ; FILE #1, INPUT FILE
000407 072 002176'      001670          LDA     PKT1+1
000412 267              001680          ORA     A       ; SENSE ERROR TERM
000413 302 000753'      001690          JNZ     CPERR8  ; REPORT ERROR BUT KEEP GOING
000416 315 000000*      001700          CALL    OPEN$   ; OPEN SECOND FILE
000421 002205'          001710          DW      PKT2    ; FILE #2, OUTPUT FILE
000423 041 000000       001720          LXI     H,0
000426 042 002241'      001730          SHLD    BLKCNT  ; CLEAR BLOCK COUNTER
000431                  001740 ; INITIATE FILE COPY
000431 315 000000*      001750 CPY8     CALL    IO      ; LOOP TO COPY FILE
000434 002215'          001760          DW      READ1   ; READ FILE
000436 072 002216'      001770          LDA     READ1+1 ; STATUS BITS
000441 267              001780          ORA     A
000442 312 000452'      001790          JZ      CPY8A   ; CONTINUE IF NO ERROR
000445 376 025          001800          CPI     025     ; SENSE EOF ON INPUT MEDIUM
```

```
000447 302 000721'      001810          JNZ     CPERR6  ; NON INPUT EOF ERROR
000452 052 002225'      001820 CPY8A    LHLD    RDCNT
000455 042 002235'      001830          SHLD    WRTCNT  ; SET NUMBER OF BYTES TO WRITE
000460 052 002241'      001840          LHLD    BLKCNT  ; INCREMENT SECTORS READ
000463 043              001850          INX     H
000464 042 002241'      001860          SHLD    BLKCNT
000467 315 000000*      001870          CALL    IO      ; WRITE
000472 002227'          001880          DW      WRITE2
000474 072 002230'      001890          LDA     WRITE2+1
000477 267              001900          ORA     A
000500 302 000677'      001910          JNZ     CPERR5  ; WRITE ERROR
000503 072 002216'      001920          LDA     READ1+1 ; GET STATUS BITS
000506 267              001930          ORA     A       ; SENSE NO ERRORS
000507 312 000431'      001940          JZ      CPY8    ; JUMP
000512 376 025          001950          CPI     025     ; SENSE EOF
000514 302 000431'      001960          JNZ     CPY8    ; NOT SO
000517                  001970 ; END FILE ON INPUT MEDIUM
000517 076 001          001980          MVI     A,1     ; CLOSE INPUT FILE
000521 315 000000*      001990          CALL    CLOF$
000524 076 002          002000          MVI     A,2     ; CLOSE OUTPUT FILE
000526 315 000000*      002010          CALL    CLOF$
000531 052 002241'      002020          LHLD    BLKCNT
000534 315 000000*      002030          CALL    BINCV$  ; TYPE OUT BLOCKS WRITTEN
000537 001737'          002040          DW      MSG5+1
000541 315 000000*      002050          CALL    MSG     ; PUT IT OUT
000544 001736'          002060          DW      MSG5
000546 041 002263'      002070          LXI     H,COUNT
000551 064              002080          INR     M       ; ADVANCE NUMBER OF FILES PROCESSED
000552 303 000171'      002090          JMP     CPY7    ; LOOP
000555                  002100 ; TERMINATE FILE COPY
000555 361              002110 CPY10    POP     PSW     ; POP RETURN OFF THE STACK
000556 076 003          002120          MVI     A,3     ; CLOSE FILELIST
000560 315 000000*      002130          CALL    CLOF$   ; THRU COOPERATIVE
000563 315 000000*      002140          CALL    MSG
000566 001765'          002150          DW      CARET   ; PUT OUT CRLF
000570 072 002263'      002160          LDA     COUNT   ; GET NUMBER OF FILES COPIED
000573 315 000000*      002170          CALL    DECOUT  ; PUT IT OUT ON CONSOLE
000576 315 000000*      002180          CALL    MSG     ;
000601 001767'          002190          DW      MSG6    ; ' FILES COPIED'
000603 315 000000*      002200          CALL    MSG     ; NOTE THAT FILE IS STILL MOUNTED
000606 002005'          002210          DW      MSG7
000610 303 000000*      002220          JMP     EXIT    ; EXIT TO USER
000613 315 000000*      002230 CPERR1   CALL    MSG     ; INDICATE IMPOSSIBLE DISK
000616 001120'          002240          DW      ERR1    ; 'DISK NUMBER NOT ON CURRENT SYSTEM'
000620 303 000056'      002250          JMP     CPY3    ; ASK AGAIN
000623 315 000000*      002260 CPERR2   CALL    MSG     ; PUT OUT I=O MSG
000626 001253'          002270          DW      ERR5
000630 303 000171'      002280          JMP     CPY7    ; CONTINUE COPY
000633 315 000000*      002290 CPERR3   CALL    MSG     ; PUT OUT OUTPUT DISK CONTAINS INPUT
000636 001325'          002300          DW      ERR6                                    FILE
000640 076 001          002310 ERRRET   MVI     A,1     ; CLOSE OPEN FILE
000642 315 000000*      002320          CALL    CLOF$
000645 076 002          002330          MVI     A,2     ; CLOSE BOTH FILES
000647 315 000000*      002340          CALL    CLOF$
000652 303 000171'      002350          JMP     CPY7
000655 365              002360 CPERR4   PUSH    PSW
000656 315 000000*      002370          CALL    MSG
000661 001156'          002380          DW      ERR2
000663 361              002390          POP     PSW
000664 315 000000*      002400          CALL    OCTOCT
000667 315 000000*      002410          CALL    MSG
000672 001431'          002420          DW      ERR7
000674 303 000640'      002430          JMP     ERRRET  ; CLOSE FILE, RETURN TO LOOP
000677 365              002440 CPERR5   PUSH    PSW
000700 315 000000*      002450          CALL    MSG     ; SAVE USER HAIR TEARING
000703 001156'          002460          DW      ERR2    ; ERROR CODE:
000705 361              002470          POP     PSW     ; GET STATUS BITS BACK
000706 315 000000*      002480          CALL    OCTOCT
000711 315 000000*      002490          CALL    MSG
000714 001473'          002500          DW      ERR8
000716 303 000000*      002510          JMP     ABORT   ; ZAP RUN
000721 365              002520 CPERR6   PUSH    PSW
000722 315 000000*      002530          CALL    MSG
000725 001156'          002540          DW      ERR2    ; 'ERROR CODE: '
000727 361              002550          POP     PSW
000730 315 000000*      002560          CALL    OCTOCT
000733 315 000000*      002570          CALL    MSG
000736 001517'          002580          DW      ERR9    ; 'READING'
000740 303 000000*      002590          JMP     ABORT   ; ZAP RUN
000743 315 000000*      002600 CPERR7   CALL    MSG     ; INSUFFICIENT FILES INITIALIZED
000746 001563'          002610          DW      ERR11   ;
000750 303 000000*      002620          JMP     ABORT   ; THATS THAT!
000753 315 000000*      002630 CPERR8   CALL    MSG     ; ''REQUESTED FILE NOT ON INPUT DISK'
000756 001622'          002640          DW      ERR12   ;
000760 076 001          002650          MVI     A,1
000762 315 000000*      002660          CALL    CLOF$   ; CLOSE FILE
000765 303 000171'      002670          JMP     CPY7    ; DO NEXT FILE
```

```
000770                  002680 ; READIN ROUTINE
000770                  002690 ; READS LIST OF FILES AND DISKS FROM FILE 3
000770 021 002264'      002700 READIN   LXI     D,BUF   ; SET ADDR OF LOGICAL LINE BUFFER
000773 315 000000*      002710 REED1    CALL    IO      ; GET ONE AT A TIME
000776 002257'          002720          DW      READ3   ; PACKET TO 103 READ
001000 072 002260'      002730          LDA     READ3+1 ; GET STATUS BITS
001003 267              002740          ORA     A
001004 312 001105'      002750          JZ      REED2   ; FILL BUFFER
001007 376 025          002760          CPI     025     ; IS IT EOF?
001011 312 000555'      002770          JZ      CPY10
001014 365              002780          PUSH    PSW
001015 315 000000*      002790          CALL    MSG     ; PUT OUT MSG
001020 001156'          002800          DW      ERR2    ; 'ERROR CODE:
001022 361              002810          POP     PSW
001023 315 000000*      002820          CALL    OCTOCT  ; PUT OUT OCTAL STUFF
001026 001542'          002830          DW      ERR10   ; 'READING FILELIST'
001030 303 000000*      002840          JMP     ABORT   ; DONE, FINISHED
001033                  002850 ; UNLOAD/LOAD DISK (PHYSICAL)
001033 062 002246'      002860 UNLOAD   STA     DSM+3   ; SAVE DISK TO UNLOAD
001036 315 000000*      002870          CALL    MSG     ;
001041 002043'          002880          DW      MSG8    ; 'IS FILE TO BE UNLOADED?'
001043 315 000000*      002890          CALL    IO      ; READ CHARACTER FROM CONSOLE
001046 002253'          002900          DW      CNSL
001050 072 002256'      002910          LDA     CNSL+3
001053 315 000000*      002920          CALL    OUTEEE  ; ECHO CHARACTER
001056 315 000000*      002930          CALL    MSG
001061 001765'          002940          DW      CARET   ; DO CR/LF
001063 376 131          002950          CPI     'Y'     ; WAS ANSWER YES?
001065 300              002960          RNZ             ; RETURN IF NOT
001066 315 000000*      002970          CALL    IO
001071 002243'          002980          DW      DSM     ; DISMOUNT DISK
001073 311              002990          RET             ; RETURN
001074 062 002252'      003000 MOUNT    STA     MNT+3   ; MOUNT DISK
001077 315 000000*      003010          CALL    IO
001102 002247'          003020          DW      MNT
001104 311              003030          RET
001105 072 002262'      003040 REED2    LDA     READ3+3 ; GET CHARACTER
001110 022              003050          STAX    D       ; PUT IT AWAY
001111 023              003060          INX     D       ; POINT TO NEXT LOC
001112 376 012          003070          CPI     012     ; WAS IT LF? (END OF LINE)
001114 310              003080          RZ              ; YES
001115 303 000773'      003090          JMP     REED1   ; READ NEXT
001120 104 111 123 113  003100 ERR1     DC      'DISK NUMBER FOR OUTPUT TOO BIG'
001124 040 116 125 115
001130 102 105 122 040
001134 106 117 122 040
001140 117 125 124 120
001144 125 124 040 124
001150 117 117 040 102
001154 111 307
001156 015 012          003110 ERR2     DB      015,012
001160 105 122 122 117  003120          DC      'ERROR CODE: '
001164 122 040 103 117
001170 104 105 072 240
001174 040 124 122 131  003130 ERR3     DC      ' TRYING TO OPEN FILELIST'
001200 111 116 107 040
001204 124 117 040 117
001210 120 105 116 040
001214 106 111 114 105
001220 114 111 123 324
001224 106 111 114 105  003140 ERR4     DC      'FILENAME > 8 CHARACTERS'
001230 116 101 115 105
001234 040 076 040 070
001240 040 103 110 101
001244 122 101 103 124
001250 105 122 323
001253 111 116 120 125  003150 ERR5     DC      'INPUT=OUTPUT DISK.  FILE WILL BE DESTROYED'
001257 124 075 117 125
001263 124 120 125 124
001267 040 104 111 123
001273 113 056 040 040
001277 106 111 114 105
001303 040 127 111 114
001307 114 040 102 105
001313 040 104 105 123
001317 124 122 117 131
001323 105 304
001325 015 012 106 111  003160 ERR6     DB      015,012,'FILE TO BE COPIED ALREADY EXISTS '
001331 114 105 040 124
001335 117 040 102 105
001341 040 103 117 120
001345 111 105 104 040
001351 101 114 122 105
001355 101 104 131 040
001361 105 130 111 123
001365 124 123 040
001370 117 116 040 117  003170          DB      'ON OUTPUT DISK, IGNORED COMMAND'
001374 125 124 120 125
```



```
001400 124 040 104 111
001404 123 113 054 040
001410 111 107 146 117
001414 122 105 104 040
001420 103 117 115 115
001424 101 116 104
001427 012 215          003180          DB      012,0215
001431 040 103 110 105 003190 ERR7      DC      ' CHECKING DIRECTORY OF OUTPUT DISK'
001435 103 113 111 116
001441 107 040 104 111
001445 122 105 103 124
001451 117 122 131 040
001455 117 106 040 117
001461 125 124 120 125
001465 124 040 104 111
001471 123 313
001473 040 127 122 111 003200 ERR8      DC      ' WRITING OUTPUT FILE'
001477 124 111 116 107
001503 040 117 125 124
001507 120 125 124 040
001513 106 111 114 305
001517 040 122 105 101 003210 ERR9      DC      ' READING INPUT FILE'
001523 104 111 116 107
001527 040 111 116 120
001533 125 124 040 106
001537 111 114 305
001542 040 122 105 101 003220 ERR10     DC      ' READING FILELIST'
001546 104 111 116 107
001552 040 106 111 114
001556 105 114 111 123
001562 324
001563 015 012          003230 ERR11    DB      015,012
001565 114 105 123 123 003240           DC      'LESS THAN 3 FILES INITIALIZED'
001571 040 124 110 101
001575 116 040 063 040
001601 106 111 114 105
001605 123 040 111 116
001611 111 124 111 101
001615 114 111 132 105
001621 304
001622 012 015 106 111 003250 ERR12     DB      012,015,'FILE REQUESTED TO BE COPIED WAS'
001626 114 105 040 122
001632 105 121 125 105
001636 123 124 105 104
001642 040 124 117 040
001646 102 105 040 103
001652 117 120 111 105
001656 104 040 127 101
001662 123
001663 040 116 117 124 003260           DC      ' NOT FOUND ON INPUT DISK'
001667 040 106 117 125
001673 116 104 040 117
001677 116 040 111 116
001703 120 125 124 040
001707 104 111 123 313
010246                  003270 MSG1     EQU     X'10A6' ; 'ENTER FILE NAME'
010035                  003280 MSG2     EQU     X'101D' ; 'ENTER DRIVE'
001713 105 116 124 105 003290 MSG3      DC      'ENTER OUTPUT DRIVE '
001717 122 040 117 125
001723 124 120 125 124
001727 040 104 122 111
001733 126 105 240
001736 040 060 060 060 003300 MSG5      DB      ' 000000 SECTORS WRITTEN'
001742 060 060 060 040
001746 123 105 103 124
001752 117 122 123 040
001756 127 122 111 124
001762 124 105 116
001765 012 215          003310 CARET    DB      012,0215
001767 040 106 111 114 003320 MSG6      DC      ' FILES COPIED.'
001773 105 123 040 103
001777 117 120 111 105
002003 104 256
002005 015 012          003330 MSG7     DB      015,012
002007 040 052 052 040 003340           DC      ' ** DISK REMAINS MOUNTED ** '
002013 104 111 123 113
002017 040 122 105 115
002023 101 111 116 123
002027 040 115 117 125
002033 116 124 105 104
002037 040 052 052 240
002043 015 012          003350 MSG8     DB      015,012
002045 111 123 040 104 003360           DC      'IS DISK TO BE UNLOADED? '
002051 111 123 113 040
002055 124 117 040 102
002061 105 040 125 116
```

```
002065 114 117 101 104
002071 105 104 077 240
002075 015 012              003370 CPYNAM   DB      015,012
002077 070 060 070 060      003380 NAM      DC      '8080 DISK FILE COPY VER 2.0 [8.2.79]'
002103 040 104 111 123
002107 113 040 106 111
002113 114 105 040 103
002117 117 120 131 040
002123 126 105 122 040
002127 062 056 060 040
002133 133 070 056 062
002137 056 067 071 335
002143 304 000 003 201      003390 PKT30    DB      0304,0,3,01+0200,6
002147 006
002150 000                  003400 DN3      DB      0       ; DISK NUMBER
002151 002153'              003410          DW      FILE3
002153 046 040 040 040      003420 FILE3    DB      '&',*       *
002157 040 040 040 040
002163 040
002164                      003430 INFILE   DS      8       ; I/O FILE NAME
002174 040                  003440          DB      ' '     ; STOPS MOVE ROUTINE
002175 304 000 001 201      003450 PKT1     DB      0304,0,1,0201,6 ; OPEN FILE# 1
002201 006
002202 000                  003460 DNI      DB      0       ; DISK NUMBER FOR INPUT
002203 002164'              003470          DW      INFILE  ;
002205 304 000 002 202      003480 PKT2     DB      0304,0,2,0202,6 ; OPEN FILE# 2
002211 006
002212 000                  003490 DNO      DB      0       ; DISK NUMBER FOR OUTPUT
002213 002164'              003500          DW      INFILE
002215 302 000 001 004      003510 READ1    DB      0302,0,1,4
002221 002264'              003520          DW      BUF     ; READ BUFFER ADDRESS
002223 000200               003530          DW      128     ; BYTE COUNT
002225 000000               003540 RDCNT    DW      0
002227 303 000 002 004      003550 WRITE2   DB      0303,0,2,4
002233 002264'              003560          DW      BUF
002235 000200               003570 WRTCNT   DW      128
002237 000000               003580          DW      0
002241 000000               003590 BLKCNT   DW      0       ; NUMBER OF BLOCKS COPIED THIS FILE
002243 241 000 006 000      003600 DSM      DB      0241,0,6,0
002247 240 000 006 000      003610 MNT      DB      0240,0,6,0
002253 002 000 101 000      003620 CNSL     DB      2,0,0101,0
002257 202 000 003 000      003630 READ3    DB      0202,0,3,0
002263 000                  003640 COUNT    DB      0       ; NUMBER OF FILES COPIED IN TOTO
002264                      003650 BUF      DS      128     ; MULTI-PURPOSE BUFFER
002464                      003660          END
```

* OUTEEE

```
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
```

* ATOB

```
000000                      000100          ENTRY   ATOB
000000 345                  000300 ATOB     PUSH    H
000001 353                  000310          XCHG            ; PUT DE IN HL
000002 257                  000600          XRA     A       ;CLEAR [A]
000003 127                  000700 AANS1    MOV     D,A     ;SAVE [A]
000004 176                  000800          MOV     A,M     ;GET BYTE OF ANS
000005 326 060              000900          SUI     '0'     ;IS IT NUMERIC
000007 332 000032'          001000          JC      AANSEX  ;NO - ALL DONE
000012 376 012              001100          CPI     10
000014 322 000032'          001200          JNC     AANSEX  ;NO - ALL DONE
000017 137                  001300          MOV     E,A     ;SAVE NEW DIGIT
000020 172                  001400          MOV     A,D     ;RESTORE [A]
000021 207                  001500          ADD     A       ;*2
000022 207                  001600          ADD     A       ;*4
000023 202                  001700          ADD     D       ;*5
000024 207                  001800          ADD     A       ;*10
000025 203                  001900          ADD     E       ;ADD DIGIT
000026 043                  002000          INX     H       ;ADV POINTER
000027 303 000003'          002100          JMP     AANS1   ;LOOP
000032 172                  002110 AANSEX   MOV     A,D     ;RESTORE [A]
000033 353                  002200          XCHG            ; RESTORE DE TO NEW ADDRESS
000034 341                  002210          POP     H       ; RESTORE HL
000035 311                  002400          RET
000036                      002500          END
```

* DECOUT

```
000000                      000100          ENTRY   DECOUT
000000                      000110          EXT     OUTEEE
000000 325                  000120 DECOUT   PUSH    D
000001 305                  000130          PUSH    B
000002 036 000              000140          MVI     E,0
000004 016 144              000150          MVI     C,100
000006 315 000026'          000160          CALL    +A
000011 016 012              000170          MVI     C,10
000013 315 000026'          000180          CALL    +A
000016 306 060              000190          ADI     '0'
000020 315 000000*          000200          CALL    OUTEEE
000023 301                  000210          POP     B
000024 321                  000220          POP     D
000025 311                  000230          RET
000026 006 057              000240 .A       MVI     B,'0'-1
000030 004                  000250 .B       INR     B
000031 221                  000260          SUB     C
000032 322 000030'          000270          JNC     -B
000035 201                  000280          ADD     C
000036 127                  000290          MOV     D,A
000037 170                  000300          MOV     A,B
000040 376 060              000310          CPI     '0'
000042 302 000051'          000320          JNZ     +C
000045 173                  000330          MOV     A,E
000046 267                  000340          ORA     A
000047 172                  000350          MOV     A,D
000050 310                  000360          RZ
000051 034                  000370 .C       INR     E
000052 170                  000380          MOV     A,B
000053 315 000000*          000390          CALL    OUTEEE
000056 172                  000400          MOV     A,D
000057 311                  000410          RET
000060                      000420          END
```

*OCTOCT

```
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
```

*OPEN$

```
000000                      000100          ENTRY   OPEN$
000000                      000105          ENTRY   CLOF$
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
```

ADDENDUM

Dear *DDJ*:

I was reveiwing the CPM printer driver which I sent you and which you have indicated might be used in your August issue, and have found one potential problem for some users. I would advise that the following be used as an addendum to the program DIABLO . PRN.

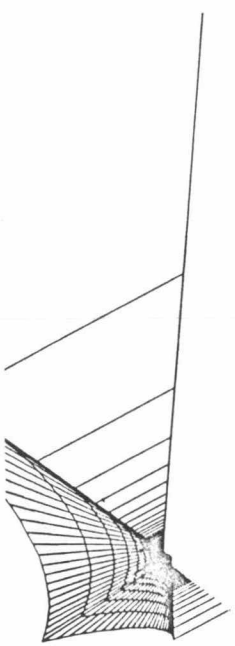
The program DIABLO contains one potential problem which is evidenced when the DIABLO keyboard is used as the console input device. If the console device is switched back to the primary device while a CONIN input is being made, a garbage character is read, since the status port is changed but the data port does not change until the routine is re-entered. To correct this problem, the assembly listing should be changed to show a jump to CONIN when the status shows not ready instead of looping to CON3. The correct listing should be:

```
020C   CA9BD4   JZ     CONIN + BASE   L   Loop till ready
```

In the article, I mentioned using Control P to turn a printer on and off as CDOS does. I have just finished a modification to my CPM CBIOS which adds this and several other features. I will pass this along at my next opportunity.

Yours truly,

Steve



```
000000                  000250 ;
000000 343              000260 OPEN$   XTHL            ; GET RETURN ADDRESS
000001 315 000000*      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
000135 311              000570         RET
000136 105 000          000580 CLOSE   DB      0105,0
000140 000              000590 CLOSER  DB      0
000141                  000600         END
```

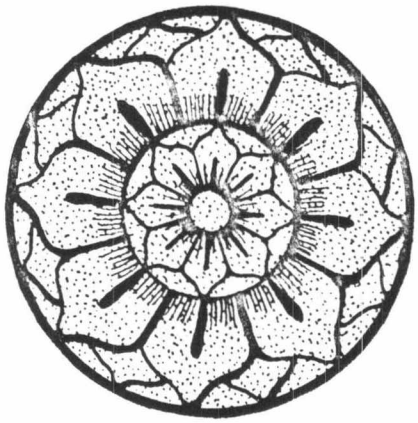
*SYSENT

```
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
```

* LASTP (MUST BE COLLECTED LAST)

```
000000                  000100         ENTRY   STACK$,LASTD$
000000                  000110 LASTD$  EQU     *
000000                  000120         DS      512
001000                  000130 STACK$  EQU     *
001000                  000140         END
```

an 8080 to Z80 translator system



BY ROBERT W. DEA
198 Lloyd Ave.
Fremont, Ca 94536

This program will accept 8080 mnemonics as input and translate them to the corresponding Z80 mnemonics; it was written in Digital Group Business Basic. I currently own a Digital Group Z80 computer system with the 50K bytes of memory, 4 digital PHI decks, and a 30 cps printer.

The 8080 to Z80 translator consists of a language skeleton and a grammar analyzer. The language skeleton in this case defines the structure of Intel 8080 mnemonics and the modified Digital Group Z80 mnemonics. The grammar analyzer parses the language skeleton in a top down left right process and verifies the correctness of the input mnemonic and generates the corresponding Z80 output. The translator is totally driven by the language skeleton and can be easily changed to translate a subset of the Z80 mnemonics to 8080 mnemonics. In fact it can be modified to perform any translation which consists of a one-to-one transformation.

This version can edit 8080 mnemonics and store them into a tape file on my Digital Group system. It will read the edited 8080 file generated and translate the mnemonics to the modified Digital Group Z80 mnemonics and store them into another tape file for later input to my Z80 assembler. The difference between the Digital Group Z80 mnemonics and the Zilog mnemonics are those instructions that deal with the accumulator. A few simple changes to the skeleton will allow the generation of Zilog Z80 mnemonics.

The skeleton definition has the following format.

n,8080mn,Z80mn,sub1,sub2, . . .

n	=	The number of subfields following
8080mn	=	The 8080 instruction
Z80mn	=	The Z80 equivalent instruction
subn	=	Subfield definitions
		<A> – regular one to one register translation
		<G> – RST number translation
		<B> – register pair translation
		<C> – register pair to accumulator
		<D> – accumulator to register pair
		<E> – edit a comma
		<F> – edit following character string to output
		<S> – one to one character string translation

The data scanner ends its scan upon detecting a 999 in the (n) field. By removing these DATA statements and deleting the speed enhancement section along with the comments should reduce storage requirements by 30-40 % but increases translation time by 100-150%.

```
TYPE (E) TO EDIT A 8080 SOURCE FILE
TYPE (T) TO TANSLATE 8080 TO Z80
>E
8080 EDIT PROGRAM
OUTPUT FILE ALWAYS ON DRIVE #1
TYPE 8080 SOURCE IN FOLLOWING FORMAT
LABEL OPERATION OPERAND COMMENT
TYPE '*EXIT' TO EXIT TO MENU
INPUT 8080 OUTPUT FILE NAME? TEST
OUTPUT FILE TEST WILL BE ON DRIVE #1
?  MOV B,B
?  MOV B,M
?  MOV M,C
?  HLT
?  MVI B,1
?  MVI M,1
?  LXI B,377
?  LXI SP,377
?  STAX B
?  LDAX B
?  SHLD 277
?  LHLD 277
?  STA 277
?  LDA 377
?  XCHG
?  SPHL
?  ANA B
?  ANA M
?  XRA B
?  XRA M
?  ORA B
?  ORA M
?  CMP B
?  RLC
?  RRC
?  RAL
?  RAR
?  CMA
?  STC
?  CMC
?  ANI 1
?  XRI 25
?  ORI 17
?  CPI 7
?  JNZ LABEL
?  JZ LABEL
?  JNC LABEL
?  JC LABEL
?  JPO LABEL
?  JP LABEL
?  JM LABEL
?  CNZ LABEL
?  CZ LABEL
```

run

```
TYPE (E) TO EDIT A 8080 SOURCE FILE
TYPE (T) TO TRANSLATE 8080 TO Z80
>T
INPUT 8080 FILE NAME? TEST
INPUT DRIVE#? 1
OUTPUT Z80 FILE NAME? TEST2
OUTPUT DRIVE#? 2
 0010  MOV B,B                LD   B,B
 0020  MOV B,M                LD   B,(HL)
 0030  MOV M,C                LD   (HL),C
 0040  HLT                    HALT
 0050  MVI B,1                LD   B,1
 0060  MVI M,1                LD   (HL),1
 0070  LXI B,377              LD   BC,377
 0080  LXI SP,377             LD   SP,377
 0090  STAX B                 LD   (BC),A
 0100  LDAX B                 LD   A,(BC)
 0110  SHLD 277               LD   (277),HL
 0120  LHLD 277               LD   HL,(277)
 0130  STA 277                LD   (277),A
 0140  LDA 377                LD   A,(377)
 0150  XCHG                   EX   DE,HL
 0160  SPHL                   LD   SP,HL
 0170  ANA B                  AND  B
 0180  ANA M                  AND  (HL)
 0190  XRA B                  XOR  B
 0200  XRA M                  XOR  (HL)
 0210  ORA B                  OR   B
 0220  ORA M                  OR   (HL)
 0230  CMP B                  CP   B
 0240  RLC                    RLCA
 0250  RRC                    RRCA
 0260  RAL                    RLA
 0270  RAR                    RRA
 0280  CMA                    CPL
 0290  STC                    SCF
 0300  CMC                    CCF
 0310  ANI 1                  AND  1
 0320  XRI 25                 XOR  25
 0330  ORI 17                 OR   17
 0340  CPI 7                  CP   7
 0350  JNZ LABEL              JP   NZ,LABEL
 0360  JZ LABEL               JP   Z,LABEL
 0370  JNC LABEL              JP   NC,LABEL
 0380  JC LABEL               JP   C,LABEL
 0390  JPO LABEL              JP   PO,LABEL
 0400  JP LABEL               JP   P,LABEL
 0410  JM LABEL               JP   M,LABEL
 0420  CNZ LABEL              CALL NZ,LABEL
 0430  CZ LABEL               CALL Z,LABEL
 0440  CNC LABEL              CALL NC,LABEL
 0450  CC LABEL               CALL C,LABEL
 0460  CPO LABEL              CALL PO,LABEL
 0470  CPE LABEL              CALL PE,LABEL
 0480  CP LABEL               CALL P,LABEL
 0490  CM LABEL               CALL M,LABEL
 0500  RNZ                    RET  NZ
 0510  RZ                     RET  Z
 0520  RNC                    RET  NC
 0530  RC                     RET  C
 0540  RPO                    RET  PO
 0550  RPE                    RET  PE
 0560  RP
```

listing

```
 10   REM
 20  REM         8080 TO Z80 TRANSLATOR
 30   REM
 40  REM  WRITTEN BY ROBERT W. DEA
 50  REM         COPYRIGHT 1978
 60  PRINT "TYPE (E) TO EDIT A 8080 SOURCE FILE"
 70  PRINT "TYPE (T) TO TRANSLATE 8080 TO Z80"
 80  INPUT ">",Q$
 90  IF Q$="E" THEN 4550
100   IF Q$<>"T" THEN 60
110    DATA  1,"ACI","ADC","<S>"
120    DATA  1,"ADC","ADC","<S>"
130    DATA  1,"ADD","ADD","<S>"
140    DATA  1,"ADI","ADD","<S>"
150    DATA  1,"ANA","AND","<A>"
160    DATA  1,"ANI","AND","<S>"
170    DATA  999,"X","Y"
180    DATA  1,"CALL","CALL","<S>"
190    DATA  3,"CC","CALL","<F>","C,","<S>"
200    DATA  3,"CM","CALL","<F>","M,","<S>"
210    DATA  0,"CMA","CPL"
220    DATA  0,"CMC","CCF"
230    DATA  1,"CMP","CP","<A>"
240    DATA  3,"CNC","CALL","<F>","NC,","<S>"
250   DATA 3,"CNZ","CALL","<F>","NZ,","<S>"
260    DATA  999,"X","Y"
270    DATA  3,"CP","CALL","<F>","P,","<S>"
280    DATA  3,"CPE","CALL","<F>","PE,","<S>"
290    DATA  1,"CPI","CP","<S>"
300    DATA  3,"CPO","CALL","<F>","PO,","<S>"
310    DATA  3,"CZ","CALL","<F>","Z,","<S>"
320    DATA  999,"X","Y"
330    DATA  0,"DAA","DAA"
340   DATA 3,"DAD","ADD","<F>","HL,","<B>"
350    DATA  1,"DCR","DEC","<A>"
360    DATA  1,"DCX","DEC","<B>"
370    DATA  0,"DI","DI"
380    DATA  999,"X","Y"
390   DATA 0,"EI","EI"
400    DATA  0,"HLT","HALT"
410    DATA  999,"X","Y"
420    DATA  5,"IN","IN","<F>","A,(","<S>","<F>",")"
430    DATA  1,"INR","INC","<A>"
440    DATA  1,"INX","INC","<B>"
450    DATA  999,"X","Y"
460    DATA  3,"JC","JP","<F>","C,","<S>"
470    DATA  3,"JM","JP","<F>","M,","<S>"
480    DATA  1,"JMP","JP","<S>"
490    DATA  3,"JNC","JP","<F>","NC,","<S>"
500    DATA  3,"JNZ","JP","<F>","NZ,","<S>"
510    DATA  999,"X","Y"
520    DATA  3,"JP","JP","<F>","P,","<S>"
530    DATA  3,"JPE","JP","<F>","PE,","<S>"
540    DATA  3,"JPO","JP","<F>","PO,","<S>"
550    DATA  3,"JZ","JP","<F>","Z,","<S>"
560    DATA  999,"X","Y"
570    DATA  1,"LDAX","LD","<D>"
580    DATA  5,"LDA","LD","<F>","A,(","<S>","<F>",")"
```

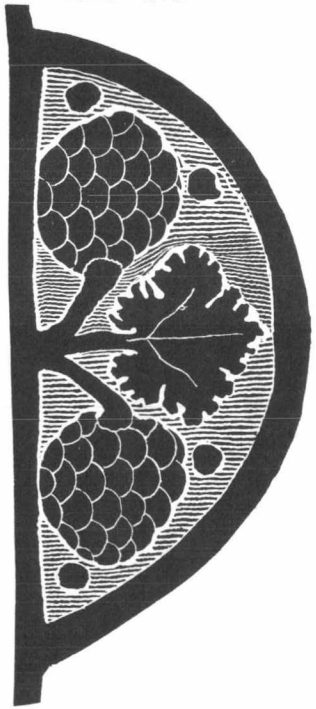
```
590   DATA  5,"LHLD","LD","<F>","HL,(","<S>","<F>",")"
600   DATA  3,"LXI","LD","<B>","<E>","<S>"
610   DATA  999,"X","Y"
620   DATA  3,"MOV","LD","<A>","<E>","<A>"
630   DATA  3,"MVI","LD","<A>","<E>","<S>"
640   DATA  0,"NOP","NOP"
650   DATA  999,"X","Y"
660   DATA  1,"ORA","OR","<A>"
670   DATA  1,"ORI","OR","<S>"
680   DATA  5,"OUT","OUT","<F>","(","<S>","<F>",")",A"
690   DATA  999,"X","Y"
700   DATA  0,"PCHL","JP    (HL)"
710   DATA  1,"POP","POP","<B>"
720   DATA  1,"PUSH","PUSH","<B>"
730   DATA  999,"X","Y"
740   DATA  0,"RAL","RLA"
750   DATA  0,"RAR","RRA"
760   DATA  0,"RC","RET    C"
770   DATA  0,"RET","RET"
780   DATA  0,"RLC","RLCA"
790   DATA  0,"RM","RET    M"
800   DATA  0,"RNC","RET    NC"
810   DATA  0,"RNZ","RET    NZ"
820   DATA  999,"X","Y"
830   DATA  0,"RP","RET    P"
840   DATA  0,"RPE","RET    PE"
850   DATA  0,"RPO","RET    PO"
860   DATA  0,"RRC","RRCA"
870   DATA  1,"RST","RST","<G>"
880   DATA  0,"RZ","RET    Z"
890   DATA  999,"X","Y"
900   DATA  3,"SBB","SBC","<F>","A,","<A>"
910   DATA  3,"SBI","SBC","<F>","A,","<S>"
920   DATA  5,"SHLD","LD","<F>","(","<S>","<F>",",),HL"
930   DATA  0,"SPHL","LD    SP,HL"
940   DATA  999,"X","Y"
950   DATA  1,"STAX","LD","<C>"
960   DATA  5,"STA","LD","<F>","(","<S>","<F>",",),A"
970   DATA  0,"STC","SCF"
980   DATA  1,"SUB","SUB","<A>"
990   DATA  1,"SUI","SUB","<S>"
1000   DATA  999,"X","Y"
1010   DATA  0,"XCHG","EX    DE,HL"
1020   DATA  1,"XRA","XOR","<A>"
1030   DATA  1,"XRI","XOR","<S>"
1040   DATA  2,"XTHL","EX","<F>","(SP),HL"
1050   DATA  999,"X","Y"
1060   DIM  A$(40),B$(30),C$(20),D$(18),E$(4),I$(80)
1070   DIM  O$(80),N$(80),G$(80)
1080   DIM  Z8$(20),H$(80)
1090   DIM  C1$(10)
1100  REM TAPE BUFFERS FOR I/O
1110  DIM R1$(256),O1$(256),I9$(80),O9$(80)
1120  DIM R$(256)
1130  REM INITIALIZE SYNTAX TREES
1140   A$="A A B B C C D D E E H H L L M (HL)"
1150   B$="PSW AF B BC D DE H HL SP SP"
1160   C$="B (BC),A D (DE),A"
1170   D$="B A,(BC) D A,(DE)"
1180   E$=" , ,"
1190   G$="0 0 1 8 2 10H 3 18H 4 20H 5 28H 6 30H 7 38H"
1200  INPUT "INPUT 8080 FILE NAME? ",I9$
1210  INPUT "INPUT DRIVE#? ",D
1220  I9$=I9$+CHR$(0)
1230  OPEN (0,E,I9$,2,D)
1240  IF E>1 THEN STOP
1250  INPUT "OUTPUT Z80 FILE NAME? ",O9$
1260  INPUT "OUTPUT DRIVE#? ",D
1270  OPEN (1,E,O9$,1,D)
1280  IF E>1 THEN STOP
1290  R=0 : F1=1 : W=1
1300  W=0 : C9=0
1310  REM GET 8080 INSTRUCTION
1320  REM GENERATE AUTO LINE NUMBERING INC BY 10
1330  REM SET TO GENERATE 5 PLACE VALUE
1340   C1=90000
1350  REM INC LINE COUNT BY 10
1360   C1=C1+10
1370  REM CONVERT TO ASCII NUMBER 5 PLACES
1380   C1$=STR$(C1)
1390   X1=LEN(C1$)
1400   C1$=" "+C1$(3,X1)
1410   #C1$ ;
1420  REM SOLICIT NEXT 8080 INSTRUCTION
1430  IF F1>C9 THEN 4370
1440  R1$=""
1450  Z1$=CHR$(0)
1460  IF R$(F1,F1)=Z1$(1,1) THEN 1500
1470  R1$=R1$+R$(F1,F1)
1480  F1=F1+1
1490  GOTO 1450
1500  F1=F1+1
1510  I$=R1$
1520  #I$
1530  REM WE HAVE END OF TRANSLATION?
1540  IF I$="*EXIT" THEN 4440
1550  REM ANY INPUT?
1560   IF  LEN(I$)=0  THEN  1360
1570  REM GO EDIT LINE# IN AND CONVERT TO Z80 INSTRUCTION
1580   GOSUB  3610
1590  REM DO WE HAVE AN 8080 INSTRUCTION IN INSTRUCTION BUFFER?
1600   IF  LEN(N$)=0  THEN  2220
1610  REM YES! NOW GO CALCULATE WHERE TO START CHECKING THE DATA
1620  REM ARRAY.
1630  REM NOW GOSUB TO SECTION THAT WILL SPEED UP THE 8080 TO Z80
1640  REM INSTRUCTION MATCH.
1650   GOSUB  3870
1660  REM GET NEXT DATA ELEMENT
1670   READ  L,Z8$
1680  REM INITIALIZE OUTPUT STRING BUFFER
1690   O$=""
1700   GOSUB  3610
1710  REM END OF DATA TABLE?
1720   IF  L=999  THEN  2090
1730  REM THE 8080 INSTRUCTION MATCH TABLE ELEMENT?
1740    IF  Z8$<>N$  THEN  2160
```

```
1750  REM YES! NOW GET Z80 OUTPUT INSTR AND EDIT TO OUTPUT
1760  REM BUFFERS.
1770   READ  X$
1780   O$=O$+"      "
1790   O$=O$+X$+"  "
1800  REM NOW READ TRANSLATION SUBFIELDS
1810   FOR  I=1  TO  L
1820      READ  S$
1830    REM CHECK SUBSET FOR DEFINATION MATCH AND TRANSFER TO
1840    REM DEFINATION PROCESS SECTION.
1850       IF  S$="<A>"  THEN  GOSUB  2320
1860     IF  S$="<G>"  THEN  GOSUB  2620
1870       IF  S$="<B>"  THEN  GOSUB  2680
1880      IF  S$="<C>"  THEN  GOSUB  2740
1890      IF  S$="<D>"  THEN  GOSUB  2800
1900      IF  S$="<E>"  THEN  GOSUB  2860
1910      IF  S$="<F>"  THEN  GOSUB  2910
1920     IF  S$="<S>"  THEN  GOSUB  3000
1930    REM IS MATCH FLAG SET?
1940      IF  F=1  THEN  2020
1950     REM  WE GET HERE  PARSE FAILED
1960    REM FLUSH OUT REST OF DATA LINE
1970     FOR  J=I+1  TO  L
1980          READ  X$
1990        NEXT  J
2000      EXIT  1670
2010    REM YES WE HAVE A GOOD TRANSLATION
2020     NEXT  I
2030   REM EDIT ANYTHING  THATS LEFT
2040   IF  T+1 > LEN(I$)  THEN  2070
2050   O$=O$+I$(T+1)
2060  REM RESET DATA POINTER TO TOP
2070   RESTORE
2080   GOTO  2230
2090  REM NO FIND EXIT
2100   O$="*NO MATCH* "+I$
2110  REM RESET DATA POINTER
2120   RESTORE
2130  REM OUTPUT *NO MATCH* MSG AND DEC LINE COUNTER BY 10
2140   C1=C1-10
2150   GOTO  2230
2160  REM SECTION TO CLEAN UP DATA POINTERS
2170   FOR  J=1  TO  L+1
2180       READ  X$
2190     NEXT  J
2200   GOTO  1670
2210  REM EDIT LINE# AND OUTPUT THE TRANSLATED Z80 INST
2220   O$=C1$+" "+O$
2230   #O$
2240  REM INITIALIZE OUTPUT BUFFER AND GO GET NEXT 8080 INPUT.
2250   O$=""
2260  O$=O$+CHR$(0)
2270  L9=LEN(O$)+LEN(O1$)
2280  IF L9>256 THEN 4490
2290  O1$=O1$+O$
2300   GOTO  1360
2310  REM PROCESS SINGLE REG SECTION "<A>"
2320   GOSUB  3070
2330  REM INITIALIZE PROCESS SECTION AND SET TO <A> TREE
2340   H$=A$
2350  REM INITIALIZE TREE SEARCH SECTION
2360   H2=0
2370   T2=1
2380  REM CHECK IF WE PARSED THE WHOLE INPUT 8080 INST YET.
2390  REM SEE IF IT SATIFIES THE 8080 DEFINATION
2400   IF  T2>LEN(H$)  THEN  2580
2410  REM GO SCAN THE DEFINATION TREE FROM LEFT TO RIGHT
2420  REM GO GET NEXT SUB DEFINATION
2430   GOSUB  3420
2440  REM DOES IT MATCH THE SUB ELEMENT WE HAVE?
2450   IF  M$=N$  THEN  2520
2460  REM NO IT DOES NOT . SO SKIP THE NEXT Z80 SUB TRANSLATION
2470  REM AND GO TRY FOR ANOTHER 8080 DUB DEFINATION MATCH.
2480   GOSUB  3420
2490   GOTO  2400
2500  REM WE HAVE A 8080 SUB DEFINATION MATCH. NOW GET THE
2510  REM Z80 TRANSLATION AND EDIT IT INTO THE OUTPUT BUFFER O$.
2520   GOSUB  3420
2530   O$=O$+M$
2540  REM SET MATCH FLAG
2550   F=1
2560   RETURN
2570  REM SET NO MATCH FLAG
2580   F=0
2590   RETURN
2600  REM PROCESS "<G>" DEFINATIONS
2610  REM GET NEXT INPUT SUB FIELD
2620   GOSUB  3070
2630  REM SET TO <G> DEFINATION TREE
2640   H$=G$
2650   GOTO  2360
2660  REM PROCESS "<B>" DEFINATIONS
2670  REM GET NEXT INPUT SUB FIELD
2680   GOSUB  3070
2690  REM SET TO <B> DEFINATION TREE
2700   H$=B$
2710   GOTO  2360
2720  REM PROCESS "<C>" DEFINATIONS
2730  REM GET NEXT INPUT SUB FIELD
2740   GOSUB  3070
2750  REM SET TO <C> DEFINATION TREE
2760   H$=C$
2770   GOTO  2360
2780  REM PROCESS "<D>" DEFINATION
2790  REM GET NEXT INPUT SUB FIELD
2800   GOSUB  3070
2810  REM SET TO <D> DEFINATION TREE
2820   H$=D$
2830   GOTO  2360
2840  REM PROCESS "<E>" DEFINATION
2850  REM GET NEXT INPUT SUB FIELD
2860   GOSUB  3070
2870  REM SSET TO <E> DEFINATION TREE
2880   H$=E$
2890   GOTO  2360
2900  REM PROCESS "<F>" DEFINATIONS
2910   READ  X$
2920  REM JUST EDIT NEXT Z80 TRANSLATION ELEMENT TO OUTPUT BUFFER
2930   O$=O$+X$
2940  REM SET MATCH FLAG
2950   F=1
2960   I=I+1
2970   RETURN
2980  REM PROCESS "<S>" DEFINATIONS
2990  REM GET NEXT INPUT SUB FIELD AND EDIT INTO OUTPUT BUFFER
3000   GOSUB  3070
```

```
3010  F=1
3020  IF  LEN(N$)=0  THEN  F=0
3030  O$=O$+N$
3040  RETURN
3050 REM SCANNER RETURNS N$
3060 REM INITIALIZE SUB FIELD SCANNER SECTION
3070  H=T+1
3080  N$=""
3090  H1=H
3100 REM SCAN FOR NEXT 8080 INST SUB FIELD STARTING WITH FIRST
3110 REM NON-BLANK.
3120  FOR  J=H1  TO  LEN(I$)
3130   REM GET NEXT CHAR
3140   REM IS IT A BLANK?
3150    IF  I$(J,J) <> " "  THEN  EXIT  3210
3160   REM YES! GO GET NEXT CHAR
3170    H=H+1
3180    T=T+1
3190    NEXT  J
3200 REM WE SHOULD BE POINTING TO START OF 8080 SUB FIELD
3210  H1=H
3220 REM SCAN UNTIL WE GET A TERMINATOR OR END OF LINE.
3230 REM ("BLANK" AND "," ARE TERMINATORS)
3240  FOR  J=H1  TO  LEN(I$)
3250   REM GET NEXT CHAR
3260   REM IS IT A BLANK?
3270    IF  I$(J,J)=" "  THEN  EXIT  3390
3280   REM IS IT A ","
3290    IF  I$(J,J)=","  THEN  EXIT  3370
3300   REM NO! ITS NONE OF THE ABOVE SO EDIT THIS CHAR INTO
3310   REM INSTRUCTION BUFFER.
3320    N$=N$+I$(J,J)
3330    T=T+1
3340   REM GO GET NEXT CHAR
3350    NEXT  J
3360  RETURN
3370  IF  LEN(N$)=0  THEN  T=T+1
3380  IF  LEN(N$)=0  THEN  N$=","
3390  REM
3400  RETURN
3410 REM SCAN THE TREES
3420  H2=T2
3430  M$=""
3440  H3=H2
3450  FOR  J=H3  TO  LEN(H$)
3460   REM TAB TO NEXT NON-BLANK CHARACTER
3470    IF  H$(J,J)<>" "  THEN  EXIT  3510
3480    H2=H2+1
3490    T2=T2+1
3500    NEXT  J
3510  H3=H2
3520 REM GET NEXT SUB DEFINATION OF 8080 INSTRUCTION
3530  FOR  J=H3  TO  LEN(H$)
3540   REM WE HAVE A BLANK YET?
3550    IF  H$(J,J)= " "  THEN  EXIT  3600
3560   REM NO! THEN ADD THIS CHAR
3570    M$=M$+H$(J,J)
3580    T2=T2+1
3590    NEXT  J
3600  RETURN
3610 REM SECTION TO EDIT A LABEL AND OUTPUT Z80 TRANSLATION
3620  H=0
3630  T=1
3640 REM WE HAVE A CHAR IN COL#1?
3650  IF  I$(1,1)=" "  THEN  3770
3660  H=1
3670  FOR  K=1  TO  LEN(I$)
3680   REM GET NEXT CHAR
3690   REM IS IT A BLANK
3700    IF  I$(K,K)=" "  THEN  EXIT  3770
3710   REM NO! NOT A BLANK THEN MUST BE PART OF THE LABEL FIELD.
3720   REM SO EDIT IT INTO OUTPUT BUFFER
3730    O$=O$+I$(K,K)
3740    T=K+1
3750    NEXT  K
3760 REM TAB TO COL 7 ON OUTPUT BUFFER
3770  FOR  K=LEN(O$)+1  TO  7
3780    O$=O$+" "
3790    NEXT  K
3800 REM NOW SCAN FOR THE 8080 INSTRUCTION
3810  GOSUB  3070
3820 REM YES THEN WE SCAN FOR FIRST BLANK
3830  RETURN
3840 REM THIS IS A SPEED ENHANCEMENT SECTION
3850 REM CHECK FIRST CHAR OF 8080 INPUT AND SET DATA POINTER
3860 REM TO POINT TO ENTRY INTO THE TRANSLATION TABLE.
3870  N1=ASC(N$)
3880  IF  LEN(N$)<2  THEN  4350
3890 REM GET FIRST CHAR OF INPUT INST AND CONVERT TO NUMERIC
3900  N2=ASC(N$(2))
3910  IF  N1>= 104  THEN  N1=N1-39
3920  IF  N2>= 104  THEN  N2=N2-39
3930  N1=N1-64
3940  N2=N2-64
3950  IF  N1>26  THEN  4350
3960  IF  N1<1  THEN  4350
3970  IF  N1>18  THEN  4280
3980  IF  N1>9  THEN  4110
3990  ON  N1  GOTO  4000,4350,4020,4050,4070,4350,4350,4070,4090
4000  RESTORE  110
4010  RETURN
4020  RESTORE  180
4030  IF  N2>14  THEN  RESTORE  270
4040  RETURN
4050  RESTORE  330
4060  RETURN
4070  RESTORE  390
4080  RETURN
4090  RESTORE  420
4100  RETURN
4110  N1=N1-9
4120  ON  N1  GOTO  4130,4350,4160,4180,4180,4200,4220,4350,4240
4130  RESTORE  460
4140  IF  N2>14  THEN  RESTORE  520
4150  RETURN
4160  RESTORE  570
4170  RETURN
4180  RESTORE  620
4190  RETURN
4200  RESTORE  660
4210  RETURN
4220  RESTORE  700
4230  RETURN
4240  RESTORE  740
4250  IF  N2>14  THEN  RESTORE  830
4260  RETURN
```

```
4270   RETURN
4280   N1=N1-18
4290   ON  N1  GOTO  4300,4350,4350,4350,4350,4330,4350,4350
4300   RESTORE  900
4310   IF  N2>16  THEN  RESTORE  950
4320   RETURN
4330   RESTORE  1010
4340   RETURN
4350   RESTORE  1050
4360   RETURN
4370  R$=""
4380  GET (0,E,R$,R)
4390  IF E>1 THEN 4440
4400  F1=1
4410  R=R+1
4420  C9=LEN(R$)
4430  GOTO 1430
4440  IF LEN(O1$)<0 THEN 4460
4450  PUT (1,E,O$,W)
4460  CLOSE (0,E)
4470  CLOSE (1,E)
4480  GOTO 10
4490  PUT (1,E,O1$,W)
4500  W=W+1
4510  IF E>1 THEN 3240
4520  O1$=""
4530  GOTO 2290
4540  REM SECTION TO EDIT A 8080 SOURCE FILE
4550  UNDIM A$,R$
4560  DIM A$(256),A1$(120),R$(80)
4570  R=0
4580  A$=""
```

```
4590  PRINT "8080 EDIT PROGRAM"
4600  PRINT "OUTPUT FILE ALWAYS ON DRIVE #1"
4610  PRINT "TYPE 8080 SOURCE IN FOLLOWING FORMAT"
4620  PRINT "LABEL OPERATION OPERAND COMMENT"
4630  PRINT "TYPE '*EXIT' TO EXIT TO MENU"
4640  INPUT "INPUT 8080 OUTPUT FILE NAME? ",R$
4650  PRINT "OUTPUT FILE ";R$;" WILL BE ON DRIVE #1"
4660  R$=R$+CHR$(0)
4670  OPEN (0,E,R$,1,1)
4680  IF E>1 THEN 4880
4690  INPUT A1$
4700  A1$=A1$+CHR$(0)
4710  A=LEN(A1$)+LEN(A$)
4720  IF A>256 THEN 4760
4730  A$=A$+A1$
4740  IF A1$="*EXIT"+CHR$(0) THEN 4770
4750  GOTO 4690
4760  IF LEN(A1$)<1 THEN 4810
4770  PUT (0,E,A$,R)
4780  R=R+1
4790  IF E>1 THEN 4850
4800  IF A1$="*EXIT"+CHR$(0) THEN 4850
4810  REM
4820  A$=""
4830  A$=A1$
4840  GOTO 4690
4850  CLOSE (0,E)
4860  UNDIM A$,R$
4870  GOTO 10
4880  PRINT "8080 EDIT FILE ERROR = ",E
4890  STOP
```



Letters

EASING MICROSOFT GLITCHES

Dear *DDJ:*

This letter is about a number of problems in interfacing the routine to basic as an example for your magazine.

Microsoft basic with CP/M was ordered from Lifeboat thru a local dealer and was finally delivered this month. There were a few problems, however. The most bothersome:

1. The printer was set up for a port that didn't exist on my system.
2. The ports starting at CF00 were initialized by track 0 correctly but then clobbered by CP/M in loading, so that when any of those ports were used the system would lock up.
3. The OSI machine uses ACIA's not the Z-80 ports for IO. This means that the Altair basic's IN OUT WAIT will not work.
4. If a CHR$(9) is printed (or LPRINTED) what is actually printed is a string of CHR$(32)s.

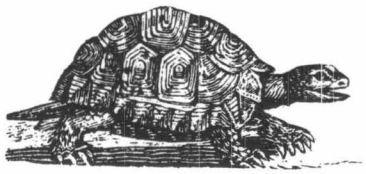
Problems one and two were corrected by rewriting the system's I/O routines. (BE00 & BF00). Since these routines were being replaced, another routine was added to solve problems three and four. In Basic ten USR calls are allowed. When the call is executed, register A contains the type of parameter passed (2=integer, 3=string, etc.). If A=2 then HL will point to the number. If A=3 then DE will point to the length of the string (same as VARPTR). By passing different parameters one USR call can be made to handle several routines. The following program will print a string to the printer if passed a string. It will input one ASC (character) entered on the keyboard if passed a single or double precision number, or will position the cursor if passed an integer (Y+X*256). I hope these routines may be of some use to someone.

Yours truly,
L. Barker

2305 W. Winona
Chicago, Ill. 60625

```
                ; OSI correction available as a USR call in BASIC CP/M

BF60            ORG     0BF60H
0000 =          CRTIN   EQU     0000H   ; fill in value
0000 =          CRTOUT  EQU     0000H   ; fill in value
0014 =          CURSE   EQU     14H     ; position cursor control
0002 =          MASKI   EQU     2       ; input port ready status

BF60 3D         START:  DCR     A
BF61 3D                 DCR     A
BF62 CA75BF             JZ      INTEG   ; A=2
BF65 3D                 DCR     A
BF66 CA83BF             JZ      STRING  ; A=3
BF69 CD0000             CALL    CRTIN   ; get one character
BF6C 6F                 MOV     L,A     ; and pass it back
BF6D 2600               MVI     H,0     ; to basic
BF6F E5                 PUSH    H
BF70 2A0501             LHLD    0105H   ; OSI's return integer
BF73 E3                 XTHL            ; address
BF74 C9                 RET

                ;
                ; If A=2 then position cursor (CRTOUT prints reg C)
BF75 0E14       INTEG:  MVI     C,CURSE
BF77 CD0000             CALL    CRTOUT
BF7A 4E                 MOV     C,M     ; value of Y
BF7B CD0000             CALL    CRTOUT
BF7E 23                 INX     H
BF7F 4E                 MOV     C,M     ; value of X
BF80 C30000             JMP     CRTOUT
                ;
                ; If A=3 then print string to LPT (even CHR$(9))
BF83 EB         STRING: XCHG            ; pointer now in HL
BF84 46                 MOV     B,M     ; B=len of string
BF85 04                 INR     B       ; If B=0
BF86 23                 INX     H
BF87 5E                 MOV     E,M
BF88 23                 INX     H
BF89 56                 MOV     D,M     ; got string address
BF8A EB                 XCHG            ; in regs HL
BF8B 05         SLOOP:  DCR     B       ; end yet?
BF8C C8                 RZ
BF8D 4E                 MOV     C,M
BF8E 23                 INX     H
BF8F CD95BF             CALL    LPT     ; print a byte
BF92 C38BBF             JMP     SLOOP
                ;
                ; Routine of how LPT could look
BF95 E5         LPT:    PUSH    H
BF96 F5                 PUSH    PSW
BF97 2AA5BF             LHLD    ADDR    ; since all the ports are
BF9A 7E                 MOV     A,M     ; handled the same
BF9B E602               ANI     MASKI   ; this output is easily
BF9D CA9ABF             JZ      LPT+5   ; redirected
BFA0 23                 INX     H
BFA1 71                 MOV     M,C
BFA2 F1                 POP     PSW
BFA3 E1                 POP     H
BFA4 C9                 RET
                ;
BFA5 00CF       ADDR:   DB      00,0CFH
BFA7                    END
*
```

```
^C
10 ' Demo progam for USR calls
20 DEFINT C
30 DEFSTR S
40 DEFUSR0 = &HBF60
50 :
60 INPUT"Position cursor at X";CX
70 INPUT"Position cursor at Y";CY
80 C=CY + CX*256
90 C=USR0(C)
100 PRINT "Done";
110 :
120 PRINT CHR$(5);      :' get cursor position
130 CY = USR0(Z)
140 CX = USR0(Z)
150 PRINT "X="; C\256
160 PRINT"Y="; C MOD 256
170 :
180 PRINT "A key? ";
190 C = USR0(Z)
200 PRINT "Thank you."
210 :
220 LINE INPUT"A string? ";S
230 S = USR0(S)
240 END
```



MAKING A GOOD THING WORK

Dear *DDJ,*

Mark Zeiger's program in *DDJ* #41 for changing XEK files to CP/M files was very good. Except for the fact that it would not work on my XEK files, it really was well written and extremely valuable. This was my first exposure to CP/M and Mark's piece cleared up a lot of confusion on my part. My Northstar system was bought secondhand with no guarantees to the usefulness of the software. Without Mark's fine article, CP/M would still be an unknown quantity to me. Now at least I have my feet wet.

I think the problem is within my XEK files. It looks like XEK treats the first three spaces encountered after the (1) length byte; (2) line number; and (3) dummy character (space), as field delimiters. Mark's program only expanded the first space to a tab and all others were left as spaces. The enclosed listing of the critical program area shows how I adjusted the original program to fix my XEK files. I also put in a (;) comment delimiter after the third tab for continuity. (My XEK treats all data after the third tab as comments with or without a field delimiter character).

Finally, there are at least two other areas of incompatability. First, in XEK all label references are truncated to five characters for comparisons but are left full length in the source files. Therefore 'MACHINE' in the label field would be evaluated as 'MACHI' and 'MACHINE' in the operand field will reference 'MACHI' in the label field. In the CP/M assembler more characters are compared and this causes 'U'nidentified errors.

The second problem has to do with negative referencing. Since I use a Z-80 CPU but no Z-80 software, I simulate relative jumps with the Z-80 code for the particular relative jump followed by a computed offset.

```
JRZ     EQU     28H
STAT    EQU     03H
MASK    EQU     02H
*
*
TTYOT   IN      STAT
        ANI     MASK
        DB      JRZ
        DB      LABEL-$-1
        IN      DATA
```

While these instructions will be assembled by XEK, the CP/M will not evaluate expressions like 'LABEL-$-1' if the result is negative. It seems to do OK with forward references.

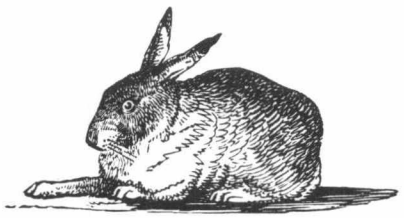
Again, thanks to Mark for his inspiring program. I would like to hear from other CP/M on Northstar users who can help me get DDT and NRELOC working or just exchange software. Except for the APPLE store here in Corvallis, there isn't much micro activity.

Sincerely,
Gene R. Head

2860 NW Skyline Drive
Corvallis, Oregon 97330

THESE LINES OF TEXT HAVE BEEN MODIFIED FROM THE ORIGINAL TEXT APPEARING IN *DDJ* #41, WRITTEN BY MARK ZIEGER.

```
NEWLINE MOV     A,M
        CPI     01
        JZ      EOFMARK
        MVI     A,0
        STA     TAB
        MVI     A,3     ;SET MAXIMUM TABS PER LINE
        STA     TABCNT  ;AND SAVE IN TABCNT
        MOV     A,M
        SUI     6
        STA     EOLCNT
        MOV     A,B
        SUI     6
        MOV     B,A
        INX     H
        INX     H
        INX     H
        INX     H
        INX     H
*       INX     H
        CM      READREC
        CZ      READREC
        CM      RESTORE
OLDLINE LDA     TAB
        CPI     0
        MOV     A,M
        JNZ     NOTAB
        CPI     03BH
        JNZ     ASTK
        STA     TAB
        JMP     NOTAB
ASTK    CPI     '*'
        JNZ     BLANK
        STA     TAB
        JMP     NOTAB
BLANK   CPI     ' '
        JNZ     NOTAB
        MVI     A,9
;
;  DON'T BLOCK FURTHER TABS WITH A SPACE
;
NOTAB   INX     H       ;SWITCH ORDER HERE SO
*       DCR     B       ;OUTBUF CAN BE UPDATED
        CZ      READREC ;WHEN INSERTING ';' WITHOUT
FILL    STAX    D       ;DESTURBING INBUF
        INX     D
        DCR     C
        CZ      WRITEREC
        CPI     09
        JNZ     PAST
        LDA     TABCNT  ;HERE IF CHANGING SPACE TO TAB
        DCR     A
        STA     TABCNT
        JNZ     PAST
        MVI     A,';'   ;INSERT (;) AFTER THIRD TAB ON LINE
        STA     TAB     ;FLAG 'TAB' TO BLOCK FURTHER FIXING THIS LINE
        JMP     FILL
PAST    LDA     EOLCNT
        DCR     A
        STA     EOLCNT
        JNZ     OLDLINE
        MVI     A,0AH
        STAX    D
        INX     D
        DCR     C
*       CZ      WRITEREC
        JMP     NEWLINE
EOFMARK MVI     A,0AH
        STAX    D
        INX     D
        DCR     C
        CZ      WRITEREC
        MVI     A,1AH
        STAX    D
        RET
;
TABCNT  DB      3       ;HOLD MAXIMUM TABS PER LINE
;
*
```



SAVE TIME

Dear *DDJ*:

Here is an observation of a serious error in programming style that has probably been noted before, but probably not as often or as strongly as it should have been.

First let me say that my programming style is as peculiar in many ways as anyone's and is certainly not beyond criticism; however, I feel that most will agree with me in regard to this particular point.

Here is an example (can you find the problem?):

```
10 REM ** THIS PROGRAM IS TO POINT OUT A PROBLEM **
20 PRINT "COMPUTERS NEVER MAKE MISTAKES"
30 LET X = X + 1
40 IF X = 10 THEN 60
50 GOTO 10
60 REM ** LETS STOP PRINTING THIS JUNK **
70 END
```

Do you see the error? It occurs twice in the listing. In lines 40 and 50 there are branches to remark statements. This may seem trivial in a small program like the one above; however, many times it is useful to strip out remarks to save space in a very large program. If in the above example line 10 were deleted, the program would not run until you changed line 50 to 'GOTO 20'. It is my contention that, if the programmer had done that to begin with, there would have been no problem. It is particularly easy to make this mistake in BASIC since every line number is potentially a label, but I have seen examples of this in other languages also.

Sincerely,
Bruce B. Thomson

216 Northwest Terr.
Silver Spring, Md. 20901

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 46 | June/July 1980 | Volume 5, Issue 6

A REFERENCE JOURNAL FOR USERS OF SMALL COMPUTERS

On The Subject Of Networking

A Survey of Computer Networks 6
Art Kleiner

The Personal Computer Network (PCNET) Project: A Progress Report 10
Dave Caulkins

MCALL-C: A Communications Protocol for Personal Computers 16
Tim J.Pugh, Jr.

On Modems 22
William Smith

Random Ruminations on Networks of the Future 40
Jonathan Sachs

... and, as always, even more

A Survey of Computer Networks

BY ART KLEINER
Box 428
Sausalito, CA 94965

As the Information Age impends, it's beginning to look more and more like an interconnected stream of computer networks. Most networks (except for the Arpanet) have been active three years or less; yet already they're having as much impact on the people who use them as the first automobiles did on early drivers. Not all networks are connectable to all types of terminals, but the differences in their design should be noted by computer users, who are prominent among the first computer network memberships and influential in the directions these networks will take.

In this survey, an update of an article in the Summer, 1979 CoEvolution Quarterly *(Box 428, Sausalito, CA 94965), I tried to ask the political questions: Who owns the networks? Who has access to them? How are they paid for? How is their interaction with their members designed? What provisions exist for privacy? How do people get in touch? This is* not *an exhaustive survey (for example, it does not include the most influential network of all, the* Arpanet*), but it should give some idea of what options are available. The two-paragraph descriptions are by necessity over-simplified.*

In meeting Dr. Dobb's *deadline, it was not possible to fact-check* all *the descriptions below. A more comprehensive article on the subject will appear in the* Next Whole Earth Catalog, *in September 1980.*

QUBE, a two-way cable TV station from Columbus, OH; may soon be introduced to Boston and Pittsburgh. Started December, 1977. Sponsored by Warner Communications Corporation and American Express. Contact: Leo Murray, Vice President of Public Affairs, 75 Rockefeller Plaza, New York, NY 10019.

This widely-publicized, two-way cable TV station offers normal cable TV, pay TV and local channels with some interactive programs. On interactive shows, viewers push yes/no/multiple choice buttons to respond to opinion polls or to order products from commercials. A central computer monitors each TV set every 6-10 seconds, recording responses and what channel the set is tuned to. The system is used for political polls, exam-taking in educational programs, talent shows where the home audience rates the contestants, and commercials with built-in market research. Since American Express purchased part of Warner Communications last year, there has been increasing emphasis on Electronic Funds Transfer and shopping at home.

Warner's research showed people wanted the simplest system possible – hence the limited amount of interaction. "Mrs. Jones doesn't want to take the time to learn to be a computer expert," spokesman Leo Murray said. QUBE warns viewers in advance when their responses are recorded, and doesn't release records without subscribers' consent.

To connect you need: a TV, a $19.95 one-time installation charge, and $10.95 monthly rental. Pay-TV programs average $1-$4 each.

VIEWDATA is a brand name for a type of interactive text service originally developed by the British Insac Corporation for the British Post Office. Like Kleenex and Xerox, the brand name has come to mean the entire type of product, even though there are many competing variations by now.

The basic British Viewdata system involves an adapted TV set with a telephone connection and a remote control switch box that looks like a pocket calculator. Viewers press numbers to call for pages of travel information, business information, retail advertising, or constantly updated news. News agencies and advertisers pay the British Post Office for the opportunity to put their messages on the air. A central computer is signalled through the viewer's telephone, and responds by finding the appropriate Viewdata "page" and flashing it on the viewer's screen. Pages appear in bright colors and block letters; each page includes a list of 10 other pages that can be called up for more information, like a cross between a library and a chain letter. (Information scientist Jacques Vallee called it "a boring maze of branching tunnels.") Sometimes people must go through seven or eight pages to find the one they want. There is more choice than with QUBE, but users are still limited; they cannot communicate with each other, except through pre-written electronic "greeting cards."

To connect, British viewers need an adapted TV ($300-500 more than a regular TV, according to one British correspondent; $30-50 more, according to Insac in New York) plus the price of a phone call. Each page carries a charge, usually 1¢-4¢.

Several companies are developing Viewdata systems in America. General Telephone and Electronics owns the American rights to the British Viewdata system, through Insac Viewdata, Inc. an American Insac subsidiary. GTE also owns Telenet, a national network for transmitting computer-signals cheaply, and Sylvania, the television manufacturing company. They are expected by some observers to combine all three into a business-oriented national Viewdata system, which may eventually also be adapted for home use.

The Knight-Ridder newspaper chain is testing a Viewdata-type system called Viewtron in Miami. Interestingly, GTE's major competitor, American Telephone and Telegraph, is supplying the hardware for this system. Contact: Viewdata Corporation of America, One Herald Plaza, Miami, FL 33101.

KSL-TV, Salt Lake City, and KMOX-TV, St. Louis, both have had experimental demonstration systems running for two years. In both cases, a public system is still pending FCC approval. KSL is affiliated with the Mormon church; KMOX is owned by the CBS Broadcasting

Network, whose executives are reportedly worried that viewers will switch to the Viewdata channel during commercials. Other Viewdata-type systems are running in France (Antiope) and Canada (Telidon, with reportedly a very sophisticated graphic "look" and animated pages.)

PCNET (Personal Computer NETwork) is nationwide. Available from People's Computer Company, Box E, Menlo Park, CA 94025.

A committee of volunteers, chaired by Dave Caulkins of Los Altos, CA, have developed an electronic mail system for Commodore PET personal computers which operates over telephone lines. Most of the work involved developing a method of correcting the large number of errors caused by garbled phone-line transmission. Users can start with PAN programs (an illegitimate son of an Arpanet mail system called Hermes, Caulkins said) to send messages but need the full PCNET program (price not available at deadline) to transfer files and programs without losing characters. Users can arrange for their computer to send the mail late at night, when they're asleep and the phone rates are lower.

There is no central switching computer in PCNET. "That's our political decision," Caulkins said; decentralized networks are harder to control. To protect privacy, users are encouraged to develop automatic message-encoding programs. Future plans include adapting other microcomputer brands and sending messages over short-wave radio.

PCNET has been recently joined by similar programs doing the same for other brands of computers. Information Unlimited, Inc., 793 Vicente Ave., Berkeley, CA 94707, has developed a message system for Apples which they call Easy Mover. Other programs are also being developed – all presumably with different protocols, and all unable to send messages to any but the same program.

THE SOURCE is nationwide. Started June, 1979. Available from Telecomputing Corporation of America (TCA), 1616 Anderson Road, McLean, VA 22101.

This was the first home-terminal consumer information network in America, and certainly the most ambitiously promoted. People with home terminals or personal computers can call up specific information, including a version of the UPI news wire and a version of the bibliographic New York Times Information bank. In addition, the Source offers electronic mail and bulletin boards, programs which analyze statistics off the Wall Street stock index, and a clientele which is hungry to communicate with eachother.

All this is transmitted over Telenet and Tymenet, two national computer-signal transmission networks which can be dialed locally from most cities, and which are much cheaper than long-distance telephone calls. The Source intends to market its accounts through franchised stores called "Telecomputing Centers." It currently has about 4,000 active subscribers.

According to many users, some of the Source's promised features – such as airline reservation guides or conferences – have been slow to materialize or have not lived up to expectations. Slow response time and frequent breakdowns have reportedly also been perennial problems. However, the Source deserves credit for bringing the idea of a home-based computer network with many more options than Viewdata out into the open. TCA just signed a contract with a national library network (OCLC) to put Source terminals in public libraries around the country.

To protect privacy, records are kept on the Source of how often each piece of information is called up, but not of who is calling for what. A directory of users is limited to first name, state, and a one-line description of interests – which, ironically, cuts short the easier connection-making which a fuller directory would provide. TCA offers a $5,000 reward to anyone who can break into the system's central account and access other people's files.

To connect, you need a terminal or personal computer and telephone hookup, a $100 entry fee (unless you buy your terminal from the Source), and $2.75 per hour of use. The service is limited to off-business hours and weekends – during business hours, it's $15/hour.

MICRONET was started in October, 1979. Contact: Personal Computing Division, CompuServe Incorporated, 5000 Arlington Centre Blvd., Columbus, OH 43220.

A home-terminal computer network, similar to the Source, piggybacking off the CompuServe Time-sharing company, Micronet is younger than the Source and concentrates on business programs, file editing, and electronic mail. Except for a stock price index, there are no news-wire style databases yet; reportedly some are planned for introduction later this year. Micronet is more conservative in its promises than the Source, but much quicker in response time. A Micronet accounts costs $9 in start-up fees and $5 for every connect hour, when signed-on via CompuServe's own packet-switching network. In some cities signing on via Tymenet is necessary, at an extra $2/hour.

EIES (Electronic Information Exchange System) is nationwide, begun October, 1976. Originally funded by the National Science Foundation; now operating as an independent computerized conferencing system. Organized by Murray Turoff and Roxanne Hiltz, co-authors of *The Network Nation*. Contact: Anita Graziano, Computerized Conferencing and Communications Center. New Jersey Institute of Technology, 323 High Street, Newark, NJ 07102.

EIES is a community of about 900 cross-continental members, divided into groups devoted to specific topics, such as Technology for the Handicapped or General Systems Theory. Many members belong to several groups, and members are encouraged to communicate with as many others as they have time and interest for. They collaborate on research, conduct several informal news networks, introduce themselves to each other via keywords that signal common interests, and gradually build up on-line libraries of relevant information.

EIES has many members active in community information exchange and social alternatives; its mix of people from corporations, government, non-profit groups and the arts is unique among computer networks. It's also the site of several specially-designed experimental programs for weaving interactive stories that respond to answers from the reader; for filtering through masses of technical information quickly; and for corresponding anonymously or under pen names. Most EIES members sign in through Telenet. There are about 30 European members.

To become a member, you need to apply at the address given above. Accounts cost $66/month and $3.75/hour. Trial accounts are also available for no monthly fee and $5.25/hour. Groups can buy cooperative slots (which is true of most networks), and can (for about $350,000) buy the hardware, software and maintenance to create their own EIES-like system.

Some other computer conferencing systems, with different mixes of membership and capabilities than EIES:

- **I. P. Sharp**, a less expensive system based in Toronto, piggy-backed onto the Sharp Timesharing network, adapted for use by businesspeople and artists. Sharp's conference program is limited to one-line comments and can be quite surreal when five or more people are involved.
- **Confer**, a university-oriented conferencing network with a strong feeling of community, based at the University of Michigan.
- **Comet**, a business conferencing system, one of the many networks developed out of the Arpanet. Contact: Computer

Corporation of America, 575 Technology Square, Cambridge, MA 02139.

• **Planet/Notepad**, a relatively expensive but easy to learn business system, managed by parapsychology investigator/information scientist Jacques Vallee. Planet grew out of Forum, one of the first and most influential computerized conferencing networks, at Menlo Park's Institute for the Future. For information on Planet, contact: Sal Suniga, InfoMedia Corporation, 530 Lytton Avenue, Palo Alto, CA 94301.

PLATO. Started in 1969. Nationwide. Originally developed at the University of Illinois as a computer-aided instruction program. Now owned and operated by the Control Data Corporation, P. O. Box O, Minneapolis, MN 55440.

Like Viewdata, PLATO produces its "lessons" in pages – but unlike Viewdata, humans can interact much more with each page, since they have a full keyboard. Each page asks the user a question and produces new pages depending on the answer. Pages compose themselves on the screen as if the program were twiddling a full-color Etch-A-Sketch behind the scenes. Users can send messages to anyone else on the system, although the system does not store the messages. Up to five conversations can go on a screen at any one time. PLATO is used for business training, remedial education, and training in prison.

The public school district in Irvine, California is developing a plan to make PLATO available through a city-wide cable TV system. Current plans call for a trial system to be developed sometime during 1980.

The specially-designed PLATO plasma-screen terminals cost $5,000 or more and connect time to the central system is expensive. Most users travel to Control Data's own "Learning Centers" in many cities to use the systems. The public school district in Irvine, California is developing a plan to make PLATO available though a city-wide cable TV system sometime this year.

Control Data is also beginning a pair of computer networks for shaing information between small-farm farmers, "urban problem-solvers," and low-income groups. These are being developed among small groups of people now, and have provoked controversy among some appropriate technology and community information advocates, who distrust CDC's motives. For information on the Urban Technology Exchange, write: Linda Jadwin, Director, Urban Technology Exchange, HQS11B, Control Data Corporation, Box O, Minneapolis, MN 55440. For the opposition point of view, write to: Gil Friend, Office of Appropriate Technology, 1530 10th Street, Sacramento, CA 95314.

PANALOG was started December, 1977. An informal community of information scientists, students, and researchers in technology for the deaf, based at General Telephone and Electronics Labs outside Boston. Contact: Ed Housman, GTE Laboratories, 40 Sylvan Road, Waltham, MA 02154.

Housman used excess computer time on the GTE computer to establish his own informal conferencing network. Members of the network use it to discuss information theory, plan meetings, and generally keep up a running rap on anything of interest. Many of its members are deaf, using it in part to keep each other informed of news involving the deaf community. More specifically, though, Panalog is notable for its whimsical human-ness (users receive three Chinese-fortune-cookie-style epigrams when they log off).

Five major American corporations are expected to introduce large-scale computer network systems for business and possibly home use during the next few years. They are:

1. **American Telephone and Telegraph**, in addition to supplying the hardware for Knight-Ridder's Viewdata experiment in Miami, AT&T is testing a terminal-in-the-home Electronic Information Service in Albany, New York. People use computer terminals attached to their phones to look up directory assistance numbers and emergency telephone numbers. In addition, there is Advanced Communications Services (ACS), a multi-computer nationwide packet switching network first proposed in 1978. ACS as first proposed was an elaborate system which could accept many different types of terminals and computer hook-ups, provide electronic mail and conferencing, rent computer programs and store data. AT&T has withdrawn its proposal, reportedly because they hadn't worked out the technical bugs of such a complex system yet (and possibly because the regulatory climate wasn't right). There is concern among current packet-switching networks and consumer groups that the monopoly status and enormous assets of AT&T would give the company an unfair advantage over the computer network competition.

More than any other large firm, AT&T has shown interest in the terminal in the home; a recent issue of a Bell employees' magazine was devoted to the subject, with a painting of an electronic home of the future by Studio 54 designer Ron Doud. Reportedly they are concerned that involvement of many other companies in computer networks would lead to involvement of many other companies in voice transmission as well.

2. **General Telephone and Electronics** is developing several concurrent plans for office-of-the-future mail and information systems, possibly including manufacturing terminals through its Sylvania subsidiary. GTE owns Telenet, and is developing its own electronic mail system on that network. It also owns the rights to develop the British Viewdata system in America. According to a staff member in their Viewdata group, "We're looking at Business applications initially; but we anticipate that this will evolve at some point into a consumer system, depending on the availability of information providers and terminals." Panalog was also developed for GTE, and some of its features may be considered for other office-of-the-future systems.

3. **Xerox**, earlier this year, released plans for two complementary computer networks: *Ethernet*, which runs through coaxial cables in a single building, is already available. *XTEN*, which would use its own satellites to transmit computer signals over radiofrequencies between *Ethernets*, is on application before the FCC. Like ACS, Xten would permit many types of terminal and computer connections; Ethernet is marketed in such a way that connecting to it may become a universal standard.

4. **Satellite Business Systems**, part-owned by IBM Comsat, and Aetna Life Insurance, plans a network to transmit computer messages and video-conferencing between large corporate offices over the same satellite.

5. **Exxon** is widely rumored to be planning some sort of national computer network, tied to subsidiaries which manufacture fiber-optic transmission lines and the laser equipment to send messages through them. Exxon owns subsidiaries which manufacture the Oyx terminal; the Zilog chip; and Qwip, the most successfully selling facsimile transmission device. Another Exxon subsidiary is developing automatic telephone-answering computer systems. Reportedly, Exxon is also developing a method for Qwip to encode and store facsimile transmissions digitally, so they could be (eventually) tied into computer networks.

Some interesting smaller-scale computer network proposals (which don't exist yet):

Datacast. San Francisco, beginning August, 1980. Formerly known as Digicast. Proposed by Wireless Digital, Inc., a new company managed by Jim C. Warren, Jr., who organizes the annual

West Coast Computer Faires. Contact: Wireless Digital, 345 Swett Road, Woodside, CA 94062.

Like Teletext, Datacast will transmit words and pictures over broadcast waves in a steady stream. Using Subcarrier Authorization channels, which are unused FM frequencies linked to existing radio stations, Datacast will be a steady stream of wire service news, financial data, real estate listings, classified ads, weather and library resources. To receive it, users will need a modified personal computer with a radio receiver ($400 or less above the cost of the computer). As in Teletext, the computer will pull out only those items which it has been programmed to catch. Until it gets going, Datacast will be covered in its own newsletter, available from Wireless Digital for $5/year.

Xanadu. Nationwide, beginning sometime this year. Proposed by Project Xanadu. Contact: Theodor Nelson, Box 128, Swarthmore, PA 19081.

Nelson, who published the milestone book *Computer Lib/Dream Machines* in 1972, described there his plan for a computer network system in which people would edit alternate versions of text, jump back and forth between the versions, and make royalties based on how many people would call for their writing. His current plans call for such a system to begin running by the end of this year, possibly through locally franchised "Information Centers." Xanadu will be used for conferencing, writing, editing and reading. Rather than a single index to everything on the system, users will be encouraged to create their own indexes to what interests them and publish them for others.

Nelson said they are looking at hourly costs of $2 (300 baud) and $5 (1200 baud). Xanadu may operate its own transmission hardware instead of relying on Telenet or Tymenet; or the Xanadu systems may only operate locally.

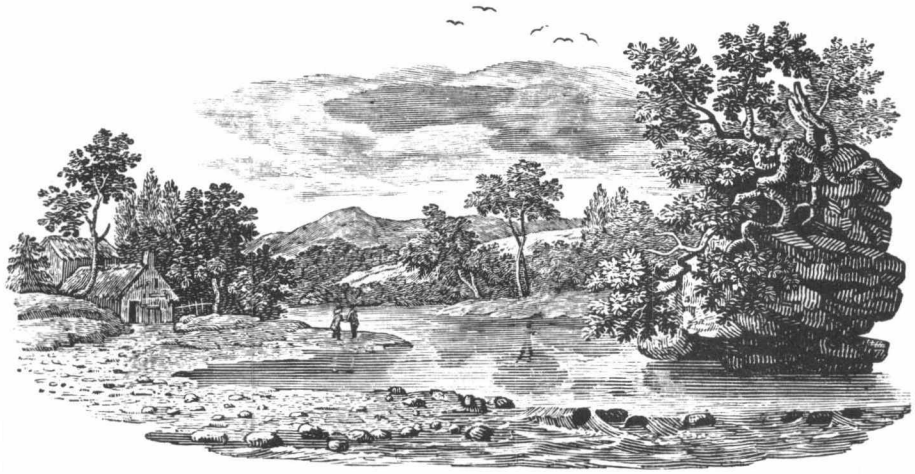
The Community Memory Project, San Francisco, is beginning sometime this year. Sponsored by the Community Memory Project. Contact: Sandy Emerson, 916 Parker Street, Berkeley, CA 94710.

Convinced that computer networks should be decentralized, open to everyone and service-oriented, Community Memory put inter-connected terminals in Berkeley public places between 1973 and 1975. People used the system as an electronic bulletin board for news, messages, classified ads and poetry. Others used the terminals to pull out information with the appropriate keywords, or to add comments to existing messages. A new similar system will be established in a Bay Area neighborhood. Up to 20 terminals will be placed in community centers and neighborhood gathering places. All information in the system will be entered by the people who use it, and no one will connect to Community Memory through private terminals. In the meantime, some of the text editing and file-arranging programs which will compose the Community Memory system are being offered for sale separately.

OCLC/Home Delivery of Library Services is starting mid-1980 in Columbus, OH at first; then possibly nationwide. Contact: Philip Schreiber, OCLC Inc., 1125 Kinnear Road, Columbus, OH 43212.

OCLC used to be the Ohio College Library Center. Their new home system will use a keypad like QUBE's (but smaller and less expensive), an adapted TV, and the telephone lines. Columbus residents will use OCLC's system to bank at home and to call up pages of an on-line encyclopedia. OCLC is also planning to put Source terminals in libraries nationwide.

AIM is an EIES-like conferencing system, with an emphasis on developing a community computer system for alternative communities. Contact: Steve Heitmann, Rt. 2, Box 502A, Portland, OR 97231.



THE PERSONAL COMPUTER NETWORK [PCNET] PROJECT:

A PROGRESS REPORT

BY DAVE CAULKINS
PCNET Project
1263 El Camino Real, Box E
Menlo Park, CA 94025

INTRODUCTION

This article describes the PCNET project, its goals, a brief history of activity, PCNET status today, and PCNET's future plans.

GOALS

The goal of the PCNET project is to provide a reliable, error free and low-cost means of transferring files, programs or messages among personal computers (PCs) via the dial-up telephone system or other media. To achieve this goal in a reasonable way the PCNET Project adopted a series of system design objectives:

1. PCNET software will be hardware- and operating-system independent; any PCNET-equipped computer will be able to communicate with any other PCNET-equipped computer.
2. PCNET will be adaptable to as broad a range of existing hardware and software systems as possible.
3. PCNET will be a distributed system; any two computers running PCNET software will be able to communicate with no requirement for additional centralized resources. However, this will not preclude time-sharing systems and other centralized services from running PCNET 'front ends' in order to communicate with PCNET equipped small computers.
4. PCNET will be capable of unattended operation; any two PCNET systems will be able to initiate a telephone connection, transfer information and terminate the connection without the presence of an operator at either end. This will allow PCNET systems to take advantage of the low telephone rates available from 11 PM to 8 AM.
5. No personal computer user will require more than a single telephone line to participate in PCNET.
6. PCNET will be layered; i.e., PCNET software will be divided into distinct functional layers. Communication between layers will be sparse and well defined. Any layer can be modified or even replaced with minimum effect on the others. This kind of design should make PCNET readily adaptable to changing opportunities in hardware and media. Modems are going to be capable of higher speeds, and communication by other media such as radio (see discussion of Packet Radio below) will become possible. Layering allows relatively easy accommodation to these technological advances.

PCNET History

The PCNET Project (then called the PCNET Committee) was started at the first West Coast Computer Faire in April of 1977. PCNET was and is a non-profit organization of volunteers; all PCNET developed software is placed in the public domain. PCNET is now part of People's Computer Company. The first year of PCNET activity produced a protocol design (a protocol is a set of conventions for computer-to-computer telecommunication) and some experimental software mechanizing protocol operation. This software was demonstrated at the second Faire in March of 1978; an 8080 based system in Santa Clara, CA sent a file to a PET at the Faire in San Jose, CA with both machines running PCNET software. Noise was deliberately introduced on the phone line to damage some blocks of data; the two machines successfully detected the bad blocks and retransmitted good ones.

In 1979 the TNW Corporation and PCNET developed PAN, a small electronic mail system. This sytem lacks the power of the full PCNET protocol, but is quite adequate for handling English text messages and allows people with limited telecommunications experience to get a feel for what telecommunication is all about. The first version of PAN runs in the Commodore PET and TNW488 modem; a program tape and instruction manual are available from P.C.C. for $12. We're working on a version for the Apple II and Micromodem II, and intend to do one for the TRS-80. See Appendices B and C for more information about PAN.

PCNET STATUS

PCNET Design Description. This section presents a brief description of PCNET design. Each PCNET installation is called a Node; it consists of a computer, a modem, and the PCNET software. The software will activate one of two capability levels: Simple, for machines of small size or limited resources, and Advanced, for more powerful machines. A 48K Apple II with a Micromodem II is an example of a machine which could become an Advanced node. In keeping with PCNET design objectives all nodes will be able to communicate in Simple mode, but two nodes able to switch to Advanced mode will be able to do more and do it faster. Node-to-node communication always starts in Simple mode, but switches to Advanced mode as soon as possible if both nodes can handle it.

Figure 1 shows a diagram of the PCNET layers as they are implemented in hardware and software in a PCNET node. The node that initiates a call (dials the phone) is called the originate node; the node that accepts a call (answers the phone) is called the answer node. Information starts at the top (server process) layer of the originate node, works its way down through the layers and across the phone line to the answer node. There it goes back up through the answer node layers to the server processes. At the top (server) layer in both nodes area the programs actually carrying out user functions – for example a File Transfer Program (FTP), an electronic mail program (MAILER), etc. PCNET can communicate data to and from several simultaneously active server programs.

Let's follow the sequence of events in detail, assuming that both FPT and MAILER are active, and that the phone connection to the answer node system has already been established. Starting at the top (server) layer in the originate node: each server process passes a stream of data to the Directing layer; MAILER is sending an electronic mail message,

and FTP is sending an object program to be executed later in the answer node computer.

Directing Layer. In this layer the two streams of data are broken up into Directed Blocks; each such block has a header with the address of the server process in the answer node to which it's to be delivered, and a data section containing a piece of actual data. The Directed Blocks are then passed to the Validating Layer.

Validating Layer. Here the Directed Blocks have additional header information added, and an error detecting code appended to the end of each block. The resulting Validated Blocks are passed to the Translating Layer.

Translating Layer. In this layer the Validated Blocks undergo a translating process; they are converted into Radix-41 format if this is a Simple node, or into transparent binary if this is an Advanced node.

Radix-41: Radix-41 is a 2 byte to a 3 character packing scheme invented by Mike Wilber; the conversion from binary to Radix-41 guarantees that no control characters appear in the transmitted data which might cause undesired actions by some operating systems. Thus use of Radix-41 guarantees that PCNET will run without interference under any operating system known to us. Either Radix-41 or transparent binary can handle data in binary, ASCII or any other form, but transparent binary supports higher transmission speeds.

Framing Layer. Here the Translated Blocks have delimiting characters added, resulting in Framed Blocks.

Hardware Layer. This layer consists of the modem and any other hardware required to interface to the dial-up telephone system. Modems suitable for PCNET use are under full software control; a detailed discussion of modems may be found in Appendix A. In the modem the bytes making up the Framed Blocks are translated into serial bits, each of which is then converted into one of two audio tones for transmission over the phone line.

Now let's follow the information as it goes through the same layers in the opposite direction in the answer node.

Hardware Layer. The answer node modem converts the sequence of audio tones into a serial bit stream, and this is turned into a sequence of Framed Blocks.

Framing Layer. Here the framing characters are used to identify the block starting points; they are then discarded resulting in Translated Blocks.

Translating Layer. In this layer the blocks are translated into binary; from Radix-41 if the nodes are operating in Simple mode, or from transparent binary if they are in Advanced mode.

Validating Layer. The error detecting codes in each block are evaluated; if the block is error-free an acknowledgement block is created and sent to the originating node, indicating to it that the current block has been successfully received and that the next block may be transmitted. The error detecting code and Validated Block header material are stripped from the block, and the resulting Directed Block is passed up to the next layer.

Directing Layer. The header of the block is examined to see whether it is intended for the FTP or MAILER process in the answer node; all remaining header material is stripped off and the data is passed to the indicated process.

PRESENT PCNET ACTIVITIES

A small group of PCNET people is presently (April, 1980) working intensively to develop 1 to 4 complete implementations of the PCNET protocol running in several different computer/modem pairs. The resulting software will be in the public domain; it will be distributed to anyone at modest cost and will probably be published here in DDJ. This software is also intended to serve as 'model' software in a PCNET implementation workshop to be held later in the year. We hope this workshop will produce many additional implementations covering the majority of computer/modem pairs in use today. All this software will support real but sparse Server level processes such as file transfer and electronic mail.

A volunteer organization like PCNET is not well suited to creating and maintaining software products. It's our hope that individuals and software companies will use PCNET software as the basis for products which will utilize PCNET protocols as a telecommunication standard.

FUTURE PLANS

Although the dial telephone network is easily accessed, it is far from ideal as a PCNET medium, since it is optimized for voice and not data. Phone system bandwidth is limited and costs are high and going higher. We believe that radio offers both lower cost and higher bandwidth. US Government research has shown that many PCNET type users can effectively share a single radio frequency allocation using a technique called packet radio. The Canadian Government has wisely assigned radio spectrum space (220-225 MHz) for Canadian radio amateurs to experiment with packet radio. We'd very much like to start similar experiments in this country. Canadian experience indicates that the entire radio transceiver and interface could be built for about $500, a price which would decline with any kind of production volume. Some reports by CARF (Canadian Amateur Radio Federation) on packet radio work in Canada are reproduced in Appendix D.

The PCNET project would like to see packet radio administered under licensing much more similar to CB radio than to amateur radio. We think packet radio equipment can be designed to prevent abusive behavior by ill-informed or malicious users.

APPENDIX A

PCNET Modems

All telephone based data communication systems require modems to function. The features required for a PCNET modem are as follows:

Hardware Features

- Receive sensitivity of at least −40dBm; −45 to −50 dBm preferred
- 10 pole receive filter; 6 pole transmit filter
- 300 baud operation

Features Under Software Control

- Answer/originate mode select
- Switch hook control; pulse dialing
- Carrier detection
- Incoming ring signal detection

Desirable But Not Necessary Features

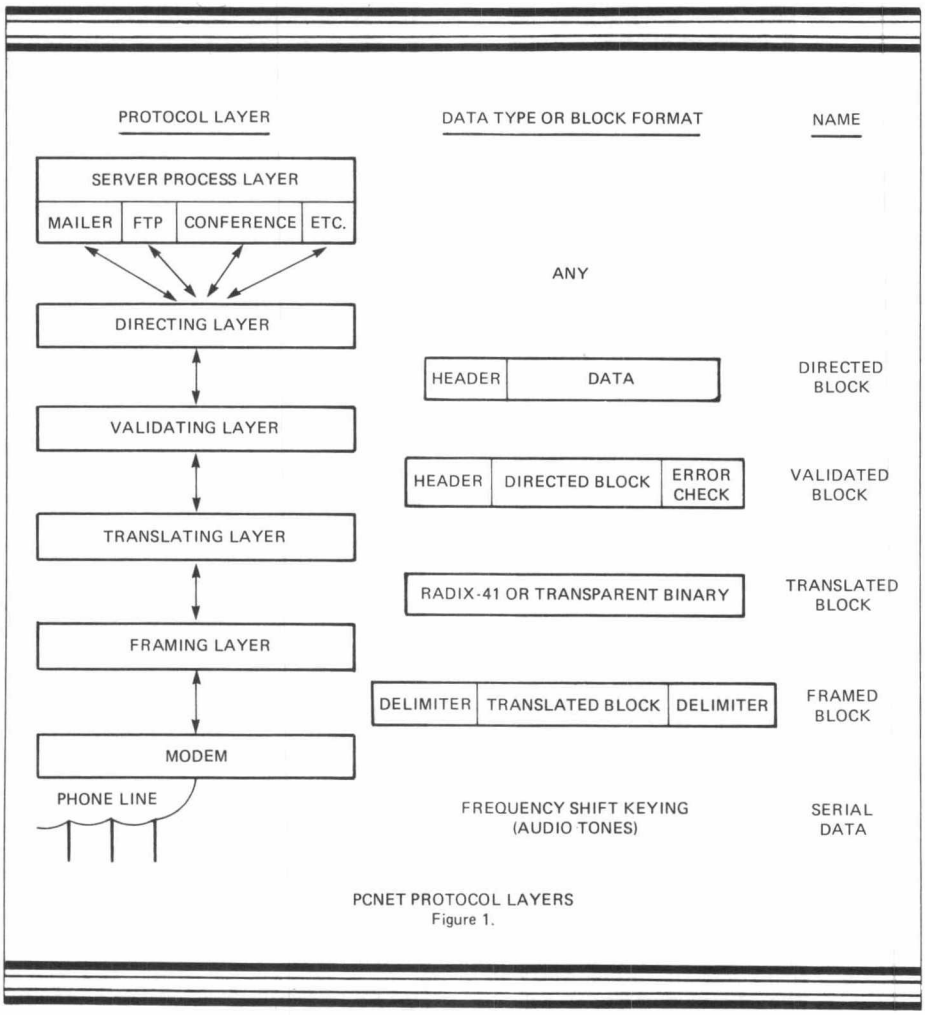
- FCC Part 68 registration, permitting direct connection to the phone line without an otherwise legally required Data Access Arrangement
- Telephone signalling tone discrimination, allowing software to detect DIAL TONE, BUSY, CALLED PARTY RING, etc.
- Software selectable baud rate
- Power up on incoming ring detect.

The PCNET Project has found a number of modems that are suitable for PCNET use. A list of them is given below:

TYPE – S100 Compatible

The MM−103, from Potomac Micro-Magic, Inc., First Lincolnia Bldg, Suite B1, 4810 Beauregard St., Alexandria, VA 22312. Costs $359.95; is FCC Part 68 certified.

The Micromodem-100, from D.C. Hayes



PCNET PROTOCOL LAYERS
Figure 1.

Associates, Inc., 16 Perimeter Park Dr., Suite 101, PO Box 9884, Atlanta, GA 30319. Costs $350.00; is FCC Part 68 certified.

The 88-MODEM, from International Data Systems, Inc., 400 N. Washington St., Suite 200, Falls Church, VA 22046. Costs $395.00

TYPE – Apple Compatible

The Micromodem II, from D. C. Hayes (see address above). Costs $389.00; is FCC Part 68 certified

TYPE – PET Compatible

The TNW488, from The TNW Corporation, 5924 Quiet Slope Drive, San Diego, CA 92120. Costs $320.00

APPENDIX B
PAN SOFTWARE

Just recently, the PCNET Project announced the availability of computer mail support software for use on personal computers. The software is called PAN, named for the illegitimate son of Hermes, the messenger of the Gods in Greek Mythology. The PAN software, in fact, was inspired by a sophisticated mail system called HERMES which was developed by Bolt, Beranek and Newman, Inc., for the DEC System 10 and System 20 computers.

PAN is a completely self-contained computer system. Any two PAN systems can exchange messages over an ordinary voice telephone line. PAN allows entry, review and automatic transmission of messages at times selected by the user. When a message is sent to a PAN it automatically answers the phone, receives and stores the message, and hangs up the phone to wait for the next incoming or outgoing message. Similarly when PAN has a message to be sent it automatically picks up the phone, dials the number of the recipient PAN, verifies that carrier is present and that PAN is active, transmits the message, and hangs up the phone to wait for the next transaction. Message exchange is possible between any two PAN systems regardless of the type of computer or modem used. Either or both PANs can record both incoming and outgoing messages on tape or disk.

PAN is the result of a collaboration between The Personal Computer NETwork (PCNET) Project of PCC and TNW Corporation. PAN is similar to Community Bulletin Board Systems (CBBS) in general functioning and command structure. It is different in that PAN is a distributed system; there is no single phone number and set of equipment through which all messages must pass.

The PAN system has communications advantages over both traditional mail and the telephone, and cost advantages over existing, expensive electronic mail systems.

Advantages of PAN over traditional mail:

- Immediacy of delivery (or if preferred, "time tag" delivery)
- A signal from the receiving computer allows the sending computer to mark the message "sent."

Advantages of PAN over telephone communication:

- PAN can record all received and transmitted messages for future reference
- The PAN user need not be present to transmit or receive information
- The PAN system has a "keep trying" feature which will wait until the line is clear to deliver the message.

Cost Advantages over existing electronic mail systems:

- Low entry cost
- PAN uses the dial-up telephone network
- Telephone charges can be quite low
- PAN software is available for a cost of $12, including manual
- PAN runs on a personal computer.

The present PAN system is best for letter (English language) communications. PAN is designed so that all existing computer *terminals* now in use in major universities and corporations can access PAN systems to read and write messages. Terminal-to-PAN communication is primitive due to the limited capacities of existing terminals.

The PCNET Project plans continued developments following this initial version of PAN. Some of these are detailed here:

- Improvements in the error management capabilities of PAN. At present the error rate of PAN messages is about 1 error for every 5 minutes of message transmission. This error rate is not adequate for computer programs or financial data. An error rate of 1 error for every two weeks of message transmission appears to be attainable.
- A file transfer capability will be supplied. This will allow a complete program to be sent from one personal computer to another over the telephone. Because of the low error rate the receiver of the program can be confident that the program will run successfully. This file transfer capability will allow 'bootstrap' loading of the PCNET program itself over the phone. To do this, a very short program is loaded by hand into the computer. This short program then manages the transfer of the much longer PCNET program from a remote computer via the phone line. By this process the PCNET programs can migrate from computer to computer very easily.
- The PCNET Project hopes to experiment with other than the telephone system as the communication link between PCNET computers. One possibility is radio which has the potential of markedly increasing the amount of data that can be transferred while decreasing the cost.

PAN Commands

E – Enter a message. The phone number of the PAN for which the message is intended must be in the

first line of the message. After message entry the user can select a mode assignment for the message.

I – Immediate; send this message now.

D – Deferred; hold this message for transmission later. The user is prompted for the time of transmission.

N – Make no mode assignment for this message. A mode can be assigned later as described below.

Q – Query; hold this message for subsequent retrieval by a non-PAN modem equipped computer or terminal.

M – Mode assignment. Assign a new mode to the message whose number is given by the user.

R – Retrieve a message. The message whose number is given by the user is displayed.

S – Summarize messages. The numbers, modes and initial portions of all messages in the buffer are displayed.

K – Kill (erase) the message whose number is given by the user.

U – Unload PAN, restore operating system pointers and write an end-of-file on the message record tape or disk.

? – List the PAN commands.

Modem equipped terminals or computers can send a message to a PAN system by calling the PAN and sending a special control character before the message text. A second control character after the text causes PAN to store the message and hang up the phone.

PAN has been implemented for the Commodore PET and TNW488 modem; we are working on a version for the Apple II and Hayes Micromodem II. We plan versions for the TRS-80, S-100 systems, etc.

To receive the current PAN package (a cassette tape of the PET version of PAN and a user's manual) sign and send the enclosed PAN license agreement with a check for $12 to: PCNET Project, People's Computer Company, 1263 El Camino Real, Box E, Menlo Park, CA 94025. Please also include the size of your PET's memory – 8K, 16K, etc.

We're distributing PAN under a license agreement for a number of reasons:

- We want to be sure that all versions of PAN will be able to talk to one another.
- We want to provide a central organization for the roster of PAN users, and for the distribution of new PAN versions.
- We want to protect the rights of TNW Corporation, our associate in the PAN development program.

APPENDIX C
PAN Lower Level Protocol

1. The originating PAN dials the number of the answering PAN.
2. When the answering PAN detects ring, it goes off the hook and sends carrier.
3. When the originating PAN detects carrier it sends carrier.
4. When the answering PAN detects carrier it sends the string "@@@ANN" where ANN consists of a single alpha character followed by two numerics. This string is the version number of the answering PAN; as of 6 Feb 80 it is P09.
5. When the originating PAN receives one or "@" characters it sends 5 CTRL-S characters.
6. When the answering PAN receives one or more CTRL-S characters it prepares to accept subsequent received characters as a PAN message. If no CTRL-S characters are received the answering PAN times out and goes on hook.
7. The originating PAN sends the message text. The first line of this text (between the first character and the first carriage return) contains the phone number of the originating PAN in the form (AAA)NNN-NNNN. By convention this first line also contains the sender's initials and the originate PAN local time of transmission in 24 hour format; HHMM SS where HH ranges from 00 to 23.
8. The originating PAN sends 4 CTRL-R characters to signal end of message; it then goes on hook.
9. When the answering PAN receives one or more CTRL-R characters it goes on hook and places the received character string in the message buffer. If the answering PAN loses carrier at any time during message reception it goes on hook.

APPENDIX D
Packet Radio in Montreal

This report about packet radio was prepared by Canadian Amateur Radio Federation, Inc. To be put on their mailing list, which means you will receive their next four Packet Radio Newsletters, send $5 to Art Blick at CARF, P.O. Box 356, Kingston, Ontario K7L 4W2. The money is to defray the costs of reproduction.

Since the Montreal group was the first into packet it is not unusual that this group is the furthest advanced in this technology. A system has been fully operational since August of 1979, and has been continually updated since then. There are currently six stations on the net, with more working on modems and computers right now.

The net operates on 223.5 MHZ currently, but a move to 222.3 is expected as soon as all of the stations have their crystals. Audio frequency shift keying is used, at 2400 bits per second. The mark tone is 2400 HZ and space is 4400 HZ. The modem is based on the Exar 2206. 2211 chip set, with a credit to Ted Balesta VE3CAF who first suggested the design. Having experimented with the design for almost a year now, it is capable of error-free transmission with S/N ratios of 15 DB or so. Its perfection is due to the efforts of Jacques Orsali VE2EHP who "massaged" the design to its present performance levels.

The radio is typically the Midland 13-509, running ten watts. It was chosen on the virtue of simplicity and price. The modificiations to it are simple and can be done in about thirty minutes or so. The interface to the radio was deliberately kept as simple as possible in anticipation of future users who may be either unwilling or unable to make extensive mods to the radio. Since just recently, Ted VE3CAF has joined the Montreal team, and we expect that he will make substantial improvements to the interface.

The local net is being used for man-to-man communications mostly, but lately the emphasis has switched to machine-to-machine transfers of programs, text files and the like. One user is currently writing a multi-user real time war game which will be implemented on the net soon. One of the unresolved questions is what major use the net will be put to once fully implemented. Since this will depend on the character of the users, it will stay a question for some time yet.

The network includes a store and forward repeater as part of the system, and this repeater is currently functioning. It was demonstrated at the RSO convention in the fall of 1979. It receives packets which are flagged as packets for repeating, and rebroadcasts them a few miliseconds later. It serves the same purpose as a conventional 2 meter job, i.e., it extends the range of a station using it. Since there are no cavities (it is a simplex device) it does not suffer from the usual problems of desense, and split antennas. Naturally, it is very efficient and as we say in the business, "it talks far."

The repeater operates on the same frequency as the rest of the net and only repeats pacs which are flagged to it. In this way we find two stations often using the repeater while two others who are close enough use simplex, all on the same frequency. In addition to its conventional roll, Digptr as it is called, offers other services to the users. By sending a pac to it (as opposed to through it) users can get a "Broadcast Message" which provides important information of immediate interest, system shutdowns, protocol changes and the like. The message can be changed remotely by the system operators, and is password protected. Users can also use the Digptr to check out their system. Using the Test facility, a packet sent to Digptr, will be transmitted back to the station the same way it was received, that is, with or without errors. By accessing the status file of the repeater, users can see the number of pacs, and acks which have gone through Digptr, as well as those which were "direct" and bypassed it. System operators can use this function for statistics on usage, and throughput.

The latest and perhaps most exciting development is that we now have a single board computer/terminal working. It is based on the Motorola Chroma II kit, sometimes called the TV BUG. This low cost ($300 or so) device will implement the full packet protocol in PROM, and requires only a keyboard, a TV set and a compatible modem to run. This means that the cost of a complete packet set-up is reduced by a factor of about ten. We figure about $500 including the radio would be the maximum cost for someone who had nothing to start with. For those people who have micro computers already, TV bug will

(continued on page 58)

MCALL-C: A Communications Protocol for Personal Computers

BY TIM J. PUGH, JR.
Micro-Call Services
9655-M Homestead Court
Laurel, MD 20810

In the protocol description which follows, I will identify alternative approaches that were considered in arriving at the present protocol. But before getting into the protocol itself I would like to try to clear up a misunderstanding that may exist concerning telecommunications protocols. The type of protocol which I am describing has no bearing on the information content of the data. The protocol acts simply as an envelope or container of data. The data is inserted into the envelope at the originating (transmitting) end and automatically removed at the receiving end. Thus, by the time the data is placed on disk, it looks exactly the same as it did on the disk of the originator.

Perhaps the first question to answer is "why have a protocol?" The principal reason for having one is to insure, as far as possible, that no data is lost or garbled in the transmission. One can, and I often do, transmit and/or receive data files with a large Time Sharing Computer (TSC). These TSCs are typically very uncooperative beasts which output the data to you in "their own good time," usually as fast as it can be sent. They are not concerned about whether you had noise on the line or whether you had time to write the last record to disk. That's your problem. Thus, if you have a system which pays rapt attention to the incoming data and if the weather is good and you have a high quality modem, then you have an excellent chance of receiving error-free data.

However, one cannot depend on all the above stated conditions being met, especially when two personal computers are involved. Such a transfer will frequently be "long distance." Each additional TELCO switching center involved in the "call" reduces the signal-to-noise ratio and often there is line "echo" distortion present. Thus, reliable data transfer becomes questionable without an intervening mechanism. An appropriate communications protocol supplies this mechanism.

Serial Versus Blocked Data Format

The simplest way to transfer a file is the way it is done with a TSC. The file is simply transferred serially to completion. It is assumed the receiving file has already been created and that the TSC is capable of recording the data at the maximum BAUD rate; e.g., 300 BAUD. Such an approach can also be used for transfers between two Personal Computers (PCs) provided the operation is properly coordinated and synchronized. However, it is preferable to have the two computers do the necessary synchronization and coordination since it is then both more convenient for the users and less error prone. We have thus posited a protocol whereby the file is automatically opened at the Transmitting (TX) end and created at the receiving (RX) end and might look something like this:

```
--> @SOH <filename> <typ>
<-- @ACK
--> <file data>
```

The directed arrows indicate the direction of data transfer where "-->" indicates data from the TX to the RX. The "@" signifies ASCII and thus @SOH indicates the ASCII Start Of Header control character. Similarly, "@ACK" signifies ASCII ACKnowledge. Assuming that the originating (TX) operator has previously specified the name of the file to be transferred; i.e., <filename><type> and further assuming that communication has been established between the two parties, then the transfer is initiated by the TX end issuing an @SOH followed by the name of the file. Since an indeterminate amount of time is required for the RX system to create the corresponding file, and ACKnowledgement is required before the data transfer can begin. Why an indeterminate amount of time to open a file? Partly because there is such a variety of floppy disks and controllers capable of running under CP/M (both 8" and 5 ¼"). Since a "handshake" character, @ACK, was required to signify "file created OK"; why not include the option of sending @NAK (Negative AcKnowledge) to signify some problem in opening the file? Let's further assume that if @NAK is received by the TX end, the "@SOH <filename><typ>" will be repeated.

Since we have provided for retransmission (reTX) of the Header Record, the next question is: how can we discontinue the retransmission, assuming the problem persists over several reTXs? This introduces a third control character, @CAN (ASCII CANcell). Thus our "handshake" in response to the Header Record can take one of three forms. This situation is handled nicely by the BACKUS-NAUR notation which follows:

```
<ack. form> : : = @ACK | @NAK | @CAN
```

which states that the acknowledge form is defined equal to either @ACK or @NAK or @CAN.

Copyright © 1980 by Tim J. Pugh, Jr.

We next look at the <file data>. The most conventional method for "recording" received data is to first buffer it in memory and subsequently write it to disk. This allows the RX system to handle high BAUD rates, e.g., 1200 BAUD, without data loss. However, if large files are transferred, buffer overflow might result. Thus, it is desirable to allow the buffer to be "dumped" to disk at appropriate times. A simple solution to this problem is to break the <file data> into small blocks followed by an <ack. form>. Thus, if the data is broken into equal blocks of 128 characters the protocol would look something like this:

```
-->    @SOH <filename> <typ>
<--    @ACK
-->    <data.blk>
<--    @ACK
-->    <data.blk>
<--    @ACK
       ...
       ...
-->    <data.blk>
<--    @ACK
```

The @ACK following each <data.blk> tells the TX system that the buffer write operation is complete and the next block can be sent. The above considerations argue strongly for a BLOCKED rather than a SERIAL data format for our protocol.

Data Integrity Considerations

The blocked protocol specified thus far provides for error recovery (reTX) in the event the <filename><typ> was garbled; e.g., noise caused a given character to be converted to a characger which is not legal when specifying file names. It also provided time for data blocks to be transferred from memory to disk. However, no provision has been made to insure that data integrity is maintained.

The most commonly-employed technique for handling this problem is to use either a Cyclic Redundancy Check (CRC) or a Checksum. The CRC usually employs a polynomial equation for its realization. The term checksum, as used here, simply means a sum of all the characters in a given data block. Since each character in the ASCII set can be represented by seven data bits, this simply means the accumulated total obtained by adding each character in turn, to the previous partial sum. I have chosen the checksum because of its simplicity and ease of implementation.

In implementing the checksum, two choices must be made: (1) The block size over which the checksum is to be taken; (2) The number of bits to be allocated to the checksum. The most convenient block size appears to be that associated with one physical record on disk; i.e., a disk sector of 128 bytes. Having chosen the first parameter we find that the largest value the checksum could take on occurs if the ASCII rubout character (@DEL = 07FH) is repreated 128 times. Since 07FH is equivalent to 127 decimal, the worst case checksum would be:

```
SUM = 128 * 127 = 16256
```

This value falls between 2^15 and 2^16. Thus, a 16 bit checksum would insure a unique sum for each unique data block. I am using the term "unique" rather loosely here. Two data blocks consisting of the same characters but in different order are not considered unique, by this definition.

At least one popular "serial I/O" card employs a UART whose output is fixed at 7 data bits. The Cromemco TU-ART employs the Texas Instruments TMS5501 device to support serial I/O. The serial output of this device is fixed at 7 data bits with a parity bit of zero. Thus, while it would be most efficient to transfer the 16 bit checksum as two eight-bit characters, this approach would exclude systems employing the TU-ART card to drive an acoustic coupler. For this reason, an ASCII HEX encoding of the checksum was employed. This results in the 16 bit value being broken into for "nibbles" of 4 bits each. The ASCII representation of each nibble ranges from ASCII 0 through 9 and ASCII A through F. There remains the question of ordering the output value; i.e., what sequence should be used in transferring the nibbles. I somewhat arbitrarily chose the most significant nibble first and least significant nibble last.

We now have a protocol where each data block, including the header block, has a checksum associated with it. As each character is transferred, both the TX and RX systems accumulate individual checksums. The TX system transfers its final value to the RX, who tests it for equality with its own value. If the values agree, an @ACK is returned, otherwise a @NAK is returned and the data block is retransmitted. How many times should reTX be allowed to occur on the same data block before cancelling (@CAN) the transfer? I have somewhat arbitrarily chosen the value of 5. Remember, it takes quite a bit of time to repeat a data block 5 times. Five repeats plus the original at 300 BAUD requires approximately:

```
retry time = 6 * (128 + 4) /30 Sec. = 26.4 Sec.
```

Also remember we are not dealing with "crud" on a disk (which uses 10 retries) but with noise on the line. If the line noise, or perhaps some system problem, causes a reTX in excess of 5 times on the same data block, brother, you have a problem! Note further that many more than 5 reTXs are allowed provided no more than 5 occur on a given data block.

While we seem to have solved the data integrity question on a data block basis, we still need some way of assuring ourselves that each and every block was received properly and that some reTXed block didn't get recorded twice by mistake. One approach is to include a block or frame count as part of the block transfer. Since the files transferred might be quite large, a 16 bit block count might be needed. Given the 7 data bit limit mentioned above, this would require an additional 4 characters per block. I chose a different approach which does not add to the size of the data block. I have found that if a cummulative total of reTXs is kept by both TX and RX systems, then the equality of these values virtually assures the transfer was error free. Thus, the TX system sends his retry count to RX system as a part of the End of TeXt (@ETX) record. This record immediately follows the last <data.blk> record. If the counts agree, the RX system sends @ACK. Else, a @NAK is sent. Both systems then display on their respective consoles that the transfer was either successful or unsuccessful along with the total number of reTXs .

We now have a protocol that looks something like the following:

```
-->  @SOH FILENAMETYP          SUM
<--  @ACK
-->  AAAAAAAAAAAAAAAA          SUM
<--  @ACK
-->  BBBBBBBBBBBBBBB           SUM
     ...
     ...
-->  JJJJJJJJJJJJJJJ           SUM
```

```
<-- @NAK                          (Retransmission Request)
--> JJJJJJJJJJJJJJJ        SUM
<-- @ACK
--> KKKKKKKKKKKKKKKK      SUM
<-- @ACK
      ...
      ...
--> ZZZZZZZZZZZZZZZZ      SUM
<-- @ACK
--> @ETX CNT
<-- @ACK
```

The "SUM" associated with each data block represents the four ASCII HEX characters associated with the checksum. The "CNT" at the end represents a single character value interpreted as a binary retransmission count value. Thus, it represents the count modulo 128 since the number of data bits in the protocol is set to 7 data bits.

Variable Block Size and Data Parity Check

The above protocol is about as far as we can go if limited to 7 data bits and no parity. However, since most UARTS/USARTS provide for both software control of the number of data bits (5 thru 8) and additional control over parity selection (even or odd) coupled with parity enable/disable, valuable additions can be made to the protocol.

Let's first consider variable block size. If we limit the block size to one sector (128 characters) then we must access and write the disk every 4 seconds (approx.) at 300 BAUD. The read/write head on most drives unloads within this interval. Thus, the head is being loaded each time a write operation is performed. This takes on the order of 36 Msec or about one character time at 300 BAUD (33.3 Msec per character). In addition, the proper sector must be found – assuming the head is already positioned at the appropriate track. Since it takes 166.6 Msec for one rotation of the disk, the average "latency" is one half that value; i.e., 83.3 Msec. Thus, each time we write the disk costs us, on average,

write time = 83.3 + 35 Msec = 118.3 Msec.

This is approximately 4 character times at 300 BAUD and 16 character times at 1200 BAUD. The worst case time would be approximately 211.6 Msec or approximately 6 character times at 300BAUD (includes 10 Msec track stepping time). These times apply to the Stugart 800 drive. The times should be fairly representative of all floppy disk drives.

Given the above statistics, it seems desirable to increase the block size and thereby reduce the number of times a disk write operation is performed. In arriving at an optimum block size (if one exists) we must consider the impact of increased block size on both buffer memory size and time lost in retransmitting a larger data block. I have no firm number in mind but since CP/M allocates disk space in 8 sector blocks, a value of 1024 = 8 * 128 seems a good starting point. This calls for a 16 bit block size value which can be sent as two 8 bit bytes interpreted as binary values rather than ASCII characters. To be consistent with the checksum ordering the most significant byte is transmitted first.

The next protocol enhancement to be introduced provides for a parity error test. If a data parity error is found, there is no need to continue the block transfer. Instead, the block should be immediately retransmitted. This further facilitates the introduction of larger block sizes. To accommodate the retransmission request due to parity error, two additions to our protocol are needed. First, we must add a resync (@SYN) request to our <ack. form>. In response to the receipt of an @SYN from the RX device, the TX device will discontinue the block transfer and begin a retransmission of the same block. Second, to assure proper synchronization, each data block should begin with a Separator (@RS) for this task. Thus, if the RX device detects an @RS following his transmission of an @SYN, he knows the block retransmission request has been received by the TX device and that this is the beginning of that block.

Finally, it is desirable to be able to identify a block with more or less characters than specified by the block count. This can be done by inserting a special character between the end of data and the beginning of the checksum. I have selected the ASCII Unit Separator (@US) for this task. If there is such a disagreement, a @SYN is immediately sent by the RX to the TX, causing a retransmission. This completes the protocol specification and leaves us with a typical data block that looks something like,

```
--> @RS N XXXXXXXXXXXXXXXXXXXX @US SUM
<-- @ACK
```

where N denotes the block count and, as before, SUM denotes the checksum.

I have given this protocol the name "MCALL-C", which stands for Micro proto "CALL"–Clear. The clear means that the data is not encoded, as is the case for the PCNET protocol. In PCNET protocol, the data is encoded in a radix 41 code rather than pure ASCII. This encoding allows binary information to be transferred while using only 41 "printable" characters to encode the data.

The complete MCALL–C definition both in BACKUS-NAUR form and in a more graphic form (section (3)) is shown in Table 1. The definition is from the point of view of the originator or (TX) system.

```
(1) DATA TRANSMITTED (@XXX SIGNIFIES AN ASCII CONTROL CHAR)

      <FILE>   ::= <F.HDR> <F.TEX>
      <F.HDR>  ::= @SOH <HEADER> @US <CHKSUM>
      <HEADER>::= <8 CHAR FILENAME> <3 CHAR FILETYPE>
      <CHKSUM>::= <16 BIT CHECKSUM ENCODED IN ASCII HEX
                   MS-NIBBLE FIRST, LS-NIBBLE LAST>
      <F.TEX>  ::= <TEXT> @ETX <ERRTTL>
      <TEXT>   ::= <BLOCK> | <BLOCK> <TEXT>
      <BLOCK>  ::= @RS <CHRCNT> <STRING> @US <CHKSUM>
      <CHRCNT>::= <NUMBER (LESS ONE)  OF CHARACTERS IN <STRING>>
      <STRING>::= <PRINTABLE ASCII CHAR STRING OF FILE DATA>
      <ERRTTL>::= <TOTAL ERROR COUNT MOD 128>

(2) DATA RECEIVED (HANDSHAKING PROTOCOL)

      <ACK.FORM>::= <@ACK> | <@NAK> | <@SYN> | <@CAN>
      <@ACK>   ::= @ACK                 (SUCESSFUL TRANSFER OF DATA)
      <@NAK>   ::= @NAK                 (RETRANSMISSION REQUEST)
      <@SYN>   ::= @SYN                 (RESYNC REQUEST)
      <@CAN>   ::= @CAN                 (ABORT TRANSMISSION)

      <HDR.ACK>::= <F.HDR> <ACK.FORM>
      <BLK.ACK>::= <BLOCK> <ACK.FORM>
      <EOF.ACK>::= @ETX <ERRTTL> <ACK.FORM>   (FILE CLOSED ACK)

(3) SAMPLE FILE TRANSMISSION

-->   @SOH FILENAMETYP @US SUM              (HEADER BLOCK)
<--   @ACK
-->   @RS N AAAAA...AAAAA @US SUM           (FIRST DATA BLOCK)
<--   @ACK
-->   @RS N BBBBB...BBBBB @US SUM           (SECOND DATA BLOCK)
<--   @ACK
              .....
              .....
              .....
-->   @RS N VVVVV...VVVVV @US SUM
<--   @NAK                                  (RETRANSMISSION REQ)
-->   @RS N VVVVV...VVVVV @US SUM
<--   @ACK
              .....
              .....
              .....
-->   @RS N YYYYY...YYYYY @US SUM
<--   @ACK
-->   @RS N ZZZZZ...ZZZZZ @US SUM
<--   @ACK
-->   @ETX M                               (END OF FILE)
<--   @ACK
```

Having specified the protocol, we must now resolve the problem presented by the existance of UARTS whose number of data bits is limited to seven and whose operation does not support parity generation and testing. The best approach might be to have two similar but different protocols. One to accommodate the TMS5501 and another to take advantage of the 8 data bits plus parity. Let's call the 7 bit version MCALL-C and the 8 bit version, MCALL-BC. The "B" related to the Big data block size. The two protocols are identical except in the way the block size and the checksum are represented and the use of parity testing in one but not the other. By limiting the differences to these three parameters, both protocols can be accommodated by the same program. A "flag" could be set in response to operator input to specify which processing path was selected.

For MCALL-C we define the block size and the checksum as follows:

<CHARCNT> : : = <NUMBER (LESS ONE) OF CHARACTERS IN <STRING> IN THE RANGE 0 TO 127>
<CHKSUM> : : = <16 BIT CHECKSUM ENCODED IN ASCII HEX WITH MS-NIBBLE FIRST>

For MCALL-BC we define these two parameters as follows:

<CHARCNT> : : = <NUMBER (LESS ONE) OF CHARACTERS IN <STRING> IN THE RANGE 0 TO 65535 with MS-BYTE FIRST>
<CHKSUM> : : = <16 BIT CHECKSUM WITH MS-BYTE FIRST>

As to the effectiveness of the protocol, I have implemented the MCALL-C protocol as part of a program I have written called "MCALL". The program fixes the UART data format at 7 data bits with even parity, although no parity test is currently performed. This protocol has been tested on a vaiety of I/O cars for the S-100 BUS and includes the Cromemco TU-ART. I find the protocol very effective in file transfers and have never had an unsuccessful transfer under normal circumstances. By normal I mean under both good and foul weather conditions with total reTX counts ranging up to 10 or more per message. I generally get error-free performance even when I tap on the telephone handset with a pencil to introduce noise. Occasionally, however, using the "bang approach" I have caused the system to "bomb" or record the same block twice. However, the operator is alerted to the latter case via an "unsuccessful file transfer" message at the end. This is where the comparison of reTX counts pays off.

The current version of MCALL (V1.6) supports only MCALL-C. I hope to add MCALL-BC in the future, along with other protocols and program enhancements. Further information relating to MCALL can be obtained by contacting the author whose address is given at the beginning of this article.

UPDATE NOTICE
February 22, 1980

The following changes to MCALL V1.5 will remove a potential problem which exists when sending a file to a remote TSC at 300 BAUD. When a TSC becomes "loaded" and its response time deteriorates, it is possible to lose some of the file data being sent. It is usually the case that, while a TSC may not be capable of receiving continuous data at 300 BAUD, it is capable of handling this rate for the duration of a single line. The proposed change to the file TX routine causes the transfer to pause at the end of each line and await acknowledgement in the form of a line-feed. Further, the remote device is allowed to follow this @LF with additional characters (e.g., a prompt character) before transmission of the next line commences.

The proposed change results in a new version number, MCALL V1.6. The following directions should be followed in order to implement the change:

1. Change the second line of the source file to reflect the new version number, V1.6. Search for the unique string "MSG1:" and change the second line of that message to reflect the new version number.
2. Either perform a negative search or revert to the start of the source file and search for the unique string "KONSW:". On the next line, insert the following storage declaration:

```
IDLCNT: DS   2          ;IDLE LINE COUNTER
```

3. Search for the unique string "TF3:" and then advance 5 lines. At that point, replace the statement "CALL SHOW" with the following:

```
MOV    C, A
BIOS$  CONOUT  ;CRT <- CHAR
```

This will cause the CRT to display only the printable characters during a TX file operation, rather than converting non-printing control characters to their printing equivalent. While this change has nothing to do with the stated problem, it seems a desirable change.

4. Search for the unique string "TFEOL:" and advance 6 lines. Kill this and the next 5 lines; i.e., the LDA, ORA, JNZ, MVI, CALL, and JMP instructions. Now, advance to the end of the subroutine and kill the last instruction; i.e., the "JMP TXBYTE" instruction. In place of this last instruction, the following instructions should be appended to the subroutine:

```
        ANI     07FH
        CPI     0LF
        JNZ     TF4       ;LINE ACKNOWLEDGED?
TF4B:   LXI     H,0       ;   YES
        SHLD    IDLCNT    ;IDLCNT <- 0
;--WAIT FOR LINE TO GO IDLE
TF5:    LHLD    IDLCNT
        INX     H
        SHLD    IDLCNT    ;++IDLCNT
        MOV     A,H
        ANI     03H       ;TEST ONLY 10 LSBS OF COUNT
        ORA     L
        JZ      TXBYTE    ;IS LINE IDLE?
        CALL    MSTAT     ;   MAYBE NOT
        ANI     RDA
        JZ      TF5A      ;RX DATA AVAILABLE?
        IN      RXDATA    ;   YES
        ANI     07FH
        MOV     C,A
        BIOS$   CONOUT    ;CRT <- CHAR
        LDA     CHCNT
        INR     A
        STA     CHCNT     ;++CHCNT
        JMP     TF4B      ;RESET IDLE COUNT AND WAIT AGAIN
TF5A:   BIOS$   CONSTS
        JZ      TF5       ;CONSOLE DATA AVAILABLE
        JMP     TFEND     ;   YES, TERMINATE FILE TX
```

That's it. Sorry this change was not included in the V1.5 update but an earlier attempt to resolve this problem in assembly language failed. It was not until I recently re-coded the program in the language "C" that I was able to resolve it. It's fairly trivial, once you see the solution. Oh well, that's the way it goes.

For those of you who want the update but would prefer to receive it on disk, I will apply the same "deal" as before. Return the old disk and receive an update for $5. Otherwise, send $10. and I will ship you the update on a new disk. As far as I know, this is the last of the significant changes that the program requires. However, I will keep you informed of any "bells & whistles" that may be added in the future.

UPDATE NOTICE

April 8, 1980

This update is principally aimed at the "ESC F" function. This update will change the version number from V1.6 to V1.7. In V1.6, the filename and filetype were specified on separate lines and lower case characters were not converted to upper case, as required by CP/M (sorry 'bout that). Also, it was not possible to specify a file as residing on other than the "logged in" disk. This update corrects these problems.

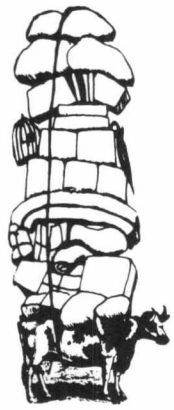
The first thing that must be changed is the version number appearing on the first "page" of the source. It also appears following the unique string "MSG1:". Next, search for the unique string "MSG20:" and replace this message with the following message:

```
MSG20:  DB  CR, LF, 'ENTER <FILENAME>.<TYP>-DISK
            SPEC (e.g. B:) '
        DB  'OPTIONAL', CR, LF, LF, '$'
```

Enclosed is a two page listing of the new subroutines required to implement the change. These two routines replace the single previous routine named "FILE." Thus, it will be necessary to search for the unique string "FILE:" and replace the existing routine with the two attached routines. Note that the old and new FILE routines are identical through the three lines following the line labeled "FILE2: ".

One last minor change fills the final sector of a "recorded" disk file with 01AH. This assures that no "garbage" is present in the last record of a recorded file. Only one line of code need be changed. To do so, search for the unique string "BB2D3:". Advance to the next line and change the symbolic jump address from "BB2D5" to "BB2D4". That's it. Simply change the '5' to a '4' and reassemble. Enjoy!

```
;++ ****************************************************************
;
;$$     FILE -- File Control Block setup
;
;               Provide for operator selection of a new file by
;               prompting him for his selection and then set up
;               the associated F.C.B.
;
;-- ****************************************************************

FILE:   MVI     B,33
        MVI     A,0
        LXI     H,FCB
;--ZERO FILL THE ENTIRE FCB
FILE1:  MOV     M,A
        INX     H
        DCR     B
        JNZ     FILE1   ;FCB ZEROED?
;--BLANK FILL <FILENAME> & <FILETYPE>
        MVI     B,11    ;   YES
        MVI     A,' '
        LXI     H,FCBFN
FILE2:  MOV     M,A
        INX     H
        DCR     B
        JNZ     FILE2   ;BLANK FILL COMPLETE?
;--PROCESS <FILENAME>.<TYPE>
        PRINT$  MSG20,$ ;REQUEST <FILENAME> & <FILETYPE> FROM OPERATOR
        MVI     C,RDCBUF
        LXI     D,CBUFF
        CALL    FDOS    ;GET <FILENAME>.<TYP>
;--PROCESS OPTINAL DISK SPECIFICATION
        LDA     CBTXT+1
        CPI     ':'
        JNZ     FILE3   ;DISK SPECIFIED?
        LDA     CBTXT   ;   YES
        ANI     3
        STA     FCBDN   ;FCBDN <- DISK_SELECT
        LDA     CBCNT
        DCR     A
        DCR     A       ;CBCNT = CBCNT - 2
        LXI     H,CBTXT+2
        JMP     FILE4
;--MOVE <FILENAME> & <TYPE> TO FCB
FILE3:  LDA     CBCNT   ;A.REG <- CHAR_CNT
        LXI     H,CBTXT ;H,L.REG <- SOURCE_ADDR
FILE4:  LXI     D,FCBFN ;D,E.REG <- DEST_ADDR
        CPI     13
        JC      FILE5   ;CHAR_CNT > 12?
        MVI     A,12    ;   YES
FILE5:  CALL    MOVCAPS ;FCB <- FILE_NAME,TYP
        CALL    CRLF
        RET

;++ ****************************************************************
;
;$$     MOVCAPS -- Move a block of characters from source to destination
;               and convert to upper case (CAPS).  Also, screen
;               out '.' if found.
;
;           /   A,REG   = count
;               H,L.REG = source_address
;               D,E.REG = dest_address
;
;-- ****************************************************************

MOVCAPS: MOV    B,A
        ORA     A
MOVC1:  RZ              ;MOVE COMPLETED?
        MOV     A,M     ;   NO, GET A CHAR
        CPI     '.'
        JNZ     MOVC2   ;CHAR = '.'?
        INX     H       ;   YES
        LXI     D,FCBFT ;D,E.REG <- FCBFT
        DCR     B
        JMP     MOVC1   ;GET THE NEXT ONE
MOVC2:  CPI     061H
        JC      MOVC3   ;LOWER CASE CHAR?
        ANI     05FH    ;   YES, CONVERT TO UPPER CASE
MOVC3:  STAX    D       ;MOVE TO DESTINATION
        INX     H
        INX     D
        DCR     B
        JMP     MOVC1
```



THE ELECTRIC PHONE BOOK

A Directory of 144 Computerized Bulletin Board Systems

A computerized bulletin board works just like an ordinary bulletin board system except that instead of paper and thumbtacks it uses a terminal, a computer, and the dial-up telephone network. It's a place to leave messages for everyone or for some particular person who, you know, browses the bulletin board occasionally.

The list below was developed from several sources including the Peripheral People in Mercer Island, Washington and the People's Message System in Santee, California. It is being maintained by PCC's PCNET project, our effort to bring computers and telecommunications into the hands of everyone. While this is the most complete listing we have as of this writing, we would appreciate additions and corrections. Send them to PCNET, PCC, P.O. Box E, Menlo Park, CA 94025 or leave them on the PCNET/PCC Bulletin Board System for Dave Caulkins, (415) 948-1474.

All the bulletin board systems listed here can be accessed by telephone using a 300-baud ASCII terminal and a Bell 103 modem. Most use carriage-return as a speed recognition character, after which they are self-teaching. All are free to anyone who calls, unlike the Arpanet, which is restricted, and The Source and MicroNet which cost money. The list has been sorted by area code; consult your local telephone directory for geographical correspondence.

We are compiling a mailing list of people interested in computers and telecommunication. If you would like to be on it, send us a note at the address above. If you are interested in PAN, PCNET's computer mail system for the PET, send us a stamped, self-addressed envelope and we'll send you back information.

(201) 457-0893
(201) 835-7228
(201) 874-6833
(201) 891-7441
(201) 968-1074
(202) 337-4694
(205) 945-1489
(206) 244-5438
(206) 482-5134
(206) 524-0203
(206) 723-3282
(206) 937-0444
(209) 638-6392
(212) 448-6576
(213) 276-4276
(213) 316-5706
(213) 329-3715
(213) 340-0135
(213) 349-5728
(213) 360-6332
(213) 394-1505
(213) 395-1592
(213) 396-3905
(213) 424-3506
(213) 428-4718
(213) 459-3177
(213) 566-8035
(213) 657-8803
(213) 675-8803
(213) 787-4004
(213) 795-3788
(213) 799-1632
(213) 799-6514
(213) 826-0325
(213) 828-3400
(213) 843-5390
(214) 288-4859
(214) 634-2668
(214) 634-2775
(214) 641-8759
(216) 745-7855
(305) 566-0805
(305) 689-3234
(305) 772-4444
(305) 821-7401
(305) 989-9647
(312) 255-6489
(312) 269-8083
(312) 337-6631
(312) 420-7995
(312) 528-7141
(312) 622-9609
(312) 767-0202
(312) 964-7768
(313) 288-0335
(313) 465-9531
(313) 477-4471
(313) 484-0732
(313) 569-2063
(314) 838-7784
(316) 746-2078
(319) 353-6528
(319) 557-9618
(404) 394-4220
(404) 733-3461
(404) 790-8614
(404) 939-1520
(404) 939-8429
(404) 953-0723
(405) 528-8009
(408) 241-1956
(408) 263-0248
(415) 348-2139
(415) 348-2396
(415) 493-7691
(415) 661-0705
(415) 792-8406
(415) 948-1474
(417) 862-7852
(451) 948-1474
(502) 245-8288
(503) 646-5510
(512) 657-0779
(513) 874-2283
(516) 938-9043
(602) 866-0258
(602) 886-0258
(602) 955-1486
(602) 957-4428
(604) 687-2640
(607) 754-5571
(609) 983-5970
(612) 929-8966
(614) 272-2759
(614) 649-7097
(615) 254-9193
(617) 354-4682
(617) 431-1699
(617) 649-7097
(617) 864-3819
(617) 897-0346
(617) 963-8310
(702) 873-9491
(703) 281-2125
(703) 734-1387
(703) 750-0930
(703) 893-9474
(703) 978-7561
(713) 693-8080
(713) 977-7019
(714) 449-5689
(714) 463-0461
(714) 526-3687
(714) 537-7913
(714) 565-0961
(714) 571-5550
(714) 582-9557
(714) 582-9957
(714) 730-1206
(714) 739-0711
(714) 751-1422
(714) 772-8868
(714) 898-1984
(714) 962-7979
(801) 753-6800
(803) 270-5372
(803) 270-5392
(803) 279-5392
(803) 771-0922
(805) 484-9904
(813) 223-7688
(816) 861-7040
(816) 931-3135
(817) 855-3916
(817) 855-3918
(817) 923-0009
(901) 276-8196
(901) 362-2222
(901) 761-4743
(904) 243-1257
(904) 243-8565
(913) 764-1520
(913) 782-5115
(915) 584-5393

ON modems

BY WILLIAM SMITH
2325½ Howe Street
Berkeley, CA 94705
Micronet ID: 70110,243
Source No: TCD650

Like all new areas that emerge in this world of microcomputers, the area of phone line connection is fraught with unclear terminology and generalities. I spent much energy sorting things out in order to set myself up 'on line' and would like to share some of what I have learned with you. To begin, some terms must be defined:

MODEM a general term that can refer to any of several different aspects of the physical device which couples a computer to the phone lines.

ACOUSTIC COUPLER a specific type of modem that makes an acoustic connection between the computer and the phone lines by setting the phone handset into a cradle which is connected to the computer.

DIRECT COUPLER another specific type of modem that makes a direct connection between the computer and the phone lines; everything stays in the form of electric signals; there are no intermediary acoustic stages.

MODULATOR/DEMODULATOR the circuitry responsible for encoding and decoding information represented by logic states (i.e., high and low) to and from information represented by vibration rates (i.e., high frequency and low frequency).

FREQUENCY SHIFT KEYED (FSK) jargon for the frequency coding technique typically used in modems (and many cassette interfaces); a sine wave is the carrier of the binary information and the data is stored by shifting the frequency to one value for a high and to another value for a low.

BANDWIDTH a continuous region or band of frequencies defined by the difference between two specific frequencies which denote its upper and lower limit, e.g., the bandwidth of FM radio is approximately 20 Mhz, from 88 Mhz to 108Mhz.

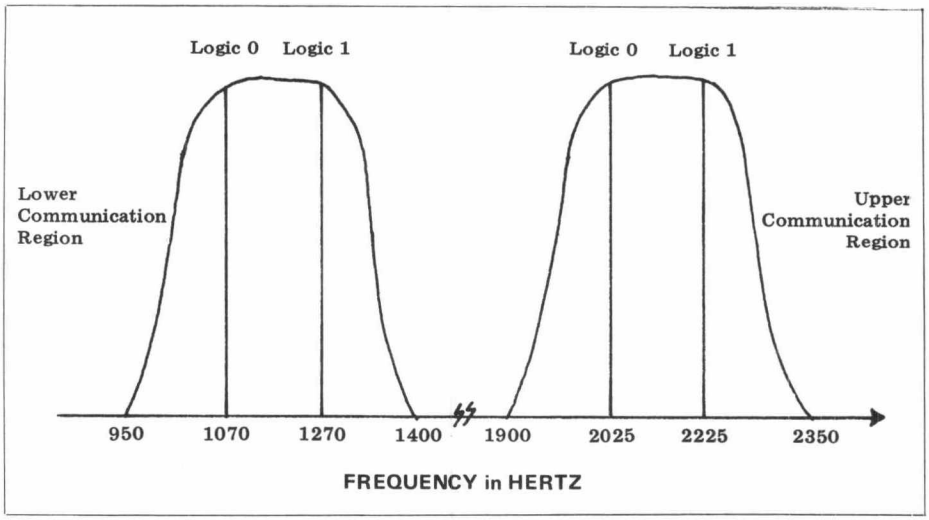
A data communcations link-up is a two way connection. Both parties can send and receive information. This is facilitated by dividing the phone line's available bandwidth into two separate bands or channels. Each one of these corresponds to one of the two different directions in which data is being sent. There is a 450 hz band in the 1K region and another 450 hz band in the 2K region. Within each of these 450 hz regions we need to distinguish to separate frequencies, corresponding to the two logic states that must be represented. The logic states are coded in FSK mode. Thus each state requires only a few hertz bandwidth. The separation between the logic one frequency and the logic zero frequency in both the upper and lower regions is 200 hz. Refer to the table for a graphic representation of all this information. The exact figures for the upper and lower bands representing the two communication directions may vary between manufacturers according to filter design but the four logic levels are precisely defined industry standards. This is necessary if machine independent communication is going to take place.

If one of the two communication channels, say the upper one, were reserved for the send direction of communication, and the other were reserved for the receive direction, then consider the problems that would ensue. Both parties would be sending their messages on the send channel and expecting to hear from the other on the receive channel where there would be silence! Clearly there has to be a more definitive agreement about who is the sender and who is the receiver.

The solution lies in defining a new term: Mode. When two parties are in communication, the one that placed the call is considered to be in Originate Mode and the other is in Receive Mode. In Originate Mode the upper communication region is your receive band and the lower region is your transmit band. In Answer Mode the two are reversed. Thus if A calls B then A is in Originate Mode and will transmit on the lower region, while B will be in Answer Mode and therefore receiving A's transmission on the lower band. On the upper band B will be transmitting and A will be receiving.

The major difference between an acoustic coupler and a direct coupler computer phone link up is the element of personal presence. In the acoustic coupled situation someone must be there to physically dial the phone and set the handset on the coupler; or, in the Answer Mode, wait till the phone rings and then make the physical connection. With the direct coupler the computer remains plugged directly into the phone lines and, via a combination of software and hardware, is responsible for all functions including answering a ringing phone, dialing numbers, dealing with broken connections, checking for data errors and just about as many other technical details that the programmer wants to get involved in.

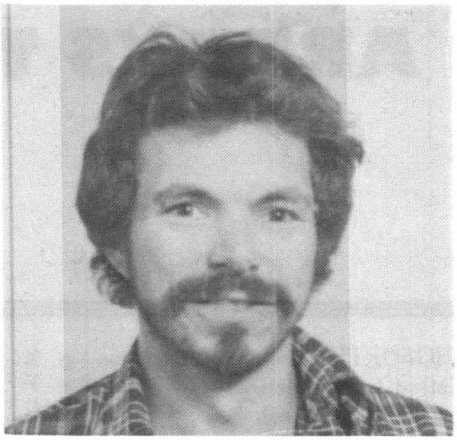
The direct coupled situation also tends to suffer less from loss of data problems due to the fact that it does not have any intermediary acoustic stages. Furthermore, with a direct coupled arrangement, a microcomputer owner is able to access his/her machine from any remote location; e.g., your computer is at the office and you have a



relatively inexpensive terminal at home giving you the full power of your computer at home. Acoustic couplers cost about $150 while the direct couplers seem to hover around the $400 mark.

In the final analysis, I chose the direct coupled equipment because I have a vision of starting a small local network for sharing resources and ideas in the Berkeley, California area. I feel that it is the responsibility of the people who own computers now to break into the new territory of Networks and thus prevent the concept from becoming solely connected with major nationwide companies. Local computer networks could revolutionize just about every aspect of our lives, giving people in communities a true and present sense of being in touch with one another.

Any persons interested in pursuing this topic should get in touch either by phone, mail or Micronet Bulletin Board.



Let me leave you with a final question, readers; if anyone knows the answer, please get in touch with me: How can data be traveling in two directions at the same time on one line, and how does the phone company send our voices two ways at the same time without having separate bands, since our voices use up the whole band? --- *WS*

ADDING ARRAYS TO FORTH

BY RALPH DEANE
Box 27
Little Fort, B.C., Canada V0E 2C0

In some versions of FORTH (notably FIGFORTH) there are no functions which can be used to define multidimensioned arrays. Other versions (those based on the Caltech manual) provide only a primitive array defining function. The power of FORTH is such, however, that these array-defining functions can be added with relative ease. This extendability of the compiler to add new classes of functions (in this case arrays) is shared by few other languages and is one of the strengths of FORTH. Functions to define one, two, three, and four dimensioned arrays will be presented here. Careful study of the program will show the underlying principles, which can also be applied to arrays with five or more dimensions.

In FORTH there are two different methods of defining functions whose purpose will be to define arrays. One way is to use ;CODE while the other is to use (BUILDS DOES). ;CODE produces the fastest runtime speed when generating addresses of array elements but is also processor-dependent due to its use of assembler language. While (BUILDS DOES) may be slower than ;CODE, all parts of the definition are in high level FORTH, which makes it processor-independent and allows the functions produced to be transportable. This is the main reason that (BUILD DOES) will be used in the function definitions. All true implementations of FORTH should have these functions available. If your version does not, find out why from the distributor. Programma International's FORTH does not have these functions and is, in any case, not a true implementation of FORTH, as defined by the FORTH interest group.

The defining functions are organized such that the functions they produce will return the absolute address of the desired array element. The indices of the element must be placed on the stack before executing the function. All indices have a lower value of 1. No error checking is provided to test for indices out of bounds.

In order to understand how the defining functions work, a knowledge of how (BUILDS DOES) works is required. An explanation of this is beyond the scope of this article and should be covered in your FORTH manual. Each of the four defining functions works in the same way.

The defining function can be broken down into two parts, compile time and runtime. The compile time portion is executed when an array is created. It consists of the code from (BUILDS to DOES). It gets the size of each dimension, stores them in the dictionary and then reserves the required words for the array storage area. Figure 1 shows the order in which the dimension sizes are stored for each array type. The amount of space reserved is equal to the product of all the dimension sizes.

```
Dimension 1 is  X
Dimension 2 is  Y
Dimension 3 is  Z
Dimension 4 is  W

1 Dimension Arrays
      first parameter field –  X

2 Dimension Arrays
      first parameter field –  Y
      second                   X

3 Dimension Arrays
      first parameter field –  Y
      second                   X
      third                    Z

4 Dimension Arrays
      first parameter field –  Y
      second                   X
      third                    Z
      fourth                   W
```

Figure 1.

The runtime is executed whenever the address of an array element is required. It expects the address of the first parameter field to be on the top of the stack, with the indices underneath it. The index of the highest dimension is closest to the top. The formulas used to compute the element addresses are outlined in Figure 2.

FPF is the address of the first parameter field. X, Y, Z, W are the dimensions of the array. I, J, K, L are the indices of the element.

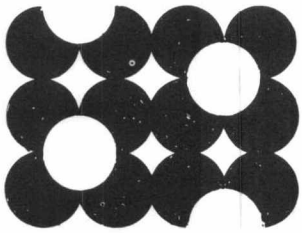
1 Dimension Arrays
I * 2 + FPF

2 Dimension Arrays
((((I – 1) * Y) + J + 1) * 2) + FPF

3 Dimension Arrays
((((I – 1) * Y) + J + 1) * 2) + (X * Y * (K – 1)) + 2 + FPF

4 Dimension Arrays
((((I – 1) * Y) + J + 1) * 2) + (X * Y * Z * (L – 1)) + 4 + FPF

Figure 2.



EXAMPLES

1 Dimensioned Arrays

50 1ARRAY PUT – define an array of length 50 called PUT to get the address of PUT[11] execute 11 PUT

2 Dimensioned Arrays

5 7 2ARRAY NAME – define a 5 X 7 array called NAME to get the address of NAME [3, 4] execute 3 4 NAME

3 Dimensioned Arrays

5 6 7 3ARRAY DIM – define a 5 X 6 X 7 array called DIM to get the address of DIM [1, 5, 2] execute 1 5 2 DIM

4 Dimensioned Arrays

2 3 4 5 4ARRAY NAX – define a 2 X 3 X 4 X 5 array called NAX to get the address of NAX[1, 3, 2, 4] execute 1 3 2 4 NAX

The version of FORTH used to develop these functions is called 6502 FORTH. There are a few differences in function names between it and FIGFORTH. These are:

6502 FORTH	FIGFORTH
'	'
SAVE	>R
UNSAVE	R>

For more information on 6502 FORTH for the KIM-1 and the Sym contact Eric Rehnke, #61 – 540 Ranchview Circle, Anaheim Hills, CA 92807.

```
0 ( SCR #01 )
1 : 1ARRAY <BUILDS DUP ' 0 DO 0 ' LOOP DOES>
2 ( RUNTIME ROUTINE )  SWAP 2 * + ;
3 : MAT ROT 1 - SWAP DUP 0 ROT * ROT + 1 + 2 * + ;
4 : 2ARRAY <BUILDS OVER OVER ' ' * 0 DO 0 ' LOOP DOES>
5 ( RUNTIME ROUTINE )  MAT ;
6 : 2MAT DUP DUP 0 SAVE 2 + 0 SAVE SAVE ROT ROT UNSAVE MAT
7 SWAP 1 - UNSAVE * UNSAVE * 2 * + 2 + ;
8 : 3ARRAY <BUILDS SAVE OVER OVER ' ' UNSAVE DUP ' * *
9 0 DO 0 ' LOOP DOES>
10 ( RUNTIME ROUTINE )  2MAT ;
11 : 3MAT SWAP SAVE DUP SAVE 2MAT 2 + UNSAVE DUP 0 SWAP DUP
12 2 + 0 SWAP 4 + 0 * * UNSAVE 1 - * 2 * + ;
13 : 4ARRAY <BUILDS SAVE SAVE OVER OVER ' ' UNSAVE DUP '
14 UNSAVE DUP ' * * * 0 DO 0 ' LOOP DOES>
15 ( RUNTIME ROUTINE )  3MAT ;
```

ROM BYTE - FINDER FOR THE APPLE II

BY STEPHEN E. BACH
Route 2, Box 50A-1
Scottsville, VA 24590

In the Sept/Nov 1977 issue of KIM User Notes, Hal Gordon, long-time friend of *DDJ*, noted that the 6502 BIT instruction can be used to non-destructively test the accumulator for any possible bit combination. The BIT instruction performs a logical AND on the byte in the accumulator and a mask byte. The mask byte must be addressable because the BIT instruction has no immediate addressing mode. Hal then presented a table of addresses in the KIM's ROM at which there were specified hex values in the range $00-FF; the KIM's ROM contained 175 of them.

The enclosed routine for the APPLE II will find such values in the Apple's Integer Basic/Monitor ROM beginning at $E000. The Apple's ROM contains all 256 values. The program stops after the screen has filled with addresses and their corresponding values to allow you to copy them if you, like me, do not have a printer. Pressing any key will start the program on its way again.

Credit should also go to Lee Moyer, who helped.

```
ROM BYTE-FINDER
      FOR
   APPLE II

0800  A9 00     LDA #00       Initialize registers to zero:
0802  85 00     STA 00        Byte-to-be-found
0804  85 01     STA 01        Addresses-printed counter
0806  85 FE     STA FE        Low byte of ROM's beginning address
0808  A9 E0     LDA #E0       High byte of ROM's beginning address
080A  85 FF     STA FF
080C  A0 00     LDY #00
080E  B1 FE     LDA (FE),Y    Load byte from ROM
0810  C5 00     CMP 00        Is it the one we're looking for ?
0812  D0 28     BNE 083C      If not go to examine next byte
0814  20 DA FD  JSR FDDA      Print byte found
0817  A9 A0     LDA #A0       Print space
0819  20 ED FD  JSR FDED
081C  98        TYA           Move low order byte of ROM address
081D  AA        TAX                to X register for printing
081E  A5 FF     LDA FF        Load hi order byte of ROM address
0820  20 41 F9  JSR F941      Print address of byte found
0823  20 48 F9  JSR F948      Print 3 spaces
0826  E6 01     INC 01        Bump addresses-printed counter
0828  A5 01     LDA 01        Test for full screen
082A  C9 5C     CMP #5C       Full screen is 92 addresses
082C  F0 05     BEQ 0833      If full go wait for key to be pressed
082E  E6 00     INC 00        Bump byte-to-be-found-register
0830  D0 D6     BNE 0808      If more bytes to find  go do it
0832  60        RTS           That's all; return to monitor

0833  20 35 FD  JSR FD35      Go watch keyboard for key-down
0836  A9 00     LDA #00       Reset addresses-printed counter
0838  85 01     STA 01
083A  F0 F2     BEQ 082E      Go to resume search for next byte

083C  C8        INY           Bump index to examine next ROM byte
083D  F0 02     BEQ 0841      If page boundary bump hi addr of ROM
083F  D0 CD     BNE 080E      Go to examine next ROM byte

0841  E6 FF     INC FF        Bump hi byte of ROM addr
0843  D0 C9     BNE 080E      Go to examine next ROM byte


0800  A9 00 85 00 85 01 85 FE A9 E0 85 FF A0 00 B1 FE
0810  C5 00 D0 28 20 DA FD A9 A0 20 ED FD 98 AA A5 FF
0820  20 41 F9 20 48 F9 E6 01 A5 01 C9 5C F0 05 E6 00
0830  D0 D6 60 20 35 FD A9 00 85 01 F0 F2 C8 F0 02 D0
0840  CD E6 FF D0 C9

Checksum = $2704
```

The ERRATA of COMPUTER PROGRAMMING

BY DONALD E. KNUTH
Computer Science Department
Stanford University
Stanford, California 94305

Herewith a portion of Donald Knuth's collected errata for The Art of Computer Programming. *Future issues of* DDJ *will contain more collected errata.* – *SR*

This report lists all corrections and changes of Volumes 1 and 3 of *The Art of Computer Programming*, as of January 5, 1979. This updates the previous list in report CS551, May 1976. The second edition of Volume 2 has been delayed two years due to the fact that it was completely revised and put in the TEX typesetting language; since publication of this new edition is not far off, no changes to Volume 2 are listed here.

The present report was prepared with a typesetting system that is now obsolete; please do not wince at the typography. All changes and corrections henceforth will be noted in TEX form on file ERRATA. TEX [ART, DEK] at SU-AI.

In spite of inflation, the rewards to error-detectors are still $2 for "new" mistakes in the second edition, $1 in the first edition.

Please do not endanger the author's morale by asking him about Volume 4. Thank you for your understanding.

The preparation of this report was supported in part by National Science Foundation grant MCS-77-23728, by Office of Naval Research contract N00014-76-C-0330, and by IBM Corporation.

1.0 throughout the book(s) *2/28/78* *2*

when the text of these books is on a computer I will try to be consistent in hyphenating compound adjectives like doubly-linked lists and storage-allocation algorithms, etc. ... but until then, such lapses are not to be considered errors

1.2 line 11 *5/27/78* *3*

Leibnitz ⇝ Leibniz

1.18 line -7 *11/29/77* *4*

the theorem ⇝ that the theorem

1.18 line 16 *11/29/77* *5*

3,... ⇝ 3,....

1.35 line 3, under the big pi *11/12/76* *6*

$n, \rightsquigarrow n$

1.41 displayed formula in exercise 32 *2/28/78* *7*

$n/\,c \rightsquigarrow n/c$

1.44 add a footnote (see p. v for style) *4/19/77* *8*

line 3 after (1): book. ⇝ book.*
footnote for bottom of page: In fact, permutations are so important, Vaughan Pratt has suggested calling them "perms." As soon as Pratt's convention is established, textbooks of computer science will be somewhat shorter (and perhaps less expensive).

1.44 lines -4, -5(twice), -7, -15, -16 *11/12/76* *9*

... ⇝ ...

1.45 lines 3, 10, 11, 12, 21 *11/12/76* *10*

... ⇝ ...

1.50 exercise 21 line 1 *7/31/76* *11*

Faa ⇝ Faà

1.51 line 13 *2/28/78* *12*

manner ⇝ matter

1.52 line 6 after Table 1 *8/25/76* *13*

Szu-yuen ⇝ *Szu-yüan*

1.56 change in Eq. (17) *11/12/76* *14*

$-r \rightsquigarrow r$ and $r \rightsquigarrow -r$

1.57 Eq. (18) *7/31/76* *15*

$n \geq 0. \rightsquigarrow n.$

1.57 line after (19) *11/12/76* *16*

$-r \rightsquigarrow r$

1.66 caption to Table 2, replace third line by: *9/21/76* *17*

see D. E. Barton, F. N. David, and M. Merrington, *Biometrika* **47** (1960), 439-445; **50** (1963), 169-176.

1.72 line -4 *11/15/78* *18*

$A_{n(k-1)} \rightsquigarrow A_{n-1)(k-1)}$

1.79 lines 8,9,10 *6/25/76* *19*

Kepler, ... life. ⇝ Johann Kepler, 1611, who was musing about the numbers he saw around him [J. Kepler, *The Six-Cornered Snowflake* (Oxford: Clarendon Press, 1966), p. 21].

1.83 line -7 *11/29/77* *20*

use same style script F in this line as in line -6 (six places)

1.90 new generalized Eq. (29) *8/25/76* *21*

$(z/(e^z-1))^n = 1 - (1/(n-1))[{n \atop n-1}]z + (1/(n-1)(n-2))[{n \atop n-2}]z^2 - \cdots = \Sigma_{k\geq 0} B_k^{(n)} z^k/k!$. (29)
(convert this to usual format for displayed equations)

1.90 update to previous correction number 25 *11/12/76* *22*

to appear, ⇝ 75-77,

1.91 replace lines 1-3 by the following new copy *8/25/76* *23*

The coefficients $B_k^{(n)}$ which appear in the last formula are called "generalized Bernoulli numbers"; Section 1.2.11.2 examines them further in the important special case $n = 1$. For small k, we have $B_k^{(n)}/k! = (-1)^k[{n \atop n-k}]\ (n-k-1)!/(n-1)!$, but when $k \geq n$ this formula breaks down since it reduces to 0 times ∞. An analogous situation holds for the power series $(z/\ln(1+z))^n$, where the coefficient of z^k for $k < n$ is $\{{n \atop n-k}\}(n-k-1)!/(n-1)!$.

1.92 line -8 *7/31/76* *24*

Faa ⇝ Faà

1.98 caption, line 2 *7/31/76* *25*

2.11 ⇝ 2.10

1.103 line 3 *7/31/76* *26*

Faa ⇝ Faà

1.110 three lines after (12) *6/25/76* *27*

$R_m \rightsquigarrow |R_m|$

1.111 line 8 *11/15/78* *28*

mately 2 ⇝ mately $(-1)^{1+k/2}_2$

1.116 line -6 *11/29/77* *29*

Analysis ⇝ A crude analysis

1.116 line -6 and Eq. (22) *11/29/77* *30*

$n^{n-1/2}$ ⇝ n^n

1.117 line 5 *11/29/77* *31*

three ⇝ two

1.118 exercise 5 *11/29/77* *32*

$n^{n-1/2}$ ⇝ n^n

1.125 line 2 *1/16/77* *33*

is loaded. ⇝ are loaded.

1.126 line 1 *1/16/77* *34*

The contents ⇝ A portion of the contents

1.126 line 7 *1/16/77* *35*

is ⇝ are

1.127 line -19 *11/15/78* *36*

Overflow may occur as in ADD. ⇝ Same as ADD but with -V in place of V.

1.127 lines -18 through -13 *11/15/78* *37*

move this paragraph in front of the SUB definition on the preceding two lines

1.134 line -12 *4/19/77* *38*

MUL requires ⇝ MUL, NUM, CHAR each require

1.137 box 05 *4/19/77* *39*

1 ⇝ *10*

1.150 lines -10,-9,-8 *4/19/77* *40*

CON ⇝ CON (4 times)

1.152 line 16 *11/29/77* *41*

facilate ⇝ facilitate

1.156 stylistic corrections *6/14/77* *42*

line 2: i.e. ⇝ e.g.
line 3: (*X* ⇝ (Here *X*
line 5: sun. ⇝ sun;
line 10: (*E* ⇝ (This number *E*
line 22: the year ⇝ that the year

1.198 lines 19-21 *6/14/77* *43*

An illustration...See also the book ⇝ See, for example, the book

1.224 line -11 *6/14/77* *44*

F = 7 ⇝ F = 9

1.225 line -9 *6/25/76* *45*

about 1946 ⇝ during 1946 and 1947

1.237 line -10 *12/19/76* *46*

down an item ⇝ an item down
up the stack ⇝ the stack up

1.248 insert new paragraph after line 4 *4/19/77* *47*

Further study of Algorithm G has been made by D. S. Wise and D. C. Watson, *BIT* **16** (1976), 442-450.

1.258 line 4 *9/21/76* *48*

we ⇝ exercise 30 describes a somewhat more natural alternative, and we

1.270 new exercise *9/21/76* *49*

30. [*17*] Suppose that queues are represented as in (12), but with an empty queue represented by F = Λ and R *undefined*. What insertion and deletion procedures should replace (14) and (17)?

1.303 exercise 9 line 4 *3/ 2/77 50*

girls ⇝ women

1.325 line 8 *4/19/77 51*

otherwise. ⇝ otherwise, making the latter node the right son of NODE(Q).

1.332 new quote to insert just before Section 2.3.2 *1/16/77 52*

Binary or dichotomous systems, although regulated by a principle, are among the most artificial arrangements that have ever been invented.
--WILLIAM SWAINSON, *A Treatise on the Geography and Classification of Animals*, Sec. 250 (1835)

1.339 line 13 *6/25/76 53*

In all ⇝ Furthermore TYPE(W) is set appropriately, depending on x. In all

1.382 line 2 *12/19/76 54*

there is a man now living having ⇝ somebody now living has

1.398 line -1 *5/27/78 55*

with ⇝ than

1.406 line -2 *1/16/77 56*

as ⇝ informally as

1.406 line 11 *5/27/78 57*

-types ⇝ -tuples

1.406 line 18 *11/15/78 58*

Polya ⇝ Pólya

1.414 step A2 lines 2-4 *2/28/78 59*

unmarked, mark it, and if ⇝ unmarked: mark it and, if (twice)

1.420 lines 14-15 *9/21/76 60*

[See the ... 372.] ⇝ An elaborate system which does this, and which also includes a mechanism for postponing operations on reference counts in order to achieve further efficiency, has been described by L. P. Deutsch and D. G. Bobrow in *CACM* **19** (1976), 522-526.

1,420 line 17 *11/29/77 61*

see ⇝ see N. E. Wiseman and J. O. Hiles, *Comp. J.* **10** (1968), 338-343,

1,437 line 18 *6/25/76 62*

For these reasons the ⇝ A contrary example appears in exercise 7; the point is that neither method clearly dominates the other, hence the simple

1,445 line 11 *1/16/77 63*

each with a random lifetime, ⇝ ***each equally likely to be the next one deleted,***

1,446 new paragraph after line 6 *1/16/77 64*

Our assumption that each deletion applies to a random reserved block will be valid if the lifetime of a block is an exponentially-distributed random variable. On the other hand, if all blocks have roughly the same lifetime, this assumption is false; John E. Shore has pointed out that type A blocks tend to be "older" than type C blocks when allocations and deletions tend to have a somewhat first-in-first-out character, since a sequence of adjacent reserved blocks tends to be in order from youngest to oldest and since the most recently allocated block is almost never type A. This tends to produce a smaller number of available blocks, giving even better performance than the fifty-percent rule would predict. [Cf. *CACM* **20** (1977), 812-820.]

1,448 line -9 *11/15/78 65*

areas ⇝ areas of the same size

1,451 line 7 *1/16/77 66*

. ⇝ ; John E. Shore, *CACM* **18 (1975), 433-440.**

1,451 yet another addition after line 7 *2/28/78 67*

. ⇝ ; Norman R. Nielsen, *CACM* **20** (1977), 864-873.

1,454 exercise 28 *4/19/77 68*

line 2: 5; for ⇝ 5. For
line 4: " ⇝ The execution time is $2u$."

1,456 line 8 *6/25/76 69*

V-1.] ⇝ V-1; and see especially also the work of Konrad Zuse, ***Berichte der Gesellschaft für Math. und Datenv.*** **63** (Bonn, 1972), written in 1945. Zuse was the first to develop nontrivial algorithms that worked with lists of dynamically varying lengths.]

1.456 line -7 *12/19/76 70*

is divisible ⇝ is not divisible

1.458 *6/25/76 71*

lines -15 thru -13: The A-1 ... code; ⇝ The machine language for several early computers used a three-address code to represent the computation of arithmetic expressions;
lines -11 and -10: the A-1 compiler language ⇝ an extended three-address code

1.460 line 2 *3/ 2/77 72*

The latter ⇝ Weizenbaum's

1.463 several changes *12/19/76 73*

line 1: . ⇝ ,
line 4: older ⇝ other
now paragraph to be inserted after line 4:

A related model of computation was proposed by A. N. Kolmogorov as early as 1952. His machine essentially operates on graphs G, having a specially designated starting vertex v_0. The action at each step depends only on the subgraph G' consisting of all vertices at distance $\leq n$ from v_0 in G, replacing G' in G by another graph $G'' = f(G')$, where G'' includes v_0 and the vertices v at distance exactly n from v_0, and possibly other vertices; the remainder of graph G is left unaltered, its components are attached to the vertices v at distance n as before. Here n is a fixed number specified in advance for any particular algorithm, but it can be arbitrarily large. A symbol from a finite alphabet is attached to each vertex, and restrictions are made so that no two vertices with the same symbol can be adjacent to a common vertex. (See A. N. Kolmogorov, *Uspekhi Mat. Nauk* **8**,4 (1953), 175-176; Kolmogorov and Uspenskiĭ, *Uspekhi Mat. Nauk* **13**,4 (1958), 3-28, *Amer. Math. Soc. Translations*, series 2, **29** (1963), 217-245.) Such graph machines can easily simulate the linking automata defined above, taking one graph step per linking step; conversely, linking automata can simulate graph machines, taking at most a bounded number of steps per graph step when n and the alphabet size are fixed. The linking model is, of course, quite close to the operations available to programmers on real machines, while the graph model is not.

1.473 exercise 44 line 2 *11/12/76 74*

x_k+y_i ⇝ x_j+y_k

1.478 line 8 *1/16/77 75*

(to appear) ⇝ **13** (1975), 251-261.

1.482 line 1 *7/31/76 76*

Faa ⇝ Faà

1.487 new answer, continued — 4/19/77 77

For example, Eq. (6) holds for all complex k and n, except in certain cases when n is a negative integer; Eqs. (7), (9), (20) are never false, although they may occasionally take indeterminate forms such as $0\cdot\infty$ or $\infty+\infty$. We can even extend the binomial theorem (13) and Vandermonde's convolution (21), obtaining $\Sigma_k \binom{r}{a+k} z^{a+k} = (1+z)^r$ and

$\Sigma_k \binom{r}{a+k} \binom{s}{b-k} = \binom{r+s}{a+b}$, formulas which hold for all complex r, s, z, a, b whenever the series converge, provided that complex powers are properly defined. [See L. Ramshaw, *Inf. Proc. Letters* **6** (1977), 223-226.]

1.487 new answer — 11/12/76 78

42. $1/(r+1)B(k+1,r-k+1)$, if this is defined according to exercise 41(b). In general it appears best to define $\binom{r}{k} = 0$ when k is a negative integer, otherwise $\binom{r}{k} = \lim_{s\to r} \Gamma(s+1)/\Gamma(k+1)\Gamma(s-k+1)$, since this preserves most of the important identities.

1.494 line 9 — 11/15/78 79

Polya ⇝ Pólya

1.499 exercise 7 — 11/15/78 80

(It is "Glaisher's constant" 1.2824271...) To ⇝ To
This formula ... n=4. ⇝ (The constant A is "Glaisher's constant" 1.2824271..., which R. W. Gosper has proved equal to $(2\pi e^{\gamma-\zeta'(2)/\zeta(2)})^{1/12}$.)

1.500 exercise 5 — 11/29/77 81

line 1: $2n-1$ ⇝ $2n+1$
line 2: has ... dx. ⇝ changes sign at $r = n - O(\sqrt{n})$, so $R = O(\int_0^n |f''(x)dx) = O(|f'(r)|) + O(|f'(n)|) = O(f(n)/\sqrt{n})$.

1.502 exercise 17(b) line 6 — 3/ 2/77 82

J2NN ⇝ J2P

1.502 exercise 19 — 4/19/77 83

24 ⇝ 42
1+1)u ⇝ 10+10)u

1.504 exercise 25 — 4/19/77 84

lines 11-12: operations" ⇝ operations," jumps on register even or odd, and binary shifts
last line: M. ⇝ M, and others could set register←rA, register←rX.

1,504

6/14/77 85

line 1: 6 ⇝ 5 (also make this change in previous correction no. 111)
line 6: 3494 ⇝ 3495 and 6 ⇝ 5
line 7: 3495 ⇝ 3496 and 5 ⇝ 4
line 9: 3506 ⇝ 3505 and 6 ⇝ 5
line 10: 16 ⇝ 14

1,511 changes to answer 14

6/14/77 86

line 1: uses as much ⇝ due in part to J. Petolino uses a lot of
line 2: as possible, in ⇝ in
line 9: INCX 1 ⇝
line 10: G ⇝ GMINUS1
lines -17 to end of page, replace by:

```
                INCA 61
                STA  CPLUS60
                MUL  =3//4+1=
                STA  XPLUS57(1:2)
CPLUS60         ENTA *
                MUL  =8//25+1=          rA = Z + 24
GMINUS1         ENT2 *                  E5.
                ENT1 1,2                rI1 = C
                INC2 1,1
                INC2 0,2
                INC2 0,1
                INC2 0,2
                INC2 773,1              rI2 = 11C + 773
XPLUS57         INCA -*,2               rA = 11C+Z-X+20+24·30 (≥ 0)
```

1,512 more changes to answer 14

6/14/77 87

delete the bottom line and replace lines 1-31 by:

```
                SRAX 5
                DIV  =30=               rX = E
                DECX 24
                JXN  4F
                DECX 1
                JXP  2F
                JXN  3F
                DEC1 11
                J1NP 2F
3H              INCX 1
2H              DECX 29                 E6.
4H              STX  20MINUSN(0:2)
                LDA  Y                  E4.
                MUL  =1//4+1=
                ADD  Y
                SUB  XPLUS57(1:2)       rA = D-47
20MINUSN        ENN1 *
                INCA 67,1               E7.
                SRAX 5                  rX = D + N
                DIV  =7=
                SLAX 5
                DECA -4,1               rA = 31 - N
                JAN  1F                 E8.
                DECA 31
                CHAR
                LDA  MARCH
                JMP  2F
1H              CHAR
                LDA  APRIL
```

1.513 new answer — 6/14/77 88

15. The first such year is A.D. 10317, although the error *almost* leads to failure in A.D. 10108+19k for $0 \le k \le 10$.

1.513 still more changes to answer 14 — 6/14/77 89

replace lines 1-6 by:

```
BEGIN   ENTX 1950          "driver"
        ENT6 1950-2000       routine,
        JMP  EASTER          uses the
        INC6 1               above
        ENTX 2000,6          subroutine.
        J6NP EASTER+1
```

1.514 line 18 — 11/29/77 90

time. ⇝ time. (It would be faster to calculate $r_n(1/m)$ directly when m is small, and then to apply the suggested procedure.)

1.515 bottom line — 11/29/77 91

Berk'ly ⇝ Berkeley

1.516 lines -4,-3 — 4/19/77 92

3)+7 ⇝ 7.5)+16

1.517 exercise 12 lines 7-10 — 5/27/78 93

delete "Thus, ...(b)."

1.518 line 5 — 5/27/78 94

19-27. ⇝ 19-27; E. G. Cate and D. W. Twigg, *ACM Trans. Math. Software* **3** (1977), 104-110.

1.516 new answer — 9/21/76 95

30. To insert, set P ← AVAIL, INFO(P) ← Y, LINK(P) ← Λ, if F = Λ then F ← P else LINK(R) ← P, and R ← P. To delete, do (9) with F replacing T.

1.550 exercise 18 — 3/ 2/77 96

denotes, ... are included. ⇝ denotes "exclusive or." Other invertible operations, such as addition or subtraction modulo the pointer field size, could also be used. It is convenient to include

1,550 exercise 2 — 3/ 2/77 97

line 2: next ... list point ⇝ next, so the links in the list must point
line 3: So ... the ⇝ Deletion at both ends therefore implies that the
line 4: ways. ⇝ ways. On the other hand, exercise 2.2.4-18 shows that two links can be represented in a single link field; in this way general deque operations *are* possible.

1,553 exercise 9 step G4 — 3/ 2/77 98

desired girls, ⇝ young ladies desired,

1,558 line -6 — 5/27/78 99

"pedigrees", ⇝ "pedigrees,"

1,575 exercise 12 line 5 — 9/21/76 100

∞. ⇝ ∞. Here $c(i,j)$ means $c(j,i)$ if $j < i$.

1,583 answer 5 — 1/ 5/79 101

There is ... exist. ⇝ When $n>1$, the number of series-parallel networks with n edges is $2c_n$ [see P. A. MacMahon, *Proc. London Math. Soc.* **22** (1891), 330-339].

1,588 fourth line before exercise 33 — 5/27/78 102

minimal. ⇝ minimal. [This argument in the case of binary trees was apparently first discovered by C. S. Peirce in an unpublished manuscript; see his *New Elements of Mathematics* **4** (The Hague: Mouton, 1976), 303-304.]

1,594 updates to previous change number 150 — 9/21/76 103

to appear, ⇝ 491-500,
(see also the important new contribution by H. G. Baker, Jr., *CACM* **21** (1978), 280-294, for which I will probably want to revise Section 2.3.5 entirely!)

1,594 update to previous change number 151 — 11/29/77 104

Clark's list-copying algorithm appeared in *CACM* **21** (1978), 351-357, and Robson's in *CACM* **20** (1977), 431-433

1,597 last line of answer 6 — 1/16/77 105

list. ⇝ list. For an alternative improvement to Algorithm A, see exercise 6.2.3-30.

1,597 exercise 8 — 6/25/76 106

line 1: also set R ⇝ also set M ← ∞, R
line 3: If R = Λ or M ⇝ If M

1.601 exercise 26 line 3 *2/28/78 107*

two. ⇝ two, with blocks in decreasing order of size.
$P \geq M \rightsquigarrow P \geq M - 2^k$.

1.601 program line number 12 *4/19/77 108*

j ⇝ *j*.

1.602 new answer *2/28/78 109*

31. See David L. Russell, *SIAM J. Computing* **6** (1977), 607-621.

1.603 addition to previous change 153 *4/19/77 110*

.] ⇝ ; Lars-Erik Thorelli, *BIT* **16** (1976), 426-441.

1.606 exercise 41, numerator in value of a[5] *6/14/77 111*

19559 ⇝ 18535

1.617L *6/25/76 112*

delete A-1 compiler, 458.

1.617L Aardenne-... *11/29/77 113*

Taniana ⇝ Tatyana

1.617R *12/19/76 114*

AMM ⇝ *AMM*

1.618L *5/27/78 115*

Baker, Henry Givens, Jr., 594.

1.618R *4/19/77 116*

add p487 to entry for Binomial theorem, generalizations of

1.619L Bobrow entry *9/21/76 117*

add p420

To be continued in future issues.

Random Ruminations On Networks Of The Future

BY JONATHAN SACHS
The Orthocode Corporation
6713 Richmond Avenue
Richmond View, CA 94805

We asked Jonathan Sachs to speculate theoretically about networks and the future . . . .

Popular Use of Networks: What It Will Take

In the past, computers have been attractive only to those oddballs who were interested in computer hardware or software. Networking can make them attractive to everyone who needs to deal with information by making them potential tools for processing information about the world at large. This promises to be the first development that makes computers directly useful to a large proportion of mankind.

There has been a great deal of speculation (and some research) on this question of how computers will affect the quality of life. Scenarios run the gamut from a plugged-in society where everyone is in touch with everything, to a totally dehumanized society where people interact with machines to the exclusion of one another. For a thorough and responsible (but highly readable) study of the issues, see *The Network Nation* by Hiltz and Turoff (Addision-Wesley Publishing Company).

Networks must be not only useful, but usable by the public, or they will win little acceptance. What do I mean by "the public?" I mean: a substantial proportion of everyone who lives in the part of the world we're considering. Here are some important characteristics of the public a successful network must serve:

- It includes large numbers of people with different technical orientations: toward computer programming, law, sculpture, animal husbandry, etc.;
- It includes large numbers of peole who have no technical orientation at all;
- Its members would demand widely varying types and amounts of information from a network;
- Most of its members would have little or no motivation to learn to use a network, except for whatever immediate benefits they could anticipate before they started.

To serve this audience, a network must be usable (at least on an elementary level) by non-technical people of average intelligence, with very little training.

But a successful network must also be able to accommodate experienced users with sophisticated demands. Is this requirement for power and versatility at odds with the equally important need for ease of use? Not necessarily.

We could design a powerful system that could be "packaged" in a variety of simple ways for relatively specialized uses. One can imagine network customers ordering options from a network service representative, just as telephone subscribers order different kinds of service from the phone company. Users willing and able to do so could build their own service packages.

The early packaged offerings of such a service would probably be digital analogs of newspapers, telephones, and other things the public is already comfortable with.

Less conventional uses would follow in time. For example, one can imagine a "therapy group" network conference in which every member but the leader was guaranteed to remain anonymous, or an "information brokerage" that acts like a combination of publicity agency and clipping service, putting generators and consumers of information in touch with each other.

A single system accommodating both beginners and old hands would have a built-in mechanism for introducing innovation to the mainstream of use. It is the simple fact that there would be no sharp dividing line between an experimental use and a publicly available one. When an advanced user developed an application that was genuinely useful to the public, the public would begin to use it. At some point the operators of the network would see fit to offer that application as an official service. Thus the network-using public would serve as both a laboratory for developing new applications and a lobby for adopting successful ones.

A Trap for Designers

There's a saying that the generals who plan military strategy are always guilty of refighting the last war. Perhaps network designers are guilty of a similar error: designing networks for the future that can be built from components available today.

It is difficult to design a system with components that do not yet exist, but it is pointless to design an eminently buildable system that turns out to be unsuccessful because it is not suited to the needs of its users. If our goal is to design a network that will be acceptable to the general public, we must determine (1) what capabilities such a net will have to have, and (2) what present-day technology, if any, can provide them at an economic cost. If no suitable technology is at hand we may change or compromise our goals, but we wll get nowhere by trying to fit the system's requirements to the technology that is available.

To take a simple example, consider terminal display *versus* hard copy. Most of the network-like systems so far available to the public (chiefly television-based systems like England's Prestel) use display only. The display was already available in the prospective user's TV set, while a hard copy printer would have to be added at extra cost. But is display output sufficient? Probably not. Unlike a television set, a computer network is used primarily to process information. And information, once processed, must often be preserved in accessible form.

Why can't information be stored electronically and displayed at need? Because electronically-stored information is not accessible in the same sense that hard copy is. There are countless things one can do to hard copy but can't do to a display; for example, one can:

- Fold it, tear it, or staple it to things.
- Mark it with a felt-tip marker.
- Scribble notes on it.
- Work with it while sitting in an easy chair.
- Take it along to the park.
- Place it next to several like objects in order to compare, collate, or cross-reference them.
- Place it on a part of a desk where one will unconsciously reach for it when one needs it.
- Cherish it as a memento.

This is not to say that video displays are not useful; only that hard copy has some functions that displays cannot usurp, and that therefore a general-purpose information system that offers no hard copy will have a hard time gaining acceptance.

For another example, take image quality. Today's standard of quality seems to be the dot-matrix printer. Is that good enough for a public network?

Consider what kinds of information the network will convey. Initially, at least, the greatest volume of use will be in fairly predictable replacements for older media.

Newspaper material? Newspapers contain many pictures—not only for their information value, but because most readers want the paper to be an easy read, and are unwilling to digest solid blocks of text. Some parts of the paper (comics, movie ads) are inconceivable without pictures. Television, the only other comparable medium, is all pictures. (Radio is an unfair comparison; it has sound, and the computer nework does not have that yet, either.)

How about instructional material and reference material? Both tend to use a lot of graphics at all levels of difficulty (in some strata the graphics are called "equations").

For computerized conferences or telecommuting (working from one's home through a computer) the requirements are a little more complex. In some situations words would be sufficient, as they are in telephone conversations, and the need for graphics would not come up. But when graphics were necessary,

there would be the added difficulty of giving each user a way to input pictures as well as output them. There would probably have to be several ways to input pictures: through keyboard commands, fingerpainting on a screen, digitizing existing line or continuous-tone images, etc.

So it appears that a generally useful public network would have to offer both line drawing and halftone graphics, at least approaching the quality of newspaper pictures, and preferably somewhat better. Could a dot matrix printer produce a satisfactory result? Maybe, but that criterion would disqualify most of the printers made today! If it is recognized as necessary, a different kind of hardware must be designed.

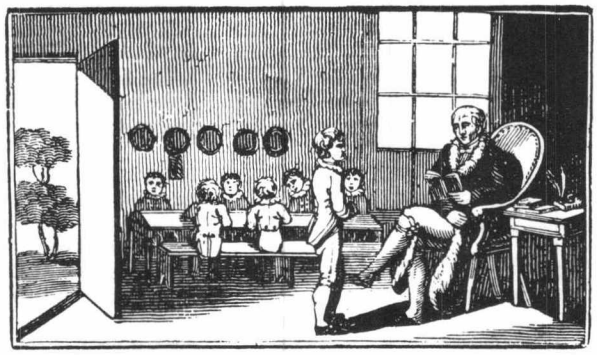
A long list of other questions could be asked along the same lines. What sort of input device(s) will users want? How fast will transmission have to be? Will color output be necessary/desirable? What about color input?

Some Wild and Crazy Possibilities

While we're on the topic of capabilities, what options could we add to a public network to make it more useful, interesting, or whatever else?

Another way to ask the question is: what limitations on the forms networked information can take have we unconsciously accepted, and what could we do if those limitations were removed?

Suppose we didn't content ourselves with moderately good black-and-white dot images, but went for high resolution color. The things the network could convey would now expand to include aesthetically meaningful reproductions of artwork. The network might even become a medium for creating original art. And there are other things the network could then accommodate, such as medical literature and maps.



Graphics may be viewed as a liberation from the restrictions of text. But it may also be viewed as a restriction itself. Computers can analyze and generate sound. Why shouldn't a network do so?

A possibility: computerized catalogs of sound recordings. A user finds a recording he wants through an index; then the computer plays it through the network. Such a system could be used to catalog sounds that were too short to deal with readily in a conventional recording library, *e.g.*, individual bird calls.

Another possibility: a data base of *digitized* music. Audiophiles could search and manipulate this data base directly, just as library researchers could search and manipulate a data base of text.

Another possibility: co-ordinated teleconferences (or classrooms, or dramatic performances) which make simultaneous use of audio, video, and computerized text/graphics. For example engineers could discuss a design using audio to communicate by speech, while using computer graphics to develp the design, and video to examine samples or the data.

The most significant developments in networking are impossible to predict. They will be products of genius, and so will be inconceivable until they start to happen!

Displaying Two Columns with North Star

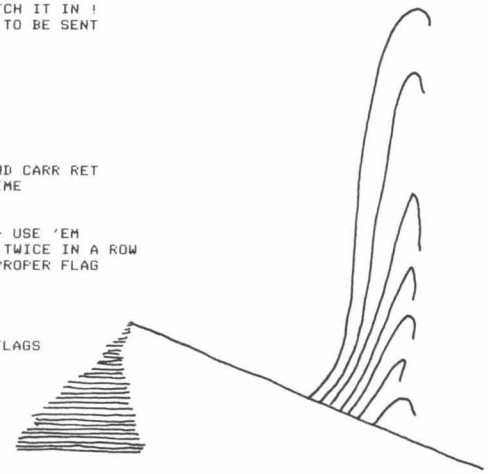
O.K. HUDSON
334 Oney Drive
San Antonio, Texas 78209

I have enjoyed the contributions from your other readers, and feel that now the time is ripe for me to reciprocate. Attached please find an 8080 Assembly listing for making a North Star Disk display its catalogue in two columns. Several other contributors have addressed this prolbem with varying degrees of success. This program is different in that it tries to make clear the required changes for the 'latest' DOS (which I think is V.4) and the old DOS (which may be V.3). Both kinds are around—in addition to some others too.

This program differs in one other important aspect: the other programs relied on the terminal generating a carriage return. This does not happen when a printer is used. This program will generate a carriage return after 80 spaces.

Keep up the good work. If you don't want to publish this, you may trash it. I don't believe in copyrights—and who cares anyway. We mostly do this for fun and if we also make a buck now and then —fine.

```
A 19EE

19EE                    0010 ************************************************
19EE                    0020 *                                              *
19EE                    0030 *        DOUBLE COLUMN FILE LISTING            *
19EE                    0040 *                                              *
19EE                    0050 *     SEE DDJ #39,43  #41,44  #42,44           *
19EE                    0060 *                                              *
19EE                    0070 ************************************************
19EE                    0080 *
19EE                    0090 *
19EE                    0100 *THIS PROGRAM REQUIRES PATCHES TO THE DOS AND AN
19EE                    0110 *EXTRA SUBROUTINE IN RAM TO CAUSE THE DIRECTORY
19EE                    0120 *LISTING TO BE PRINTED IN TWO COLUMNS.
19EE                    0130 *THE EXTRA SUBROUTINE IS LOCATED AT 19EE BECAUSE
19EE                    0140 *MY DOS PERSONALIZATION AREA IS FULL.
19EE                    0150 *
19EE                    0160 *IN ORDER TO GET CORRECT ADDRESSES ON FILES WHERE
19EE                    0170 *THEY ARE NOT NORMALLY SPECIFED, ONE MUST DO A
19EE                    0180 * *TY 'WHATEVER' 1  ADDRESS - THEN REPEAT WITH
19EE                    0190 *THE PROPER FILE TYPE -- THE ADDRESS WILL
19EE                    0200 *BE REMEMBERED BY THE DISK FILE.
19EE                    0210 *
19EE                    0220 *THIS SOURCE WILL ALSO HANDLE THE OLD DOS IF THE
19EE                    0230 *FOLLOWING EQUATES ARE SUBSTITUTED:
19EE                    0240 *   DOS  EQU 25EEH
19EE                    0250 *   DOS2 EQU 2709H
19EE                    0260 *   LFCR EQU 270FH
19EE                    0270 *
19EE                    0280 *SYSTEM EQUATES FOR N* DOS V.4
19EE                    0290 ORG    EQU   19EEH
19EE                    0300 DOS    EQU   2614H
19EE                    0310 DOS2   EQU   2704H
19EE                    0320 LFCR   EQU   270CH
19EE                    0330 *
19EE                    0340 *IN ADDITION THE FOLLOWING PATCHES MUST BE MADE
19EE                    0350 *WITHIN THE DOS ITSELF:
19EE                    0360 *    2601  FE 0A
19EE                    0370 *    2603  D2 EE 19
19EE                    0380 *    2611  CD F6 19
19EE                    0390 *
19EE                    0400 *
19EE                    0410 *FOR THE OLD DOS (V.3 ?), THE DOS PATCHES ARE:
19EE                    0420 *    25DB  FE 0A
19EE                    0430 *    25DD  CA EE 19
19EE                    0440 *    25EB  CD F6 19
19EE                    0450 *
19EE 0E 12              0460        MVI   C,12H         *THIS IS THE SPACE ALLOWED FOR AN ENTRY
19F0 CD FB 19           0470        CALL  BLANK         *THIS SUBROUTINE IS ALREADY IN THE DOS
19F3 C3 14 26           0480        JMP   DOS           *GET ANOTHER ENTRY
19F6 C3 03 1A           0490        JMP   PSUB          *DOS RE-ENTRY POINT
19F9 0E 0F              0500 REN    MVI   C,0FH         *THIS IS THE SPACE BETWEEN COLUMNS
19FB CD 04 27           0510 BLANK  CALL  DOS2          *THIS IS THE ACTUAL SOURCE OF THE BLANKS
19FE                    0520 *                          'WITHIN THE DOS --- IF YOU WANT SOME
19FE                    0530 *                           OTHER CHARACTER YOU MAY PATCH IT IN !
19FE 0D                 0540        DCR   C             *COUNT THE BLANKS REMAINING TO BE SENT
19FF C2 FB 19           0550        JNZ   BLANK         *FINISHED ??
1A02 C9                 0560        RET
1A03                    0570 *
1A03                    0580 *
1A03                    0590 *SUBROUTINE TO PROVIDE LINE FEED AND CARRIAGE RETURN
1A03                    0600 *
1A03 3A 1C 1A           0610 PSUB   LDA   STOR1         *SAVE THE ACCUMULATOR VALUE
1A06 FE 00              0620        CPI   0             *WAS IT A 0 ??
1A08 CA 13 1A           0630        JZ    DSUB          *IF IT WAS THEN LINE FEED AND CARR RET
1A0B 3E 00              0640        MVI   A,0           *MAKE SURE IT IS A 0 NEXT TIME
1A0D 32 1C 1A           0650        STA   STOR1         *BY PUTTING IT IN STORAGE
1A10 C3 F9 19           0660        JMP   REN           *GO PRINT THE NEXT ENTRY
1A13 CD 0C 27           0670 DSUB   CALL  LFCR          *THESE ARE ALSO IN THE DOS - USE 'EM
1A16 3E 0F              0680        MVI   A,0FH         *MAKE SURE WE DON'T DO THIS TWICE IN A ROW
1A18 32 1C 1A           0690        STA   STOR1         *BY FILLING STORAGE WITH A PROPER FLAG
1A1B C9                 0700        RET
1A1C                    0710 *
1A1C                    0720 *
1A1C                    0730 *STORAGE TABLE
1A1C 00 00              0740 STOR1  DW    0             *THIS IS WHERE WE KEEP THE FLAGS

SYMBOL TABLE

BLANK 19FB    DOS   2614    DOS2  2704    DSUB  1A13    LFCR  270C
ORG   19EE    PSUB  1A03    REN   19F9    STOR1 1A1C

B
```



A Versatile Memory Dump in Basic

BY HAL KNIPPENBERG
2514 Blueberry Drive
Augusta, GA 30906

Dear *DDJ*:

I am often asked how to develop memory dump programs. The main complaint is that most memory dump programs are too specialized—they work only in one mode. For instance, they will dump Hex when you want Decimal. Or they will dump Octal when you want Hex.

The program I have developed in answer to these questions is a generalized MEMORY DUMP program written in BASIC. It uses two base conversion subroutines and can work in four different modes. In addition, it prints an ASCII dump as well.

I wrote the program in Digital Group's MAXI-BASIC. If being used with another BASIC which uses LEFT, RIGHT, and MID string functions, change the following:

LINE 500 AND 655 . . .Change the A$(1,1) to LEFT $(A$,1)
LINE 725 . . . . . .Change A$(I,1) to MID$(A$,I,1)
LINE 915 . . . . . .Change H$(1,3) to LEFT$(H$,3) and change H$(4) to RIGHT $(H$,3)
LINE 565 . . . . . .Many BASICS use the command PEEK instead of EXAM

I have used many REMARK statements to help show the structure of the program, but you do not need to type in any of the REM lines for the program to work properly.

Two other lines might cause trouble, lines 240 and 510. MAXI-BASIC does not have a quick way to clear the monitor, so I use a machine language subroutine. TRS-80 users can use CLS to do the same thing. If your BASIC does not have a clear-screen function, GOSUB to the following BASIC subroutine:

```
FOR I = 1 to 16
      PRINT
NEXT I
RETURN
```

This will also clear the screen, although it takes a moment longer.

```
LIST
  100 REM  MEMDMP    1 DECEMBER 1979     HAL KNIPPENBERG
  105 REM
  110 REM  THIS PROGRAM WILL DUMP MEMORY USING USING OCTAL,
  115 REM  SPLIT-OCTAL, HEX OR DECIMAL MODES AND ASCII
  120 REM
  125 REM  VARIABLES:  A,A$ . . ANSWERS TO MACHINES QUESTIONS
  130 REM              A1$ . . . ASCII CODE
  135 REM              B . . . . NUMBER BASE.
  140 REM              B$ . . . MODE NAME
  145 REM              C . . . . COUNTS DIGITS
  150 REM              D . . . . DIGITS IN ADDRESS
  155 REM              D1 . . . . DIGITS PER NUMBER OUTPUT
  160 REM              E . . . . ERROR FLAG
  165 REM              F . . . . FINISH ADDRESS
  170 REM              H . . . . HOLDS DIGIT BEING CONVERTED
  175 REM              H$ . . . . HOLDS ASCII VERSION OF NUMBER
  180 REM              I,J . . . INDEX COUNTERS
  185 REM              L . . . . LINE COUNTER
  190 REM              M . . . . MODE FLAG
  195 REM              S . . . . START ADDRESS
  200 REM              S1 . . . . SCREENFULL OR SCROLL FLAG
  205 REM              T . . . . TENS PARAMETER FOR SUBROUTINES
  210 REM              X . . . . POWER OF DIGIT BEING CONVERTED
  215 REM
  220          DIM A$(7),A1$(8),B$(11),H$(7)
  225 REM
  230 REM  * * * * * * * START MAIN ROUTINE * * * * * * * * * * *
  235 REM
  240          Z = CALL(12762)          :REM  CLEAR SCREEN
  245 REM
  250 REM  - - - - - -  GET MODE - - - - - - - - - - - - - - - - -
  255 REM
  260          PRINT TAB(15);"MEMORY DUMP"
  265          PRINT
  270          PRINT"THIS PROGRAM WILL OPERATE IN FOUR MODES:"
  275          PRINT
  280          PRINT" 1)  DECIMAL          2)  HEX"
  285          PRINT" 3)  OCTAL            4)  SPLIT-OCTAL"
  290          PRINT
  295          INPUT"WHICH MODE DO YOU WANT (1 - 4)  ? ";A
  300          IF (A < 1) OR (A > 4) THEN 295
  305             IF A <> 4 THEN 350
  310                PRINT
  315                PRINT"IN SPLIT-OCTAL YOU MUST SEPARATE THE PAGE FROM"
  320                PRINT"THE BYTE ADDRESS WITH A SPACE.  FOR EXAMPLE:"
  325                PRINT"001234 SHOULD BE ENTERED AS 001 234"
  330                PRINT
  335 REM
  340 REM  - - - - - SET CONSTANTS AND FLAGS - - - - - - - - - -
  345 REM
  350          ON A GOTO 355,360,365,370
  355             B = 10 :D = 5 :M = 1 :B$ = "DECIMAL" :GOTO 390
  360             B = 16 :D = 4 :M = 2 :B$ = "HEX"     :GOTO 390
  365             B = 8  :D = 6 :M = 3 :B$ = "OCTAL"   :GOTO 390
  370             B =  8 :D = 6 :M = 4 :B$ = "SPLIT-OCTAL"
  375 REM
  380 REM  - - - - - GET START AND FINISH ADDRESSES - - - - - - -
  385 REM
  390          PRINT
  395          PRINT"ENTER THE START ADDRESS USING ";B$;
  400          INPUT" ? ";A$
  405          GOSUB 705
  410          IF E = 1 THEN PRINT"ADDRESS INCORRECT" :GOTO 390
  415          LET S = T
  420 REM
  425          INPUT"ENTER THE END ADDRESS ?  ";A$
  430          GOSUB 705
  435          IF E = 1 THEN PRINT"ADDRESS INCORRECT" :GOTO 425
  440          LET F = T
  445 REM
  450          IF F > S THEN 485
  455             PRINT
  460             PRINT"END ADDRESS MUST BE GREATER THAN START !"
  465             GOTO 390
  470 REM
  475 REM  - - - - COMPUTE AND PRINT MEMORY DUMP - - - - - - - - -
  480 REM
  485          PRINT"DO YOU WANT TO STOP WITH EACH SCREENFULL ";
  490          INPUT A$
  495          IF A$(1,1) = "Y" THEN S1 = 1 ELSE S1 = 0
  500 REM
  505 REM  * * * * * * * END OF MAIN ROUTINE * * * * * * * * * * *
  510 REM
  515 REM  - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  520 REM  SUBROUTINE:  BASE TO DECIMAL CONVERSION
  525 REM  IN:    B, A$
  530 REM  OUT:   T
  535 REM
  540          T = 0 :E = 0 :X = 1 :C = 0
  545          FOR I = LEN(A$) TO 1 STEP -1
  550             C = C + 1                :REM  COUNTS DIGITS
  555             H = ASC(A$(I,I)) - 48    :REM CHANGE ASCII TO NUMBER
  560             IF H > 9 THEN H = H-7    :REM CHANGE A-F TO 10-15
  565 REM
  570             IF H = -16 THEN X = 256 :GOTO 770
  575             IF (H < 0) OR (H > B-1) THEN E = 1
  580             IF (M = 4) AND ((C=3 OR C=6) AND (H > 3) THEN E = 1
  585 REM
  590             T = T + (H*X)
  595             X = X * B
  600          NEXT I
  605          IF (T < 0 OR T > 65535) THEN E = 1
  610          RETURN
  615 REM  - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  620 REM  SUBROUTINE:  DECIMAL TO BASE CONVERSION
  625 REM  IN:    B, D, T
  630 REM  OUT:   H$
  635 REM
  640          H$ = "" :E = 0
  645          IF B <> 10 THEN 835
  650             H$ = STR$(T)
  655             H$ = H$(2)
  660             GOTO 910
  665          IF B = 16 THEN X = 4096 ELSE X = 32768
  670          IF M = 4 THEN X = 16384
  675          H = INT(T/X)
  680          T = T - (H*X)
  685          IF H$ = "" THEN IF H = 0 THEN 885
  690 REM
  695          H = H + 48                  :REM CHANGE NUMBER TO ASCII
  700          IF H > 57 THEN H = H + 7    :REM CHANGE 10-15 TO A-F
  705 REM
  710             J = H$ + CHR$(H)
  715             X = X/B
  720             IF X < 1 THEN 910
  725             GOTO 845
  730 REM
  735          IF LEN(H$) < D1 THEN H$ = "0" + H$ :GOTO 910
  740          IF (M=4) AND (D1=6) THEN H$=H$(1,3)+""+H$(4)
  745 REM
  750          RETURN
  755 REM
  760 REM
  765 REM
  770 REM
  775 REM
  780 REM
  785 REM
  790 REM
  795 REM
  800 REM
  805 REM
  810 REM
  815 REM
  820 REM
  825 REM
  830 REM
  835 REM
  840 REM
  845 REM
  850 REM
  855 REM
  860 REM
  865 REM
  870 REM
  875 REM
  880 REM
  885 REM
  890 REM
  895 REM
  900 REM
  905 REM
  910 REM
  915 REM
  920 REM
  925 REM
READY
SAVE#1 MEMDMP
```

CONVERSION: ALS-8 TO CP/M

BY EDWARD L. ELIZONDO
6 Cypress Court
E. Windsor, N.J. 08520

I was delighted to find in your January issue not one, but two contributed programs to convert Processor Technology assembler format files to a format compatible with the CP/M assembler. I am presently running a 48K CP/M system, using CUTER in ROM at C000 hex to drive a Processor Technology VDM. My weekend project started with the idea of converting the CUTER source file (generously distributed by Processor Tech on the CUTER/BASIC5 cassette) to CP/M format, make a few alterations, and reassemble it at F000 hex.

Unfortunately, neither published program quite met my requirements. Hollis Frampton's CPCON is limited to programs small enough to fit in memory and requires downloading to tape, while Mark Zeiger's CHANGE did not correctly copy with commented lines, ASC pseudo-ops and other ALS-8 peculiarities such as dummy period operands.

Since frustration is the spur of creativity, I took it upon myself to generate yet a third conversion program, herein submitted to you.

ALS8CPM successfully converts the bulk of all ALS-8 statements to a form digestible by the CP/M ASM. Using ALS8CPM, the CUTER source files (total size = 58K) were converted with only seven residual errors flagged by ASM on reassembly. The program is fairly straightforward and fully commented, so it can easily be modified if desired.

A few comments about the program: I chose not to convert negative numbers used in immediate operands (MVI A, -1) for fear of creating subtle errors in programs using negative signs in arithmetic expressions. Not processed by ALS8CPM are the infrequent ALS-8 pseudo-ops COM, NST and LST, the INTEL pseudo-op TITL, and program use of ASM reserved words such as SET, IF and ENDIF. Also not handled are PSW and SP equate statements used to add these register names to the ALS-8 symbol table (an unforgiveable commission on the part of the late Processor Tech). Residual errors after conversion involving any of the preceding can quickly be located by a trial reassembly and corrected by editing.

I intend to submit this program to the CP/M User's Group and to the Processor Technology User's Society (PROTEUS). Incidentally, even though I have never owned a SOL, I must publicly compliment Stan Sokolov and the latter group for three years of publishing an excellent newsletter and attempting to protect the member's investment in time and money by providing a wealth of information, reviews, valuable user feedback on various commercial products, tutorials on such diverse subjects as PASCAL and CP/M, and much inside information on what really was happening at Processor Tech.

For any impatient souls out there who don't feel like typing up ALS8CPM, I will provide the source code on an 8 inch standard CP/M format diskette for $10 to cover the cost of the diskette, mailer and shipping charges, until the user group distributions are available.

```
;*************************************************
;*                                               *
;*                ALS8CPM.ASM                    *
;*                                               *
;*  PROGRAM FOR CONVERTING PROCESSOR TECHNOLOGY  *
;*  ALS-8/SPKG-1 ASSEMBLER FILES TO CP/M FORMAT  *
;*                                               *
;*           (C) COPYRIGHT 1980                  *
;* ED ELIZONDO 6 CYPRESS CT EAST WINDSOR NJ 08520 *
;*     A L L   R I G H T S   R E S E R V E D     *
;*                                               *
;* THIS PROGRAM MAY BE USED FREELY BY COMPUTER    *
;* HOBBYISTS FOR NON-COMMERCIAL APPLICATIONS.     *
;* IT MAY NOT BE USED FOR COMMERCIAL APPLICATIONS *
;* OR SOLD WITHOUT WRITTEN PERMISSION OF THE      *
;* AUTHOR.                                        *
;*                                               *
;*          REV 1.6      31 JAN 1980             *
;*                                               *
;*************************************************
;
; NOTE:  This program processes assembly language files
; prepared for the Processor Technology ALS-8 or Software
; Package No 1 assemblers to a format compatible with
; the CP/M ASM assembler. It requires the presence of
; CUTER/SOLOS or some other means of loading the ALS-8
; source file onto memory so that it can be saved by CP/M.
; It does not require the presence of anything other than
; CP/M during the remaining operations.
;
; The translation process operates in accordance with
; the following algorithms:
;  Each line is converted independently.  The line byte count
; and line number fields are deleted from the source lines
; and each line is terminated with a CR/LF sequence.
;  A semicolon is placed ahead of all comment lines (for
; compatibility with MAC) and ahead of all comment fields.
; Leading spaces are deleted from comment fields only.
; Comment lines and the remainder of comment fields are
; passed unchanged, except for deletion of any exclamation
; marks.
;  Periods used as dummy operands for formatting purposes
; are removed, as well as any leading spaces appearing in
; instructiona and operand fields.  Operands enclosed in
; single quotes are passed unchanged.
;
;  ASC pseudo-operators are changed to DB and the corresponding
; delimiters changed to single quotes.
;  The '>' pseudo-operator is converted to an 'AND OFFH' post
; operator and the '<' is similarly converted to a 'SHR 8'.
;  Instruction, operand and comment fields are tabbed
; and colons are added to labels to "pretty" the output.
;  The program utilizes slightly modified versions of the
; CP/M User's Group LIOS.ASM disk I/O routines (CPMUG 1.19)
;
; TO USE: 1) load your ALS-8 format program into RAM at 100H
;             (If using SOLOS/CUTER, note the first two digits
;              of program size returned by the load message.
;              Convert to decimal and add 1 to obtain the
;              number nn required in step 3).
;         2) boot up CP/M
;         3) type SAVE nn NAME1.EXT (where nn is number of
;            pages of program size in decimal)
;         4) run the conversion program by typing:
;              ALS8CPM NAME1.EXT NAME2.ASM
;            where NAME2.ASM is the desired name of the CP/M
;            compatible output file (note spaces between
;            the above name fields)
;         5) edit the resulting ASM file to add the appropriate
;            ORG statement.
;         6) assemble to locate any residual unique errors.
;
;*************************************************************
;
;       BDOS ADDRESSES:
;
BDOS:   EQU     0005            ;BDOS ENTRY POINT
SOURCE: EQU     5CH             ;SOURCE FILE FCB
TBUF:   EQU     0080H           ;TRANSFER BUFFER
;
;       BDOS FUNCTION CODES:
;
PRINT:  EQU     9
OPEN:   EQU     15
CLOSE:  EQU     16
MAKE:   EQU     22
READ:   EQU     20
WRITE:  EQU     21
DELETE: EQU     19
```

```
;
;       ASCII CODES:
;
SPACE:  EQU     20H
QUOTE:  EQU     27H
TAB:    EQU     09H
CR:     EQU     0DH
LF:     EQU     0AH
;
;*************************
;*      MAIN PROGRAM     *
;*************************
;
        ORG     100H
;
        LXI     H,0             ;GET OLD STACK POINTER
        DAD     SP
        SHLD    OLDSTK          ;SAVE IT FOR EXIT
        LXI     SP,NEWSTK       ;SET UP NEW STACK
;
        LXI     D,MSSG0         ;SAY HELLO
        CALL    MESSAGE
;
        CALL    GETNAMES        ;GET FILE NAMES FROM COMMAND LINE
        CALL    OPENFILES       ;OPEN FILES FOR ACCESS
        LXI     D,MSSG8         ;WE'RE DOING FINE
        CALL    MESSAGE
;
        CALL    PROCESS         ;PROCESS LINE BY LINE
        CALL    WEOF            ;FILL LAST INCOMPLETE PAGE
;
ABEND:  CALL    CLOSEFILE       ;CLOSE FILES
        LXI     D,MSSG9         ;WE'RE FINISHED
        CALL    MESSAGE
;
FIN:    LHLD    OLDSTK          ;RESTORE OLD STACK POINTER
        SPHL
        RET                     ;RETURN TO CP/M
;
;
;*************************************
;*          G E T N A M E S          *
;* SUBROUTINE TO CHECK & GET FILE NAMES *
;*************************************
;
GETNAMES:
        LDA     SOURCE+1        ;FIRST CHAR OF SOURCE NAME
        CPI     SPACE           ;EMPTY?
        JZ      ERROR6          ;YES, ERROR
        LDA     SOURCE+17       ;FIRST CHAR OF DEST NAME
        CPI     SPACE           ;EMPTY?
        JZ      ERROR7          ;YES, ERROR

;       MOVE DEST FCB TO SAFE PLACE
        LXI     D,SOURCE+16     ;START OF TEMP DEST FCB
        LXI     H,DEST          ;WORKING DEST FCB
        MVI     B,16            ;NUMBER OF FCB CHARACTERS
        CALL    MOVE            ;MOVE NAME
;
;       INITIALIZE FCB'S
        XRA     A               ;GET A 0
        STA     SOURCE+32       ;ZERO SOURCE NR FIELD
        STA     DEST+32         ;ZERO DEST NR FIELD
        RET
;
;*************************************
;*        O P E N F I L E S          *
;*  SUBROUTINE TO OPEN FILES FOR ACCESS *
;*************************************
;
OPENFILES:
        MVI     C,OPEN          ;GET OPEN FUNCTION
        LXI     D,SOURCE
        CALL    BDOS            ;OPEN SOURCE FILE
        CPI     0FFH            ;OPEN ERROR?
        JZ      ERROR1          ;YES, ABORT
;
        MVI     C,DELETE        ;DELETE FUNCTION
        LXI     D,DEST
        CALL    BDOS            ;DELETE EXISTING DEST FILE
        MVI     C,MAKE          ;MAKE FUNCTION
        LXI     D,DEST
        CALL    BDOS            ;CREATE NEW DEST FILE
        CPI     0FFH            ;MAKE ERROR?
        JZ      ERROR3          ;YES, ABORT
        MVI     C,OPEN          ;OPEN FUNCTION
        LXI     D,DEST
        CALL    BDOS            ;OPEN DEST FILE
        RET
;
;*************************************
;*          P R O C E S S            *
;*  ROUTINE TO PROCESS EACH LINE     *
;*************************************
;
PROCESS:
        XRA     A               ;CLEAR STRING FLAG
        STA     FLAG
        CALL    GETBYT          ;GET LINE CHAR COUNT
        CPI     1               ;END OF FILE?
        JZ      EOFMARK         ;YES, PROCESS IT
        DCR     A               ;UPDATE CHAR LINE COUNT
        STA     EOLCNT          ;SAVE IT

        MVI     B,5             ;FIELD COUNT
LNUM:   PUSH    B               ;MOVE PAST LINE NUMBER FIELD
        CALL    GETBYT          ;READ A BYTE
        POP     B
        CALL    COUNT           ;DECREMENT LINE COUNT
        JZ      PREMEND         ;PREMATURE END OF LINE
        DCR     B               ;DECREMENT FIELD COUNT
        JNZ     LNUM            ;UNTIL DONE
;
LABEL:  CALL    GETBYT          ;PROCESS LABEL FIELD
        CPI     '*'             ;COMMENT LINE?
        JZ      COMLINE         ;YES, PROCESS IT
        CPI     SPACE           ;MISSING LABEL?
        JZ      INSTR           ;THEN PROCESS INST
LABEL1: CALL    OUTBYT          ;OUTPUT LABEL CHAR
        CALL    COUNT           ;DECREMENT LINE COUNT
        JZ      EOLINE          ;IF END OF LINE
        CALL    GETBYT          ;GET NEXT BYTE
        CPI     SPACE           ;END OF ALS-8 LABEL?
        JZ      LABEL2          ;YES, EXIT LOOP
        CPI     ':'             ;END OF INTEL LABEL?
        JNZ     LABEL1          ;NO, KEEP GOING
LABEL2: MVI     A,':'           ;YES, OUTPUT A COLON
        CALL    OUTBYT          ;AND DROP THRU
INSTR:  MVI     A,TAB           ;PROCESS INSTR FIELD
        CALL    OUTBYT          ;TAB TO NEXT COLUMN
        CALL    DELSP           ;DELETE LEADING SPACES
        JC      EOLINE          ;IF END OF LINE

;       CHECK FOR ASC PSEUDO-OP
        CPI     'A'             ;POSSIBLE ASC PSEUDO OP?
        JNZ     INSTR2          ;NO WAY
        CALL    COUNT           ;DECREMENT LINE COUNT
        JZ      EOLINE          ;IF END OF LINE
        CALL    GETBYT          ;GET NEXT BYTE
        CPI     'S'             ;ASC PSEUDO OP?
        JNZ     RECOV2          ;NO, RECOVER LAST TWO BYTES
        CALL    COUNT           ;DECREMENT LINE COUNT
        JZ      EOLINE          ;IF END OF LINE
        CALL    GETBYT          ;GET NEXT BYTE
        CPI     'C'             ;ASC PSEUDO OP?
        JNZ     RECOV3          ;NO, RECOVER LAST 3 BYTES
;
;       CONVERT ASC PSEUDO OP TO DB
        LXI     D,DBOP          ;OUTPUT A 'DB'
        CALL    PUTSTRING
        MVI     A,TAB           ;END OF INSTR FIELD
        CALL    OUTBYT          ;TAB TO NEXT COLUMN
;
;       HANDLE ASC OPERAND AND CONVERT DELIMITERS
        CALL    DELSP           ;DELETE EXTRA SPACES
        JC      EOLINE          ;IF END OF LINE
        STA     DELIM           ;SAVE THE ASC DELIMITER
        MVI     A,QUOTE         ;GET CPM DELIMITER
ASC2:   CALL    OUTBYT          ;OUTPUT IT
        CALL    COUNT           ;DECREMENT LINE COUNT
        JZ      EOLINE          ;IF END OF LINE
        LDA     DELIM           ;GET DELIMITER
        MOV     B,A             ;INTO B
        PUSH    B               ;SAVE B
        CALL    GETBYT          ;GET NEXT BYTE
        POP     B               ;RESTORE B
        CMP     B               ;DELIMITER?
        JNZ     ASC2            ;NO, KEEP GOING
        MVI     A,QUOTE         ;YES, GET CPM DELIMITER
        CALL    OUTBYT          ;OUTPUT IT
        JMP     COMMENT         ;REMAINDER MUST BE COMMENT
;
;       RECOVER LAST TWO BYTES (NOT ASC PSEUDO OP)
RECOV2: STA     TEMP2           ;SAVE LAST BYTE
        MVI     A,'A'           ;FIRST BYTE WAS AN 'A'
        CALL    OUTBYT          ;OUTPUT IT
        LDA     TEMP2           ;GET LAST BYTE
        JMP     INSTR2          ;CONTINUE NORMALLY
;
;       RECOVER LAST 3 BYTES (NOT ASC PSEUDO OP)
RECOV3: STA     TEMP2           ;SAVE LAST BYTE
        LXI     D,ASOP          ;WE KNOW LAST 2 BYTES WERE 'AS'
        CALL    PUTSTRING       ;SO OUTPUT THEM
        LDA     TEMP2           ;GET LAST BYTE
;
;       HANDLE REMAINDER OF INSTRUCTION FIELD
INSTR2: CALL    PASS            ;OUTPUT REST OF FIELD
        JC      EOLINE          ;IF END OF LINE
;
OPERAND:
        MVI     A,TAB           ;PROCESS OPERAND FIELD
        CALL    OUTBYT          ;TAB TO NEXT COLUMN
        CALL    DELSP           ;DELETE LEADING SPACES
        JC      EOLINE          ;IF END OF LINE
;
;       CHECK FOR '<' SPECIAL OPERATOR
OPER1:  CPI     '<'             ;LEFT BYTE MASK?
        JNZ     OPER2           ;NO WAY
        CALL    COUNT           ;DECREMENT END OF LINE COUNT
        JZ      EOLINE          ;IF END OF LINE
        CALL    GETBYT          ;GET NEXT BYTE
        CALL    PASS            ;OUTPUT REST OF FIELD
        JC      EOLINE          ;IF PREMATURE END OF LINE
        LXI     D,LBYTE         ;OUTPUT A SHIFT RIGHT EXPR
        CALL    PUTSTRING
        JMP     COMMENT         ;REMAINDER MUST BE COMMENT
;
;       CHECK FOR '>' SPECIAL OPERATOR
OPER2:  CPI     '>'             ;RIGHT BYTE MASK?
        JNZ     OPER3           ;NO WAY
```

```
        CALL    COUNT           ;DECREMENT LINE COUNT
        JZ      EOLINE          ;IF END OF LINE
        CALL    GETBYT          ;GET NEXT BYTE
        CALL    PASS            ;OUTPUT REST OF FIELD
        JC      EOLINE          ;IF END OF LINE
        LXI     D,RBYTE         ;THEN A MASK RIGHT EXP
        CALL    PUTSTRING
        JMP     COMMENT         ;REMAINDER MUST BE COMMENT

;       HANDLE REMAINDER OF OPERAND
OPER3:  CPI     QUOTE           ;IS IT A QUOTE?
        CZ      TOGGLE          ;YES, TOGGLE STRING FLAG
        CPI     '.'             ;IS IT A PERIOD?
        JZ      STRING          ;YES, CHECK IF PART OF STRING
        CPI     SPACE           ;IS IT A SPACE?
        JNZ     OPER4           ;NO, PROCESS IT
STRING: STA     TEMP            ;SAVE BYTE
        LDA     FLAG            ;IS CHAR WITHIN STRING?
        ORA     A
        JZ      COMMENT         ;NO, END OF OPERAND FIELD
        LDA     TEMP            ;YES, GET BACK BYTE
OPER4:  CALL    OUTBYT          ;OUTPUT BYTE
        CALL    COUNT           ;DECREMENT LINE COUNT
        JZ      EOLINE          ;IF END OF LINE
        CALL    GETBYT          ;GET NEXT BYTE
        JMP     OPER3           ;KEEP GOING
;
;
COMLINE:                        ;PROCESS COMMENT LINE
        MVI     A,';'           ;GET A SEMICOLON
        JMP     CLOOP           ;OUTPUT IT & REST OF LINE
COMMENT:                        ;PROCESS COMMENT FIELD
        MVI     A,TAB
        CALL    OUTBYT          ;TAB TO NEXT COLUMN
        MVI     A,';'           ;GET A SEMICOLON
        CALL    OUTBYT          ;OUTPUT IT
        CALL    DELSP           ;DELETE LEADING SPACES
        JC      EOLINE          ;IF END OF LINE
        CPI     '!'             ;EXCLAMATION MARK?
        JZ      CLOOP2          ;YES, DELETE IT
CLOOP:  CALL    OUTBYT          ;NO, OUTPUT IT
CLOOP2: CALL    COUNT           ;DECREMENT LINE COUNT
        JZ      EOLINE          ;IF END OF LINE
        CALL    GETBYT          ;GET NEXT BYTE
        CPI     '!'             ;EXCLAMATION MARK?
        JZ      CLOOP2          ;YES, DELETE IT
        JMP     CLOOP           ;KEEP GOING

PREMEND:                        ;PREMATURE END OF LINE
        MVI     A,CR            ;OUTPUT A CARRIAGE RET
        CALL    OUTBYT
EOLINE: MVI     A,LF            ;NORMAL END OF LINE
        CALL    OUTBYT          ;OUTPUT A LINE FEED
        JMP     PROCESS         ;PROCESS NEXT LINE
;
EOFMARK:
        MVI     A,'Z'-40H       ;GET A CPM EOF MARK
        CALL    OUTBYT          ;OUTPUT IT
        RET                     ;EXIT PROCESS
;
;
TOGGLE: PUSH    PSW             ;FLAG SET IF STRING IN QUOTES
        LDA     FLAG            ;TOGGLE FLAG
        CMA
        STA     FLAG            ;THAT'S ALL
        POP     PSW
        RET
;
;       DELETE EXTRA SPACES
DELSP:  CALL    COUNT           ;DECREMENT LINE COUNT
        JZ      DELSP2          ;IF PREMATURE END OF LINE
        CALL    GETBYT          ;GET NEXT BYTE
        CPI     SPACE           ;SPACE?
        JZ      DELSP           ;YES, DELETE IT
        ORA     A               ;RESET CARRY FLAG
        RET                     ;RETURN WITH BYTE
DELSP2: STC                     ;SET CARRY FLAG FOR EOL
        RET                     ;RETURN WITH FLAG SET
;
;       OUTPUT BYTES UNTIL END OF FIELD
PASS:   CALL    OUTBYT          ;OUTPUT BYTE
        CALL    COUNT           ;DECREMENT LINE COUNT
        JZ      PASS2           ;IF PREMATURE END OF LINE
        CALL    GETBYT          ;GET NEXT BYTE
        CPI     SPACE           ;END OF FIELD?
        JNZ     PASS            ;NO, KEEP GOING
        ORA     A               ;RESET CARRY FLAG
        RET                     ;RETURN WITH SPACE
PASS2:  STC                     ;SET CARRY FLAG FOR EOL
        RET                     ;RETURN WITH FLAG SET
;
COUNT:  LDA     EOLCNT          ;DECREMENT END OF LINE COUNT
        DCR     A
        STA     EOLCNT          ;THAT'S ALL
        RET
;
;***********************************
;       P U T S T R I N G         *
; ROUTINE TO OUTPUT A STRING TO DISC *
;***********************************
;
PUTSTRING:
        LDAX    D               ;GET A BYTE
        CPI     '$'             ;TERMINATOR?
        RZ                      ;YES, RETURN
        PUSH    D               ;SAVE D&E
        CALL    OUTBYT          ;NO, OUTPUT BYTE
        POP     D               ;RESTORE D&E
        INX     D               ;BUMP POINTER
        JMP     PUTSTRING       ;KEEP GOING
;
;*******************
;  CANNED STRINGS:  *
;*******************
;
ASOP:   DB      'AS$'           ;STRING TO RECOVER AS STRING
DBOP:   DB      'DB$'           ;STRING TO REPLACE ASC OP
RBYTE:  DB      ' AND 0FFH$'    ;STRING TO REPLACE ">" OP
LBYTE:  DB      ' SHR 8$'       ;STRING TO REPLACE "<" OP


;*********************************
;          O U T B Y T          *
;   ROUTINE TO OUTPUT A BYTE    *
;                               *
;  INPUT:     A=BYTE            *
;*********************************

OUTBYT: STA     TEMP            ;SAVE BYTE
OUT1:   LXI     H,OBUF+128
        XCHG                    ;BUFFER END ADDR IN DE
        LHLD    OUTPT           ;CURRENT ADDR IN HL
        CALL    CPHL            ;TEST FOR END OF BUFFER
        JZ      OUT2            ;YES, WRITE
        LDA     TEMP
        MOV     M,A             ;STORE DATA BYTE IN BUFFER
        INX     H               ;BUMP BUFFER POINTER
        SHLD    OUTPT           ;SAVE BUFFER POINTER
        ORA     A
        RET                     ;EXIT
OUT2:   LXI     D,OBUF          ;POINT TO OUTPUT BUFFER
        LXI     H,TBUF          ;TEMP BUFFER
        MVI     B,128
        CALL    MOVE            ;COPY BUFFERS
        MVI     C,WRITE         ;WRITE FUNCTION
        LXI     D,DEST          ;OUTPUT FCB
        CALL    BDOS            ;ISSUE WRITE
        CPI     0FFH
        JZ      ERROR3
        CPI     2
        JZ      ERROR4
        CPI     1
        JZ      ERROR5
        LXI     H,OBUF          ;RESET POINTER
        SHLD    OUTPT
        JMP     OUT1            ;CONTINUE

;*********************************
;          G E T B Y T          *
;   ROUTINE TO READ A BYTE      *
;                               *
;  OUTPUT:    A=BYTE            *
;*********************************

GETBYT: LXI     H,IBUF+128
        XCHG                    ;BUFFER END ADDR. IN DE
        LHLD    INPTR           ;CURRENT POINTER IN HL
        CALL    CPHL            ;TEST FOR END OF BUFFER
        JZ      GETB2           ;YES, READ
GETB1:  MOV     A,M             ;GET BYTE
        INX     H               ;BUMP POINTER
        SHLD    INPTR           ;SAVE POINTER
        ORA     A               ;RESET CARRY
        RET
GETB2:  MVI     C,READ          ;READ FUNCTION
        LXI     D,SOURCE        ;FCB ADDRESS
        CALL    BDOS            ;ISSUE READ
        CPI     0               ;ERROR?
        JNZ     ERROR2          ;YES
        LXI     D,TBUF          ;POINT TO TEMP BUFFER
        LXI     H,IBUF          ;INPUT BUFFER
        MVI     B,128
        CALL    MOVE            ;COPY BUFFER
        LXI     H,IBUF          ;RESET BUFFER POINTER
        SHLD    INPTR
        JMP     GETB1           ;CONTINUE

;*********************************
;          M O V E              *
;  ROUTINE TO MOVE BLOCKS OF DATA *
;*********************************
MOVE:   LDAX    D               ;GET BYTE
        MOV     M,A             ;STORE BYTE
        INX     H
        INX     D               ;BUMP POINTERS
        DCR     B               ;DECREMENT COUNT
        JNZ     MOVE            ;LOOP
        RET

;*********************************
;          C P H L              *
;  ROUTINE TO COMPARE HL VS DE   *
;*********************************

CPHL:   MOV     A,H
        CMP     D
        RNZ
        MOV     A,L
        CMP     E
```

Continued on page 48

access addend ALGORITHM algebra analog application ai

BY DENNIS ALLISON

MERGE SORT REVISITED

My old friend, Sylvan Rubin, sent on to me a major improvement in the nonrecursive merge program published in *DDJ* #44. Rather than build the merged list backwards and then, later, reverse it, he builds it in the right order to begin with. This cuts the number of assignments per data element from 9.5 to 4.5 assuming randomly distributed data. He also improved the control structure a bit to eliminate some tests. All in all a much better program.

```
/* MERGE algorithm using linked lists.
/* written in C.  24 Mar 80   Sylvan Rubin, Algorithmic Eng.

merge (i,j) int i,j; { int t,k,m;
     if(a[i]<a[j]) { m=i; i=j; j=m;}
     t=i; k=t;
     i=link[i];
     while(i!=0) {
          if(a[i]<a[j]) {m=i; i=j; j=m;}
          link[k]=i; /* link next largest element to tail of
          k=i;                                      merged list
          i=link[i];
     }
     link[k]=j; /* link remaining list to tail of merged list
     return (t);
}
```

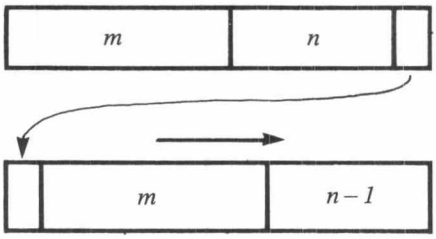
RECURSION VERSUS ITERATION

Sylvan comments that he doesn't think that we should publish the recursive form of algorithms since their translation into assembler or BASIC is a nontrivial task. I don't completely agree. The purpose of this column is to expose some of the fundamental mechanisms behind algorithmic solutions and to gain some appreciation of the aesthetics. That goal must involve recursion in some fashion. In the real world, here is a tradeoff. Recursive code is usually smaller and shares temporary random access memory with other disjoint computations; interactive code is usually longer, uses dedicated storage, and runs much faster. Which form of algorithm one wants to use depends upon the application.

INTERCHANGE AGAIN

Richard Karp pointed out two other solutions to the interchange problem presented last time. And I decided I should have said something about the shift-register version.

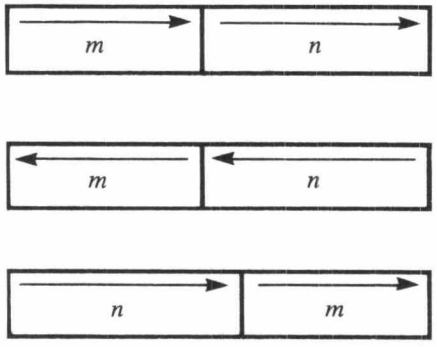
The shift-register version requires $m * n$ operations. The idea is to start at the shortest segment, pluck out the element at the end, move everything over one (to the right or the left depending upon which end the shortest segment appears) stick in the selected element and repeat until the interchange is completed. Graphically the procedure is



Another approach to the interchange problem uses a swap primitive. It works by simply exchanging the smaller segment to put it into its proper position and then applying the same procedure to the remaining segment which itself requires an interchange. A C program which does this is given below:

```
int a[80];

interchange (m,n) int m,n; {
        int i,j,k;
        i = 1;
        n = n-m;  /* n and m are segment sizes */
        for(;;) {
                k = (m>n)?n:m;
                if(k==0) return;
                swap(i, i+m+n-k, k);
                if(m<n) {
                        n =- m;
                } else {
                        i =+ n;
                        m =- n;
                }
        }
}


swap ( i, j, k ) int i,j,k; {
        register int t;
        while(k--) {
                t = a[i]; a[i++] = a[j];
                a[j++] = t;
        }
}
```

But the truly elegant solution is this one attributed by E.W. Dijkstra to J. Ichbiah. One needs an array reversal primitive – an easy thing to do with a single temporary. The algorithm is best seen graphically:



One first reverses each of the segments independently and then reverses the entire array. Not as efficient as the group theoretic scheme, but it does have a pleasing aesthetic quality.

BIT REVERSAL

There's never really much call for an algorithm to reverse the bits in a word. Nonetheless, this algorithm, due to C. Strachey [CACM 4:3 961 p.146] is interesting because of its aesthetic beauty.

To see how it works it's best to pretend we have a double length register and write out the bits in the order we have them and the order we want them. We then write down the shifts required to get each bit from where it is to where it should be and write down the binary representation for that number. We then program a select-bits, shift, and merge sequence for each power of two, thus generating the result. Figure 1 shows the whole procedure worked out for a 16-bit operand.

A double length register is not necessary. One can do it in place by observing that after the selection one can shift both operands, one right and the other left. Further, one can use the same mask to handle the selection. The C program below will reverse the bits of a 16-bit word. Note that only one more step would be needed for a 32-bit word since the complexity grows as the logarithm of the number of bits.

There are, of course, several alternative ways to accomplish the same task. The obvious one of performing the reversal a bit at a time is costly to perform in place and is linear in the length of the word. A 16-bit word requires 16 operations to reverse the bits. Anyone knowing a good application for bit-reversed data won't suggest the FFT (which leaves its results in reverse binary order) since there are better ways to reorganize its result than direct selection.

On a byte-oriented processor it might be better to have a table of reversed bytes and simply look up each byte of the number and then reverse the bytes. While this approach is also linear in the number of bits, it has the advantage of being small and efficient of space and time.

```
p' o' n' m' l' k' j' i' h' g' f' e' d' c' b' a' a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p
                                                1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31  shift numbers

                                                0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  shift 16
                                                0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1  shift 8
                                                0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1  shift 4
                                                0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  shift 2
                                                1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  shift 1

                                                a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  initial position
                                 i  j  k  l  m  n  o  p  a  b  c  d  e  f  g  h  .  .  .  .  .  .  .  .  shift 16
                           m  n  o  p  i  j  k  l  e  f  g  h  a  b  c  d  .  .  .  .  .  .  .  .  .  .  shift 8
                     o  p  m  n  k  l  i  j  g  h  e  f  c  d  a  b  .  .  .  .  .  .  .  .  .  .  .  .  shift 4
               p  o  n  m  l  k  j  i  h  g  f  e  d  c  b  a  .  .  .  .  .  .  .  .  .  .  .  .  .  .  shift 2
p  o  n  m  l  k  j  i  h  g  f  e  d  c  b  a  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  shift 1
```

Figure 1. Schematic Bit Reversal

```
int mask[] {
        0xFFFF, 0x5555, 0x3333, 0x0F0F, 0x00FF
};


reverse( val ) {
        register int i,j,k;
        i = val;
        j = 8; k = 4;
        while(j) {
                i = ((i&mask[k])<<j)|((i>>j)&mask[k]);
                j =>> 1;
                k--;
        }
        return(i);
}
```

Continued from page 46

```
        RET

;**********************************
;       W E O F                  *
;  ROUTINE TO FILL UP LAST RECORD *
;  WITH CP/M EOF MARKS            *
;**********************************

WEOF:   LXI     H,OBUF+128      ;END OF BUFFER
        XCHG
        LHLD    OUTPT           ;WHERE WE ARE
        CALL    CPHL            ;SAME?
        JZ      OUTBYT          ;YES, WRITE LAST RECORD
        MVI     A,'Z'-40H       ;NO, GET A CPM EOF MARK
        CALL    OUTBYT          ;OUTPUT IT TO BUFFER
        JMP     WEOF            ;KEEP GOING
;
;**********************************
;       C L O S E F I L E        *
;  SUBROUTINE TO CLOSE FILES     *
;**********************************
;
CLOSEFILE:
        MVI     C,CLOSE         ;CLOSE FUNCTION
        LXI     D,DEST
        CALL    BDOS            ;CLOSE DESTINATION FILE
        MVI     C,CLOSE         ;CLOSE FUNCTION
        LXI     D,SOURCE
        CALL    BDOS            ;CLOSE SOURCE FILE
        RET
;

;
;*****************************
;       ERROR HANDLERS       *
;*****************************
;
ERROR1: LXI     D,MSSG1
        JMP     ERRMSG
ERROR2: LXI     D,MSSG2
        JMP     ABORT
ERROR3: LXI     D,MSSG3
        JMP     ERRMSG
ERROR4: LXI     D,MSSG4
        JMP     ABORT
ERROR5: LXI     D,MSSG5
        JMP     ABORT
ERROR6: LXI     D,MSSG6
        JMP     ERRMSG
ERROR7: LXI     D,MSSG7
ERRMSG: CALL    MESSAGE
        JMP     FIN             ;RETURN TO CPM
ABORT:  CALL    MESSAGE
        JMP     ABEND           ;CLOSE FILES & RETURN TO CPM
;
;*****************************
;       M E S S A G E        *
;  SUBROUTINE TO OUTPUT MESSAGE *
;*****************************
;
;
MESSAGE:
        MVI     C,PRINT         ;PRINT FUNCTION
        CALL    BDOS
        RET
;
;***************************
;       MESSAGES:         *
;***************************
;
MSSG0   DB      'ALS-8 to CP/M File Converter - ver 1.6',CR,LF
        DB      'by Ed Elizondo',CR,LF,'$'
MSSG1   DB      'Source file not on disk',CR,LF,'$'
MSSG2   DB      'Read past end of file',CR,LF,'$'
MSSG3   DB      'Directory full',CR,LF,'$'
MSSG4   DB      'Disk full',CR,LF,'$'
MSSG5   DB      'Error in extending file',CR,LF,'$'
MSSG6   DB      'No source file indicated',CR,LF,'$'
MSSG7   DB      'No destination file indicated',CR,LF,'$'
MSSG8   DB      'Converting file',CR,LF,'$'
MSSG9   DB      'Done',CR,LF,'$'

;*************************
;       DATA BUFFERS:    *
;*************************
;
TEMP:   DS      1               ;TEMPORARY SAVE BYTE
TEMP2:  DS      1               ;ANOTHER ONE
DELIM   DS      1               ;ASC DELIMITER SAVE BYTE
FLAG:   DB      0               ;SWITCH FOR QUOTED STRINGS
EOLCNT: DB      0               ;END OF LINE COUNT
DEST:   DS      33              ;DEST FILE FCB
;
OUTPT:  DW      OBUF            ;OUTPUT POINTER
INPTR:  DW      IBUF+128        ;INPUT POINTER
IBUF:   DS      128             ;INPUT BUFFER
OBUF:   DS      128             ;OUTPUT BUFFER
;
OLDSTK: DS      2               ;OLD STACK POINTER
        DS      40H             ;STACK AREA
NEWSTK: EQU     $
;
        END

A>^L
```

Letters

AN IMPROVEMENT TO ZANT'S MODULAR PROGRAMMING

Dear Suzanne:

The article on modular programming, authored by me and published in *DDJ* #38, contains a program for appending Applesoft programs. The published version of the program works on tape-oriented systems and on systems using DOS 3.1, but will not work correctly on later versions of Apple's DOS.

The problem is caused by DOS 3.2 recalculating the chain pointers for Applesoft statements each time a program is loaded. The Append program must therefore be modified to remove the instructions that perform this same function. A listing of the modified program is enclosed.

Sincerely,
Robert F. Zant
Associate Professor of
Information Systems

North Texas State University
Denton, Texas 96203

```
]ASM
                        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 *  01-07-80
                        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 *
                        1190        .OR $308
0308- 00                1200 FLAG   .DA #*-*
0309- AD 08 03          1210        LDA FLAG
030C- D0 24             1220        BNE SECOND
030E- EE 08 03          1230        INC FLAG
0311- A5 67             1240        LDA BEGINL
0313- 8D 04 03          1250        STA SAVEL
0316- A5 68             1260        LDA BEGINH
0318- 8D 05 03          1270        STA SAVEH
031B- A0 00             1280 AGAIN  LDY #$00
031D- B1 67             1290        LDA (BEGINL),Y
031F- AA                1300        TAX
0320- D0 06             1310        BNE CONT
0322- C8                1320        INY
0323- B1 67             1330        LDA (BEGINL),Y
0325- D0 04             1340        BNE CONT1
0327- 60                1350        RTS
0328- C8                1360 CONT   INY
0329- B1 67             1370        LDA (BEGINL),Y
032B- 85 68             1380 CONT1  STA BEGINH
032D- 86 67             1390        STX BEGINL
032F- 4C 1B 03          1400        JMP AGAIN
                        1410 *
0332- CE 08 03          1420 SECOND DEC FLAG
0335- AD 04 03          1430        LDA SAVEL
0338- 85 67             1440        STA BEGINL
033A- AD 05 03          1450        LDA SAVEH
033D- 85 68             1460        STA BEGINH
033F- 60                1470        RTS
                        1480        .EN
SYMBOL TABLE
BEGINL 0067   BEGINH 0068   SAVEL  0304
SAVEH  0305   FLAG   0308   AGAIN  031B
CONT   0328   CONT1  032B   SECOND 0332
```

MAYBE JUST GALACTIC

Dear *Dr. Dobb's*,

In *DDJ* # 43, Bud Gramer states that CP/M is "the only true universal operating system." Others have also made this statement. Obviously a "universal" operating system must run on (almost) everything. Therefore CP/M (or anything else) will not be a universal operating system until there are versions of it for the 6800, 6502, 6809, Z-8000, 68000, etc., etc.

One approach to a really universal operating system would be to pick a suitable high-level language such as C or PASCAL, and write most of an operating system and the required utility programs in that language. For each target computer, a small amount of assembly language code would be needed for device drivers and interrupt handlers. Most of the system would be the same (or nearly so) for all computers.

While Mr. Gramer is "sick" of hearing about certain non-CP/M systems, I am tired of the attitude of much of the microcomputing community that the 8080, Z-80, and S-100 bus are the only things that exist.

Sincerely,
Jim Howell

5472 Playa Del Rey
San Jose, CA 95123

A FEW THINGS . . .

Dear Suzanne,

I would like to inform your readers of an address correction necessary for their responses to the article "Some Useful Subroutines for a Z-80 Assembler" in *DDJ* #44.

I have received a significant response to that article, but my mailing address has changed since the article was submitted. For information concerning the subroutine library, or for a copy of it, requests should be sent to: J. W. Carter, 3538 Deerwood, Memphis, TN 38111.

I would like to also add another opinion to the "advertising controversy." I consider it unfortunate that *Dr. Dobb's* finds it necessary to include advertising in the publication. *Dr. Dobb's* is one of the few journals in this field with such a select, and consequently devoted, audience. With no pretty pictures and pie-in-the-sky dreams generated by advertising, the reader receives only top-quality articles in the field. Also authors, knowing a select audience exists are encouraged to provide good quality manuscripts. With advertising entering the picture, I see this philosophy decaying; for with advertising comes additional profit which is usually passed on to the *author*. I would expect that, with advertising, quality authors will tend to send manuscripts to other journals which provide fiscal compensation.

I would rather see you raise the subscription rate and include more of the material you have been publishing, than see *Dr. Dobb's* go the way of the modern majority.

Sincerely,
John W. Carter
Computer Systems Technology

Memphis State University
Memphis, Tennessee 38152

REMARKS AS ENTRY POINTS

Dear *DDJ*:

Bruce Thompson's letter in *DDJ* #45 states that jumping to a remark in a program is a "serious error in programming style," but he fails to recognize that cramming as much active code in a given amount of memory as possible is not the only programming consideration. In many cases, readability and ease of modification may be more important.

As a professional programmer, I know that memory is usually cheaper than software, and have adopted a style that is totally contrary to Mr. Thompson's. In my programs, remarks are *defined* as entry points either to in-line code or to subroutines. The advantage to this style is twofold: First, since you know that a remark is an entry point, you will leave it alone and won't accidentally change it when performing modifications to the program. You can add more code at the beginning of a routine without having to shuffle line numbers. Second, it is a tremendous aid in promoting readability to a program. When you see the statement GOTO 1000, you can simply look at the remark at line 1000 to see what happens when you get there without having to wade through the code.

Not all entry points in my programs are remarks since this would be wasteful in trivial cases, but all remarks are entry points major logical procedures. Once it is understood that this is the case, it adds a great deal of flexibility to the program.

Very Truly Yours,
Burks A. Smith

DATASMITH Micro Software Systems
15501 West 109th St.
Lenexa, Kansas 66219

IMPROVEMENTS TO APPLE USER INTERFACE

Dear Suzanne,

I was very pleased to see the User Interface to the APPLE II Program Renumbering in *DDJ* #42.

I found some errors and a possible error.

The errors:

1. Line 106 of the code is out of place. It should be between lines 102 and 103. 2. At line 432 the $94B0 should be 94CO. 3. At line 551 on page 50 the code should read 20 58FC.

The possible error – it is at least an omission – is at line 563 and thereafter: the ASCII bytes for the headers. Perhaps it is supposed to be obvious that they are not there. But a beginner might be very perplexed, and certainly would be dispmayed at not having the program work properly. Below is the correct code:

```
957E  C5 CE
9580  D4 C5 D2 A0 CC C9 CE C5
9588  A0 CE D5 CD C2 C5 D2 A0
9590  C4 C5 C6 C1 D5 CC D4 8D
9598  A0 A0 CE C5 D7 A0 D3 D4
95A0  C1 D2 D4 A0 A0 A0 A0 A0
95A8  A0 A0 A0 A0 B1 B0 B0 A0
95B0  A0 A0 A0 A0 CE C5 D7 A0
95B8  C9 CE C3 D2 C5 CD C5 CE
95C0  D4 A0 A0 A0 A0 A0 B1 B0
95C8  B0 A0 A0 A0 A0 A0 D2 C1
95D0  CE C7 C5 A0 D3 D4 C1 D2
95D8  D4 A0 A0 A0 A0 A0 A0 A0
95E0  A0 A0 B0 A0 A0 A0 A0 A0
95E8  D2 C1 CE C7 C5 A0 C5 CE
95F0  C4 A0 A0 A0 A0 A0 A0 A0
95F8  B6 B5 B5 B3 B5 A0 A0 A0
```

A few comments. If you are not going to use the program with a disk (because you might not have one, for instance) change each of the three bytes in line 549 to EA(NOP). If you would like to have different default values for the NEW START and NEW INCREMENT change the byte at $9424 to the desired value. For example, I wanted my NEW START to be Line 10 and my NEW INCREMENT 10 also. I changed the value at $9424 to $0A. (I see no way of changing one of those default values and not the other.) After I made that change I then changed the ASCII data for the headers so that 'NEW START 10' and 'NEW INCREMENT 10' are displayed.

That's all. Hope this is helpful .

Yours,
Stephen E. Bach-AA4B

Rt 2 Box 50A-1
Scottsville, VA 24590

ON DIABLO

Dear Suzanne:

The enclosed program is based around a Z-80 USR call that calculates the advance necessary for a Diablo printer to print a string any desired width. The code has been expanded somewhat to make it easier to install on other machines. The last line of code could be replaced by a jump to whatever location serves the same purpose. If the routine is used as is, the defaults are a 72 character line with a 12 point type daisy wheel.

Yours truly
L. Barker — Chicago, IL

```
10 ' Variable printing for a Diablo
20 CLEAR 400
30 DEFINT I,J
40 DEFUSR0 = &HA000
50 W = 41123!   :' Width to be printed (in characters)
60 FOR I= 0 TO 176
70       READ J
80       POKE 40960!+I, J
90 NEXT
100 INPUT "Print what file (.TXT)";F$
110 INPUT "Print file what width ";J
120 IF J < 20 OR J > 132 GOTO 110
130 POKE W, J
140 INPUT "Advance (= 120/point) ";I
150 IF I=10 OR I=12
        THEN POKE W+1, I
        ELSE GOTO 140
160 OPEN "I",1, F$+".TXT"
170 LINE INPUT #1, A$
180 IF INSTR(A$,"=END") > 0 GOTO 220
190 IF LEN(A$) < .75 * J
        THEN LPRINT A$;
        ELSE A$ = USR0(A$)
200 LPRINT
210 IF NOT EOF(1) GOTO 170
220 CLOSE
230 END
240 ' Program loads at 0A000H   (40960)
250 DATA 235, 70, 35, 94, 35, 86, 33,  0,  0, 34,159,160, 34,161,160,213
260 DATA  58,163,160, 95, 84, 58,164,160,205,132,160, 34,157,160,225, 58
270 DATA 161,160,184, 40, 77,229, 60, 50,161,160,237, 91,157,160,205,132
280 DATA 160, 14, 33,175, 24,  5,184, 56,  1,144, 63,237,106,235,237,106
290 DATA 235, 23, 13, 32,241, 31,237, 91,159,160, 34,159,160,163,237, 82
300 DATA  35, 58,162,160,235,225,187, 40, 18, 14, 27,205,165,160, 14, 31
310 DATA 205,165,160, 75,205,165,160,123, 50,162,160, 78, 35,205,165,160
320 DATA  24,173, 14, 27,205,165,160, 14, 31,205,165,160, 58,164,160, 60
330 DATA  79,195,165,160, 33,  0,  0,229, 14,  8, 41,227,237,106,227,  7
340 DATA  48,  6, 25, 48,  3,227, 35,227, 13, 32,239,209,201,194, 65,210
350 DATA  75, 69,210, 72, 10
360 DATA  58,  0,207,230,  2, 40,249,121, 50,  1,207,201
370 '
380 ' The last data line is to send register C to the line printer.
390 '
400 '   LINEP:  LD      A,(PORT)
410 '           AND     2
420 '           JR      Z,LINEP
430 '           LD      A,C
440 '           LD      (PORT+1),A
450 '           RET
460 '   ORG     0CF00H                  ; modify as system requires
470 '   PORT:
```

A NEW LISP INTERPRETER

Dear Suzanne:

This is to announce availability of a new version of the LISP interpreter which appeared in *DDJ* # 30. This program is available from me on 8 inch, single density IBM format CP/M diskette for $10.

GCLISP is a version of the *Dr. Dobb's* LISP which has been modified to run under CP/M (by addition of appropriate drivers, has improved behavior when presented with user program errors and can collect list garbage when memory is exhausted. In addition, the code has been rearranged into a slightly more logical format (all variables have been collected at the end, data statements done more compactly) and initialization has been made more understandable. Provisions have also been made for saving the system after defining new (interpreted) functions.

CP/M features: The interpreter now sets up its own stack. There is an assembly time option to use CP/M line buffered input. The advantage of line buffered input is that LISP doesn't see the line until carriage return is typed, so the line can be corrected during type-in. The assembly also generates an equate for PAGSAV which is the number of pages for a CP/M save command to allow using DEFINE to create a set of functions and constants, then ctrl-C back to CP/M and save the result. The interpreter uses the CP/M string type routine to send error messages.

Improved control: In the original interpreter, errors in a user program caused the interpreter to abort. In this version, most errors result in a message and return to the EVALQUOTE driver. In addition, EVAL checks for ctrl-D from the terminal to abort long computations. The EVALQUOTE driver now prompts for input with an underscore.

The interpreter has one new function, EXIT, of zero arguments which returns control to CP/M.

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

ELF TINY MONITOR

Dear *Dr. Dobb's*:

Here's an Elf Tiny Monitor for the Netronics Elf II that has proven to be a big timesaver for me. The Elf II is virtually identical to the 1802 based Elf introduced in a series of Popular Science articles by RCA's Joseph Weisbecker starting in August 1976. The monitor presented here is an improvement over a similar program in the March 1977 issue because it is shorter, performs the same functions and displays memory location. I'd like to hear from anyone having a shorter program with equal features.

Load the following program starting at memory location 00HEX. Then turn off the RUN, LOAD, and Memory Protect switches.

```
F8  1D  AF  EF  6C  30  1C  7B  3F  08
6C  AE  37  0C  8E  5F  EF  64  2F  3F
13  EE  C5  6C  C4  64  30  0C  30
```

1. To start execution at any memory location press the two HEX keys that define the location and turn on the RUN switch. The monitor writes into location 1DHEX, so user programs must start after this.
2. To examine memory start execution at location 08HEX according to instructions in (1) above, then press and release the I key. The memory location (address) will be displayed. Press and hold the I key to examine memory contents, then release the key to display the next address. Continue to press and release the I key to examine successive memory addresses and contents.
3. To change memory start execution at location 07HEX and follow the same initial sequence as in (2). The Q light will come on indicating the memory may be altered. After the memory location is displayed press the two HEX keys that define the memory contents (data) to be entered and press the I key. When the key is released the next address will be displayed and the monitor is again ready for input data.
4. To change modes turn off the RUN switch and begin anew.

Sincerely,
R.C. Briggs — 468 Velarde Street, Mountain View, CA 94041

THE LOW DOWN ON TOP-DOWN

Dear Editor:

The letter by John R. Culleton, Jr. in *DDJ* #43 claims that top-down programming is impossible in PASCAL (in contrast to COBOL). I have used PASCAL heavily for five years, and I always thought that on my best days I was working in a top-down fashion, and quite successfully. The definition of "top-down" must need clarifying, so here is mine.

"Top-down" programming is a method of designing and writing programs which proceeds from the most general to the most specific aspects of the problem to be solved. Typically, the first step will be to write a skeleton program (or a narrative outline) of what will be done. Subsequent steps, as many as are necessary, provide detailed expansion of things that are only outlined or named at a higher (previous) level. The task is done when everything has been reduced to a form which can be executed by a computer or which can be automatically translated to such a form.

Niklaus Wirth calls this "stepwise refinement" and has written two excellent books showing its use with PASCAL. It can be used with any language. With none is it impossible because it is almost independent of the qualities of the language used. One does not program *in* a language but *into* a language.

Can Mr. Culleton define "top-down" in a way that makes PASCAL unsuitable? And if so, can he defend the usefulness of such a definition in the multi-lingual world we all inhabit?

Sincerely,
David Rowland

6440 Regent St.
Oakland, Calif. 94618

(Continued from page 13)

be able to act as a front end processor, which will handle all the communications chores for a host machine. Once this is running, we can see some very powerful and exciting possibilities for distributed processing!

The protocol currently used is a simple "Aloha" type random access system. It has been enhanced however and uses carrier sense and priority acknowledgments. Since the current protocol will probably be defunct by the time it is published, here is the new version outline:

The Format

SYN			
SYN			
SOH			
Destin	Call	6 bytes	
Destin	Node	2 bytes	
Origin	Call	6	
Origin	Node	2	
Service	Flag	1	(bit mapped)
Number	Flag	1	(bit mapped)
Length	Byte	1	(bit mapped)
Data		0 to 256 max	
Checksum		2 bytes	

The Protocol can almost be derived by looking at the format of a packet. It is carrier sensed, that is, stations don't transmit if they hear another station on the air. Priority acknowledged, i.e., if a station hears a packet and the checksum is correct, it is presumed that the destination station also got it correctly and that an acknowledgment will be forthcoming. The transmission of a waiting packet is delayed to allow the ack to go by unhindered.

Where a station fails to get an ack to a transmitted pac, it will try again, up to three times. The interval between retries is random to avoid two stations continually clobbering each other's pacs. All pacs are preceded by an ASCII SYN (synch) byte and the ASCII Start of Header characters. The Service flag is used to indicate by setting a bit high (1) whether the packet is an ack, a repeated pac or ack, whether this is a file transfer, and ASCII or binary data field, with a couple of bits to spare. The bits are arranged as follows in this flag:

Bit	7	6	4	3	2	1	0
	A	R	F	B	?	?	? ?

Where
A = Ack
R = Pac or Ack from the repeater
F = File transfer in progress
B = Data contains binary coded info
? = Spare for future use

Note: there was a terrific temptation to rearrange the flags to read as B A R F, but in software the ARFB series was easier.

The repeat flag is used to prevent a station from receiving two packets if he hears both the origin station and the destination. The computer will respond to and keep the one R flagged, and ignore the other. Don't forget the repeater transmitted pac will get there after the original since it is store and forward. The file flag is used to indicate that the receiving station should file the packets sequentially and not allow other packets to intrude in the file. You can imagine what getting a "Hi Bob" pac would do if it was inserted in the middle of a program I was receiving. Files are also written to disk on the receiving computer. The binary flag is used when a file (program etc.) is binary coded as this would allow us to apply offset loading etc., to the file. It will also allow a program to "come up running" on a user's machine.

The next flag in the sequence count is used to keep packets composing a file in the right order. It will also be used eventually to implement a virtual circuit where a certain number of pacs will be allowed outstanding without acks. This will allow for improved throughput with an ACK to NUMBER (X) arrangement. The length byte is obvious: it simply tells the computer how long the packet is. The only restriction is that packets cannot have more than 256 bytes in that data field. The last two bytes are the checksum which is used to establish that the packet has been received error free. This is currently done in software but we expect to do it in hardware with the Fairchild 9401 CRC generator checker chip in the final version.

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 47 | August 1980 | Volume 5, Issue 7

A REFERENCE JOURNAL FOR USERS OF SMALL COMPUTERS

ZX65: Simulating a Micro 4
Richard M. Kruse

Clean Starts for COSMAC 1861 Video Output 14
Gary H. Price

Indirect Addressing for the 8080 and Z80 22
Anthony Skjellum

A Z-80 Circuit for the H-8 20
Ted T. Stowe

Harlie: A "No-Programming" Computer 11
J. L. Lawrence and Jay Sprenkle

...and, as always, even more

ZX65: Simulating a Micro

BY RICHARD M. KRUSE
6221 Woodlow Drive
Wichita, KS 67220

ABSTRACT

This article gives complete details, including source code, of an assembly language program which functionally simulates the operation of a 6502 microprocessor device using a Z80 based host computer system. Under control of ZX65, a wide variety of 6502 software may be directly executed in any of several modes. Useful for 6502 software development as well as for its tutorial value, ZX65 should prove to be a significant addition to the Z80 user's software library. Occupying only 4K Bytes of system memory, ZX65 is sufficiently flexible to be readily adapted to system environments other than the one described.

Having logged many hours of software development work on my Z80-based system, I recently decided that I was up to a particular challenge: using my system as a development tool for a totally different microprocessor; specifically, the 6502. This decision, I must confess, was not altogether based on a burning lust for knowledge, but rather on a more practical requirement: I had become involved in the development of a 6502-based device, and had no ready access to a 6502 development system. I decided that a simple cross-assembler wasn't good enough, and set out to write a 6502-to-Z80 object code translator/simulator/interpreter.

The program described here, which I have dubbed ZX65, is the result of my efforts to generate a software package to perform this task. It is useful not only for the intended purpose of software development, but also for the rewarding experience of exploring a whole new field of good software written for a processor other than one's own.

ZX65 operates as an interpretive simulator; that is, a simulated 6502 CPU is maintained in system memory, and each encountered 6502 instruction is decoded and processed interpreter-fashion to properly act upon the simulated CPU registers. The entire package resides in just over 4K Bytes of high system memory, leaving the remaining portion, including the all-important (for 6502) base page, free for program and data storage. Note that CP/M serves only as a loader and is not used thereafter. In fact, while the program loads normally into CP/M's Transient Program Area, it immediately relocates itself into high memory, overlaying both the CCP and the BDOS portions of CP/M. The user I/O linkages (CBIOS), however, remain intact and are used by the package to perform disk and console functions. The functions which must be provided by the system are a subset of those required by CP/M and should thus be readily available. These functions are all accessed via a jump table within the interpreter and thus can be readily changed to accommodate different system requirements (See Table 1). Parameters must be passed to the drivers in register "C" (register pair "BC" for item six) and data, if any, is returned in the accumulator.

ZX65 is referred to as a package since it includes three distinct function groups. The first and largest is the interpreter/simulator itself. The second group is an elementary monitor providing the functions of memory examine and modify. The third group consists of a self-contained, elementary disk operating system (DOS) (Note: ZX65 files are not CP/M compatible). The normal configuration assumes a two-drive system, with the system diskette on drive A, and all internal disk access routed to drive B (Modification for a single drive system could easily be done, requiring the user to switch diskettes after the package is loaded). The package is written in standard Zilog/Mostek source code and is quite heavily commented. Much use is made of the Z80's special instructions and index registers, and operation on an 8080 is definitely not possible without extensive modification. Although the commented listing should guide you through the program's operation without too much trouble, a brief description of the interpreter portion is in order. A study of the small but powerful instruction set of the 6502 will reveal certain patterns in the op codes, as also occurs with most other processors. The eight "BRANCH ON CONDITION" codes, for example, all have zero as a low-order nybble, while the high-order portion is always an odd value.

Another pattern, not so obvious, is that every instruction having an odd-valued op code uses multiple addressing modes. These and other such patterns are exploited by the program. Even so, much of the code is devoted solely to decoding the op code and transferring control to the proper instruction processor. The code beginning at "STEP" (line 241) performs the initial decoding, and fully decodes certain instructions such as "RTI", "JMP", and a few others. In most cases, decoding proceeds to a point where the instruction is known to be part of a group, such as the branch group described above. In these cases, look-up tables are used, but not always in the same way. The branch group and the flag set/reset group, for example, derive mask patterns from the tables which are used for testing, setting, or resetting the condition flags.

In the case of multiple addressing mode instructions, however, the tables serve an entirely different purpose. The table contents are, in these instances, decoded into addresses for indirect jumps; first to the addressing mode processors, and then to the actual instruction processors. Each processor performs its instruction as required. The "ADC" instruction, as an example, causes a byte of data to be fetched from memory or from the object code, depending on the addressing mode. After setting the Z80 carry flag to match the simulated 6502 carry flag, this data byte is added to the value currently in the simulated accumulator, and the result is stored there. The simulated carry flag is then updated to again match the Z80 carry, and the simulated sign, overflow, and zero flags are set or reset according to the results in the accumulator (In this case, the overflow flag is set or reset according to a logical exclusive "OR" of bits six and seven). Considerable care has been given to treatment of the flags, since Z80 flags and their 6502 namesakes do not always have the same meaning (Case in point: the 6502 carry flag assumes a logical

"NOT" BORROW meaning during subtract and compare operations, while the Z80 and most other common processors invert the carry flag internally during these operations so that it represents a "TRUE" BORROW; other flag differences, for the most part, are more subtle).

Finally, each individual processor is responsible for exiting with codes for the mnemonic, the addressing mode, and the number of bytes occupied by the instruction available for use by the "DISPLAY" routine (line 173) to index into yet another set of tables and retrieve the ASCII messages for mnemonic and addressing modes. Upon exiting the instruction processors, the value stored in "NEXT PROGRAM COUNTER" is updated, and control is passed back to the COMMAND section where the operating mode in effect determines whether to display the updated CPU contents, and whether or not to continue execution with the next instruction.

Several distinct operating modes are available, defined briefly as follows:

Single-Step Mode: One instruction is decoded and executed. The console then displays the address of the instruction just executed (Current Program Counter), the decoded instruction (Mnemonic and Operand), the state of all CPU registers, and the address of the next instruction to be executed (Next Program Counter). Control is then returned to the user.

Multi-Step Trace Mode: The user specifies up to 255 instructions to be executed with the results displayed as above following each step. This occurs at the rate of approximately two instructions per second. Control is returned to the user after the required number of steps have been executed and displayed.

Run-to-Breakpoint Mode: Instructions are executed without display up to and including the user-specified breakpoint. CPU registers are then displayed as above and control is returned to the user.

Free-Run Mode: Instructions are executed with no display and no breakpoints. The console keyboard, however, is monitored, and CPU contents are displayed as above and control is returned to the user at any point upon console input.

The user may examine and modify any or all CPU registers, including the program counter, at any time that he has control, without otherwise disturbing any operation in progress.

The 6502 "BRK" instruction (software interrupt) is handled through a vector location, just as in a normal system. Optionally, however, the BRK instruction may be used (and is initialized) to return control to the user in the same manner as the breakpoint mode described above. The BRK instruction and breakpoint mode operate independently, and both may be used in the same program. Additionally, up to five different "user" subroutines may be called for transfer of control to Z80 routines (such as console I/O). These are invoked by executing either a "JMP" or "JSR" instruction to addresses 0000, 0001, 0002, 0003, and 0004 (HEX). Control is passed via a jump table located within the intepreter, which the user may load with Z80 "JP" instructions to the desired subroutines. The subroutines must be terminated in the normal Z80 fashion by one of the "RET" instructions. Note that the first two entries in the table (JMP or JSR 0000, 0001 HEX) are initialized as console input and output via the 6502 accumulator. Three parameters may be passed to and from all user subroutines, as outlined in Table 2. Table 3 lists the locations of the BRK vector and the user subroutine jump table.

Although the interpreter is quite versatile and all 6502 instructions are handled properly, there are some limitations to its use. Perhaps the most important of these is that, even in the "free-run" mode, execution is quite slow compared to an actual 6502 running at normal clock speeds. No attempt has been made to correlate execution times with normal 6502 clock periods (this may, in fact, be impossible due to the way in which the system handles multiple addressing modes). Therefore, programs incorporating software timing loops will not execute properly. Also, since the pacage was intended for software development, hardware interrupts are not supported, and only a limited amount of hardware interfacing can be performed through the "user" subroutine calls.

Two program listings are included here. One, "ZXLD" is located at the end of this article. Its function is to boot the main program from the CP/M Transient Program Area to high memory. The other listing is the interpreter itself, assembled to reside from 6F00 to 7DFF (HEX) in a 32K system.

Note that this leaves 512 bytes, normally containing CBIOS, unaffected. The program is not relocatable, so that in order to run in a different memory configuration, it will be necessary to reassemble the source with a new value in the "SIZE EQU XX" statement. All other addresses are derived from this value. If the object code is to be manually loaded for the 32K system then "ZXLD" must be loaded first, starting at 0100 (HEX). Then start loading ZX65 at 0110 (HEX), continuing until the end. Finally, use the CP/M "SAVE" command (SAVE 15 ZX65.COM) to write the object code to disk. The package can now be invoked by answering "ZX65" to the CP/M prompt. If you are reassembling the package, then you should also reassemble the ZXLD routine (modified for your system) so that you have a .HEX file for each module. Then load the package using DDT as follows:

```
DDT ZXLD.HEX
IZX65.HEX
Rdisp
```

Where "disp" is the load offset value required to load the object code from the .HEX file at a different address than that specified. (DDT will calculate "disp" if you type "H1 10, ssss", where "ssss" is the actual starting address of your program. DDT will respond with the sum and difference of the two values, and the difference will be the required value "disp".) Finally, save the object code as directed above for manual loading.

To run the program, place the disk with ZX65.COM on drive A (this disk should also have a copy of CP/M so that the system can be restarted, if necessary, without changing disks). Place the ZX65 data disk on drive B. Start CP/M and invoke ZX65 by answering "ZX65" to the CP/M input prompt. The system will prompt you with a header and a ">" symbol. You are now in the command mode, and typing one of the command codes (see Table 4) will cause the appropriate action to occur. Note that if you are using a disk not previously used for ZX65 on the drive B, you will be unable to use the ZX65 DOS functions until you have initialized the disk. This is done simply by typing "I", at which point the system will ask you to verify this request with a "Y" or "N". The reason for this

(note this well) is that the "I" command clears and initializes the directory on disk B. This is the equivalent of the CP/M command "ERA *.*" and must be used with caution.

For those unwilling to type in a 4K Byte program by hand, I will make the package available on a standard 8 inch IBM format, single side, single density diskette (Sorry, I can't handle any other format!) at a nominal charge of $15 to cover time and material. Write to me and specify your system memory size. I will send a diskette containing both source and object files assembled to reside in high memory as described above, as well as a copy of ZXLD. A command summary, operating hints, and other general information will be included on the disk.

Finally, although this program has been rather exhaustively tested with a variety of 6502 software (both simple and sophisticated) I cannot guarantee it bug-free. I have included no copyright notices of any kind, and I hope users will feel free to copy, add, delete, and improve at will. (I would like to hear of any major modifications . . . I may want to include them myself!) As stated at the outset, the package was developed as a side benefit of another project, not as a money maker, and if anyone else finds it useful, so much the better.

Acknowledgments:

CP/M is a registered trademark of Digital Research, Pacific Grove, CA.

While not specifically mentioned in the text, I have relied heavily on the information contained in the book "6502 Assembly Language Programming" by Lance A. Leventhal (Osborne/McGraw-Hill Inc., 1979).

Function	Address
1. Test console input status	7A2EH
2. Get single character from console	7A2BH
3. Send single character to console	7A28H
4. Select disk drive (A or B)	7A34H
5. Home R/W head of selected drive	7A31H
6. Set disk transfer buffer address	7A3DH
7. Set sector number for disk access	7A37H
8. Set track number for disk access	7A3AH
9. Read one sector from selected drive	7A40H
10. Write one sector to selected drive	7A43H

Table 1: Functions which ZX65 requires of the host system, and their locations within the interpreter.

6502 Register	Passed as	Z80 Register
Accumulator		Accumulator
X Index		B Register
Y Index		C Register

Table 2: 6502/Z80 Parameter passing for user subroutines.

Description	Address of vector or jump instruction
BRK Vector	7A55H
User 0	7A46H
User 1	7A49H
User 2	7A4CH
User 3	7A4FH
User 4	7A52H

Table 3: Locations for 'BRK' vector and user jump table.

Command	Function
C	CPU register display and modify.
D	Directory display of ZX65 files on disk B. Format: FILNAM.TYP LOAD ADDRESS #RECORDS
E	Examine a block of memory. System will prompt for starting address and number of bytes.
G	Go to 6502 program per Current Program Counter and execute without display (free-run).
I	Initialize a fresh disk on drive B. (Must be formatted.) System will prompt for verification.
K	Kill a ZX65 file on drive B. Similar to CP/M ERA. System will prompt for file name.
L	Load a ZX65 file from disk B. System will prompt for file name.
M	Memory substitution. View sequential memory locations and update if desired. System will prompt for address. Mode will continue in effect until a '.' (period) is typed.
R	Load and run a ZX65 file from disk B, starting at the address specified in the directory. (Must be an executable 6502 program.) System will prompt for file name.
S	Save a ZX65 file on disk B. System will prompt for file name, starting address, and number of bytes.
T	Trace several 6502 instructions with display following each step. System will prompt for desired number of steps.
SPACE	Single-step command. System will execute one 6502 instruction, display the results, and stop.

Table 4: A command summary for ZX65 (in alphabetical order).

LISTING

```
0001 ; ZX65: SIMULATING A MICRO      1/80      R M KRUSE
0002 ; FOR Z80-BASED SYSTEMS USING THE CP/M DOS
0003 ;
0004 ; DESIGNED TO REPLACE ALL OF CP/M EXCEPT FOR USERS CBIOS
0005 ; THRU WHICH DISK AND I/O CALLS ARE MADE.
0006 ;
0007 ; THIS VERSION IS ASSEMBLED FOR A 32K SYSTEM. TO CHANGE
0008 ; TO ANOTHER SIZE, INSERT APPROPRIATE VALUE FOR 'SIZE'
0009 ; AND RE-ASSEMBLE. SOURCE CODE IS IN PROPER FORMAT FOR
0010 ; THE S. D. SALES Z80 ASSEMBLER V1.0.
0011 ;
0012 ; PROGRAM MAY BE LOADED BY THE FOLLOWING PROCEDURE:
0013 ;       DDT SIMLD.HEX
0014 ;       IZX65.HEX
0015 ;       R9210
0016 ;       G0
0017 ;       SAVE 15 ZX65.COM
0018 ;
0019 ; THE .RES. OPERATOR CAUSES THIS ASSEMBLER TO CLEAR ANY
0020 ; OVERFLOW IN THE OPERAND FIELD.
0021 ;
0022 SIZE    EQU     32              ; # K CP/M SYSTEM
0023 TOP     EQU     .RES.SIZE*1024  ; TOP OF RAM
0024 BASE    EQU     .RES.TOP-1100H  ; ACTUAL SYSTEM RUN ADDR
0025 HOST    EQU     .RES.TOP-200H   ; HOST SYSTEM CBIOS
0026 SYSTK   EQU     HOST            ; SYSTEM STACK
0027 CPWRM   EQU     .RES.HOST+3     ; CP/M WRM ST
0028 CONST   EQU     .RES.HOST+6     ; HOST SYSTEM INPUT TEST
0029 CONIN   EQU     .RES.HOST+9     ; HOST SYSTEM CONS IN
0030 CONOUT  EQU     .RES.HOST+0CH   ; HOST SYSTEM CONS OUT
0031 HOME    EQU     .RES.HOST+18H   ; DRIV HOME
0032 SETDRV  EQU     .RES.HOST+1BH   ; SELECT DRVE
0033 SETTRK  EQU     .RES.HOST+1EH   ; SELECT TRACK
0034 SETSEC  EQU     .RES.HOST+21H   ; SELECT SECTOR
0035 SETBUF  EQU     .RES.HOST+24H   ; SET XFER ADDRESS
0036 READ    EQU     .RES.HOST+27H   ; READ A SECTOR
0037 WRITE   EQU     .RES.HOST+2AH   ; WRITE A SECTOR
0038 ;
0039         ORG     BASE
0040         LD      SP,SYSTK
0041         LD      C,1
>0020
>8000
>6F00
>7E00
>7E00
>7E03
>7E06
>7E09
>7E0C
>7E18
>7E1B
>7E1E
>7E21
>7E24
>7E27
>7E2A
>6F00 31007E
6F03 0E01
```

```
6F05  CD347A      0042          CALL    DRVS            ; ACTIVATE DRV B
6F08  CD317A      0043          CALL    HOMB            ; AND INIT IT
6F0B  FD21667A    0044          LD      IY,INTRO
6F0F  CDD979      0045          CALL    PRTXT           ; PRINT HEADER
6F12  CD4F79      0046          CALL    CRLF
                  0047 ;
                  0048 ; COMMAND LEVEL...RETURN HERE FROM ANY POINT IN PGM
                  0049 ;
6F15  21156F      0050 CMND     LD      HL,CMND
6F18  E5          0051          PUSH    HL              ; RET ADDR ON STACK
6F19  CD4F79      0052          CALL    CRLF
6F1C  CD5979      0053          CALL    PROM
6F1F  CD167A      0054          CALL    GETCHR          ; GET COMMAND
6F22  FE45        0055          CP      'E'
6F24  CAE975      0056          JP      Z,EXAM          ; EXAM MEMORY
6F27  FE4D        0057          CP      'M'
6F29  CA4B76      0058          JP      Z,SUBM          ; SUB MEMORY
6F2C  DD215D7A    0059          LD      IX,VREG         ; FOR SIMX
6F30  FE42        0060          CP      'B'
6F32  2864        0061          JR      Z,BRKP-$        ; RUN TO BREAKPOINT
6F34  FE43        0062          CP      'C'
6F36  CACE6F      0063          JP      Z,RGSTR         ; DSPLY/MODIFY CPU REGS
6F39  FE47        0064          CP      'G'
6F3B  CAC16F      0065          JP      Z,RUN           ; RUN FROM NPC
6F3E  FE54        0066          CP      'T'
6F40  282D        0067          JR      Z,MULTX-$       ; TRACE/DISPLAY N STEPS
6F42  FE20        0068          CP      ' '
6F44  2836        0069          JR      Z,EXONE-$       ; EX ONE INSTR & STOP
6F46  DD216A7B    0070          LD      IX,BUFFR        ; FOR DOS
6F4A  FE44        0071          CP      'D'
6F4C  CAA876      0072          JP      Z,PDIR          ; DOS DIR
6F4F  FE49        0073          CP      'I'
6F51  CACB77      0074          JP      Z,INIT          ; DOS INITIALIZE
6F54  FE4B        0075          CP      'K'
6F56  CAB677      0076          JP      Z,KILL          ; DOS ERASE
6F59  FE4C        0077          CP      'L'
6F5B  CAF576      0078          JP      Z,LOAD          ; DOS LOADER
6F5E  FE52        0079          CP      'R'
6F60  CAEB76      0080          JP      Z,LDGO          ; LOAD AND RUN PGM
6F63  FE53        0081          CP      'S'
6F65  CA1377      0082          JP      Z,SAVE          ; DOS SAVER
6F68  0E3F        0083          LD      C,'?'
6F6A  CD287A      0084          CALL    PRNC            ; INVALID COMMAND
6F6D  18A6        0085          JR      CMND-$
                  0086 ;
                  0087 ; EXECUTE N STEPS
                  0088 ;
6F6F  FD211F7B    0089 MULTX    LD      IY,MLTMSG
6F73  CDD979      0090          CALL    PRTXT
6F76  CD6379      0091          CALL    GETWRD          ; GET N
6F79  7B          0092          LD      A,E
6F7A  2002        0093          JR      NZ,DOMULT-$
6F7C  3E01        0094 EXONE    LD      A,1             ; EXECUTE 1 STEP
                  0095 ;
                  0096 ; EXECUTE # STEPS IN A
                  0097 ;
6F7E  32577A      0098 DOMULT   LD      (NSTEPS),A      ; STORE N IN TEMP
6F81  CD9670      0099          CALL    STEP            ; EXEC & DSPLY INSTR
6F84  CD1170      0100          CALL    DSPLY
6F87  3A577A      0101          LD      A,(NSTEPS)
6F8A  3D          0102          DEC     A
6F8B  C8          0103          RET     Z
6F8C  F5          0104          PUSH    AF
6F8D  01597A      0105          LD      BC,DLYCON       ; DELAY FOR VIEWING
6F90  0B          0106 DOLP     DEC     BC
6F91  78          0107          LD      A,B
6F92  B1          0108          OR      C
6F93  20FB        0109          JR      NZ,DOLP-$
6F95  F1          0110          POP     AF
6F96  18E6        0111          JR      DOMULT-$
                  0112 ;
                  0113 ; RUN WITHOUT DISPLAY TO BREAKPOINT
                  0114 ;
6F98  FD21317B    0115 BRKP     LD      IY,BKPMSG
6F9C  CDD979      0116          CALL    PRTXT
6F9F  CD6379      0117          CALL    GETWRD          ; GET IT
6FA2  ED535B7A    0118          LD      (BREAK),DE      ; STORE IT IN TEMP
6FA6  CD9670      0119 BKLP     CALL    STEP            ; EXEC ONE INSTR
6FA9  ED5B5B7A    0120          LD      DE,(BREAK)
6FAD  7C          0121          LD      A,H             ; CHECK IF AT BKPT
6FAE  BA          0122          CP      D
6FAF  2009        0123          JR      NZ,INTST-$
6FB1  7D          0124          LD      A,L
6FB2  BB          0125          CP      E
6FB3  2005        0126          JR      NZ,INTST-$
6FB5  CD9670      0127          CALL    STEP            ; DO ONE MORE
6FB8  1857        0128          JR      DSPLY-$
6FBA  CD2E7A      0129 INTST    CALL    KBST            ; TEST FOR KBD INP
6FBD  28E7        0130          JR      Z,BKLP-$        ; CONTINUE IF NONE
6FBF  1808        0131          JR      QDSP-$
                  0132 ;
                  0133 ; RUN ONLY..NO DISPLAY OR BREAKPOINT
                  0134 ;
6FC1  CD9670      0135 RUN      CALL    STEP            ; DO A STEP
6FC4  CD2E7A      0136          CALL    KBST            ; CHECK FOR KB INPUT
6FC7  28F8        0137          JR      Z,RUN-$         ; IF NONE JUST REPEAT
                  0138 ;
                  0139 ; STOP, DISPLAY, & RET TO CMND
                  0140 ;
6FC9  CD2B7A      0141 QDSP     CALL    KBIN            ; CLEAR UART
6FCC  1843        0142          JR      DSPLY-$         ; DISP CPU & RET
                  0143 ;
                  0144 ; DISPLAY & UPDATE CPU REGISTERS
                  0145 ;
6FCE  DDE5        0146 RGSTR    PUSH    IX
6FD0  CD4F79      0147          CALL    CRLF
6FD3  FD21AA7A    0148          LD      IY,HEADR+42
6FD7  CD8C70      0149          CALL    PRN6            ; PRINT RG NAME
6FDA  ED5B647A    0150          LD      DE,(NPC)        ; NEXT PC TO DE
6FDE  CDB279      0151          CALL    PRTWRD          ; PRINT IT
6FE1  CD5979      0152          CALL    PROM            ; PROMPT FOR CHANGE
6FE4  CD6379      0153          CALL    GETWRD
6FE7  2804        0154          JR      Z,AREG-$        ; SPACE = NO CHANGE
6FE9  ED53647A    0155          LD      (NPC),DE        ; PUT IN NEW VALUE
6FED  FD218C7A    0156 AREG     LD      IY,HEADR+12
6FF1  0605        0157          LD      B,5
6FF3  CD4F79      0158 NREG     CALL    CRLF            ; DO REST OF REGS
6FF6  CD8C70      0159          CALL    PRN6            ; PRINT NAME
6FF9  DD5E00      0160          LD      E,(IX+0)
6FFC  CDB679      0161          CALL    PRTBYT          ; PRINT CONTENTS
6FFF  CD5979      0162          CALL    PROM            ; PROMPT FOR CHANGE
7002  CD6379      0163          CALL    GETWRD
7005  2803        0164          JR      Z,NXTREG-$      ; SPACE = NO CHANGE
7007  DD7300      0165          LD      (IX+0),E
700A  DD23        0166 NXTREG   INC     IX              ; POINT TO NEXT REG
700C  10E5        0167          DJNZ    NREG-$          ; DO AGAIN IF MORE
```

```
700E DDE1       0168          POP     IX
7010 C9         0169          RET
                0170 ;
                0171 ; DISPLAY PC, INSTRUCTION, AND ALL REGISTERS
                0172 ;
7011 DDE5       0173 DSPLY    PUSH    IX              ; SAVE POINTER
7013 C5         0174          PUSH    BC              ; SAVE MNEMONIC CODE
7014 CD4F79     0175          CALL    CRLF
7017 FD21807A   0176          LD      IY,HEADR
701B CD8C70     0177          CALL    PRN6            ; 'NPC'
701E ED5B627A   0178          LD      DE,(CPC)
7022 CDB279     0179          CALL    PRTWRD          ; VALUE
7025 CD8C70     0180          CALL    PRN6            ; 'INS'
7028 C1         0181          POP     BC
7029 FDE5       0182          PUSH    IY
702B C5         0183          PUSH    BC
702C FD21A87C   0184          LD      IY,MNMTBL       ; DET MNEM FROM BC
7030 3EFC       0185          LD      A,0FCH
7032 A1         0186          AND     C
7033 CB3F       0187          SRL     A
7035 5F         0188          LD      E,A
7036 1600       0189          LD      D,0
7038 FD19       0190          ADD     IY,DE
703A FD5E00     0191          LD      E,(IY+0)        ; GET PACKED MNEM
703D FD5601     0192          LD      D,(IY+1)
7040 CB23       0193          SLA     E               ; UNPACK IT
7042 CB12       0194          RL      D
7044 0E03       0195          LD      C,3             ; & PRINT THREE CHRS
7046 AF         0196 MNM2     XOR     A
7047 0605       0197          LD      B,5
7049 CB23       0198 MNM1     SLA     E
704B CB12       0199          RL      D
704D 17         0200          RLA
704E 10F9       0201          DJNZ    MNM1-$
7050 CBF7       0202          SET     6,A
7052 C5         0203          PUSH    BC
7053 CD277A     0204          CALL    PRNA
7056 C1         0205          POP     BC
7057 0D         0206          DEC     C
7058 20EC       0207          JR      NZ,MNM2-$
705A C1         0208          POP     BC              ; DECODE ADDR MODE
705B 78         0209          LD      A,B
705C CB27       0210          SLA     A
705E CB27       0211          SLA     A
7060 CB27       0212          SLA     A
7062 5F         0213          LD      E,A
7063 1600       0214          LD      D,0
7065 FD21187D   0215          LD      IY,AMDTBL
7069 FD19       0216          ADD     IY,DE
706B CD9170     0217          CALL    PRN8            ; PRINT ADDR MODE
706E FDE1       0218          POP     IY
7070 0605       0219          LD      B,5
7072 CD8C70     0220 DSP1     CALL    PRN6            ; PRINT ALL REGS
7075 DD5E00     0221          LD      E,(IX+0)
7078 CDB679     0222          CALL    PRTBYT
707B DD23       0223          INC     IX
707D 10F3       0224          DJNZ    DSP1-$
707F CD8C70     0225          CALL    PRN6            ; PRINT NEXT PC
7082 ED5B647A   0226          LD      DE,(NPC)
7086 CDB279     0227          CALL    PRTWRD
7089 DDE1       0228          POP     IX
708B C9         0229          RET
                0230 ;
708C 1E06       0231 PRN6     LD      E,6             ; PRINT 6 CHARS PER IY
708E C3DB79     0232          JP      PRNLP
7091 1E08       0233 PRN8     LD      E,8             ; PRINT 8 CHRS PER IY
7093 C3DB79     0234          JP      PRNLP
                0235 ;
                0236 ; CODE FOR DECODING AND EXECUTING ONE INSTRUCTION
                0237 ; ENTER WITH HL POINTING TO INSTR TO BE DECODED.  EXITS
                0238 ; WITH HL POINTING TO NEXT INSTR, B HOLDING ADDRESSING
                0239 ; MODE CODE, C HOLDING MNEMONIC CODE
                0240 ;
7096 2A647A     0241 STEP     LD      HL,(NPC)        ; NEXT PC TO HL
7099 22627A     0242          LD      (CPC),HL        ; BECOMES CURRENT PC
709C 7E         0243          LD      A,(HL)          ; OP CODE
709D B7         0244          OR      A               ; ZERO?
709E CAF670     0245          JP      Z,PBRK          ; IF SO, 'BRK'
70A1 CB47       0246          BIT     0,A             ; TEST LSD FOR ODD/EVEN
70A3 C21873     0247          JP      NZ,MLTMOD       ; ODD INSTRS MULTI MODE
70A6 E60F       0248          AND     0FH
70A8 2018       0249          JR      NZ,FLGTST-$
70AA 7E         0250          LD      A,(HL)          ; RESTORE OP CODE
70AB CB67       0251          BIT     4,A             ; MSD ODD/EVEN
70AD C22A71     0252          JP      NZ,PBRNCH       ; IF ODD, BRANCH
70B0 FE20       0253          CP      20H
70B2 CA6271     0254          JP      Z,PJSR          ; PROCESS JSR
70B5 FE40       0255          CP      40H
70B7 CAD971     0256          JP      Z,PRTI          ; PROCESS RTI
70BA FE60       0257          CP      60H
70BC CAED71     0258          JP      Z,PRTS          ; PROCESS RTS
70BF C31873     0259          JP      MLTMOD          ; ELSE MULTI MODE
70C2 FE08       0260 FLGTST   CP      8
70C4 200E       0261          JR      NZ,IMP2-$
70C6 7E         0262          LD      A,(HL)          ; RESTORE
70C7 CB67       0263          BIT     4,A             ; MSD ODD/EVEN
70C9 CA2172     0264          JP      Z,PIMPL         ; IF EVEN, MST BE IMPLD
70CC FE98       0265          CP      98H             ; MSD = 9?
70CE CADB72     0266          JP      Z,PTYA          ; YES, PROC TYA SPECIAL
70D1 C3E872     0267          JP      PFLGS           ; ELSE PROCESS FLAG OPS
70D4 FE0A       0268 IMP2     CP      0AH
70D6 2010       0269          JR      NZ,JMPTST-$     ; MUST BE JUMP
70D8 7E         0270          LD      A,(HL)          ; RESTORE
70D9 E6F0       0271          AND     0F0H            ; TESTS ON MSD
70DB CB67       0272          BIT     4,A             ; ODD/EVEN
70DD C20672     0273          JP      NZ,PTSS         ; PROSESS SP OPS SPECIAL
70E0 FE70       0274          CP      70H
70E2 DA1873     0275          JP      C,MLTMOD        ; PROCESS ACCUM MODES
70E5 C32172     0276          JP      PIMPL           ; ELSE MORE IMPLIEDS
70E8 7E         0277 JMPTST   LD      A,(HL)          ; RESTORE OP CODE
70E9 FE4C       0278          CP      4CH
70EB CA8471     0279          JP      Z,PJABS         ; PROCESS ABS JMP
70EE FE6C       0280          CP      6CH
70F0 CA8A71     0281          JP      Z,PJIND         ; PROCESS INDIRECT JMP
70F3 C31873     0282          JP      MLTMOD
                0283 ;
                0284 ; INSTURCTION PROCESSORS BEGIN HERE.  ENTER WITH
                0285 ; HL = PC, IX POINTING TO SIM CPU
                0286 ;
                0287 ; IF BRK VECTOR (IPOINT) = 0, RETURN TO CMND
                0288 ;
70F6 DDCB01E6   0289 PBRK     SET     4,(IX+1)        ; SET BRK STATUS BIT
70FA 2A557A     0290          LD      HL,(IPOINT)     ; SEE IF RET TO CMND
70FD 7C         0291          LD      A,H
```

```
70FE  B5        0292          OR      L
70FF  2007      0293          JR      NZ,BRK1-$
7101  C1        0294          POP     BC              ; ADJUST SP
7102  012904    0295          LD      BC,429H         ; RET TO CMND
7105  C31170    0296          JP      DSPLY
7108  23        0297 BRK1     INC     HL              ; NORMAL BREAK
7109  CD0475    0298          CALL    TSPY            ; SP TO IY
710C  FD7400    0299          LD      (IY+0),H        ; 'PUSH' PC
710F  FD2B      0300          DEC     IY
7111  FD7500    0301          LD      (IY+0),L
7114  FD2B      0302          DEC     IY
7116  DD7E01    0303          LD      A,(IX+1)        ; 'PUSH' P
7119  FD7700    0304          LD      (IY+0),A
711C  FD2B      0305          DEC     IY
711E  CD0D75    0306          CALL    TYSP            ; IY TO SP
7121  2A557A    0307          LD      HL,(IPOINT)     ; GET INTERRUPT VECTOR
7124  012904    0308          LD      BC,429H         ; CODE FOR # BYTES & MNEM
7127  C3D774    0309          JP      END1            ; AND FINISH
                0310 ;
                0311 ; PROCESS BRANCHES
                0312 ;
712A  0600      0313 PBRNCH   LD      B,0             ; FORM INDEX INTO MSKTBL
712C  4E        0314          LD      C,(HL)          ; INSTR TO C
712D  FD213C7C  0315          LD      IY,MSKTBL
7131  CB39      0316          SRL     C               ; GET MSD
7133  CB39      0317          SRL     C
7135  CB39      0318          SRL     C
7137  CB39      0319          SRL     C
7139  0D        0320          DEC     C
713A  FD09      0321          ADD     IY,BC
713C  23        0322          INC     HL              ; POINT TO OFFSET
713D  CB49      0323          BIT     1,C             ; MSD ODD/EVEN
713F  DD7E01    0324          LD      A,(IX+1)        ; P REG TO ACC
7142  FD4E00    0325          LD      C,(IY+0)        ; # CODE
7145  FD4601    0326          LD      B,(IY+1)        ; FLAG MASK
7148  2005      0327          JR      NZ,BRSET-$      ; ODD/EVEN DET SET/CLR
714A  A0        0328          AND     B               ; ISOLATE BIT FOR TEST
714B  2807      0329          JR      Z,TRUE-$
714D  180E      0330          JR      FALSE-$
714F  A0        0331 BRSET    AND     B
7150  2002      0332          JR      NZ,TRUE-$
7152  1809      0333          JR      FALSE-$
7154  1600      0334 TRUE     LD      D,0             ; CONDITION MET; BRANCH
7156  5E        0335          LD      E,(HL)          ; OFFSET TO E
7157  CB7B      0336          BIT     7,E             ; CHECK SIGN OF OFFSET
7159  2801      0337          JR      Z,TRU1-$
715B  15        0338          DEC     D               ; IF B7 IS SET, DO NEG
715C  19        0339 TRU1     ADD      HL,DE          ; ADD OFFSET FOR NEW PC
715D  0600      0340 FALSE    LD      B,0             ; COND NOT MET; GO ON
715F  C3D674    0341          JP      END
                0342 ;
                0343 ; PROCESS JUMPS AND JSR...JMP 0-4 AND JSR 0-4 GO INTO Z80
                0344 ; ROUTINES PER JUMP TABLE AT USR0. CONTROL THEN RETURNS
                0345 ; TO SIMULATOR UPON Z80 'RET' (C9).. ALL OTHERS PROCESSED
                0346 ; NORMALLY.
                0347 ;
7162  23        0348 PJSR     INC     HL              ; PROCESS JSR
7163  CDA471    0349          CALL    SPCTST          ; TEST FOR SPECIALS
7166  2005      0350          JR      NZ,PJS2-$
7168  CDB271    0351          CALL    INUSR           ; GO TO USER ROUTINES
716B  1811      0352          JR      PJS3-$
716D  CD0475    0353 PJS2     CALL    TSPY            ; NORMAL JSR
7170  FD7400    0354          LD      (IY+0),H        ; PUSH CURRENT PC
7173  FD2B      0355          DEC     IY
7175  FD7500    0356          LD      (IY+0),L
7178  FD2B      0357          DEC     IY
717A  CD0D75    0358          CALL    TYSP
717D  EB        0359          EX      DE,HL           ; NEW PC
717E  017302    0360 PJS3     LD      BC,273H
7181  C3D774    0361          JP      END1
7184  23        0362 PJABS    INC     HL              ; ABS JUMP
7185  016F02    0363          LD      BC,26FH
7188  1808      0364          JR      DOJP-$
718A  23        0365 PJIND    INC     HL              ; INDIRECT JUMP
718B  5E        0366          LD      E,(HL)          ; JUMP POINTER
718C  23        0367          INC     HL
718D  56        0368          LD      D,(HL)
718E  EB        0369          EX      DE,HL           ; ADDR OF POINTER TO HL
718F  016F0C    0370          LD      BC,0C6FH
7192  CDA471    0371 DOJP     CALL    SPCTST          ; TEST FOR SPECIALS
7195  2009      0372          JR      NZ,DOJ1-$
7197  C5        0373          PUSH    BC
7198  CDB271    0374          CALL    INUSR           ; FOR SPECIALS
719B  CDF671    0375          CALL    PRET            ; DO SR RETURN
719E  C1        0376          POP     BC
719F  EB        0377          EX      DE,HL
71A0  EB        0378 DOJ1     EX      DE,HL           ; NEW PC
71A1  C3D774    0379          JP      END1
                0380 ;
                0381 ; TEST FOR USER SUBROUTINE CALLS/JUMPS.  IF TARGET
                0382 ; ADDRESS < 0005 THEN RET WITH Z FLG SET, ELSE CLR
                0383 ;
71A4  5E        0384 SPCTST   LD      E,(HL)          ; TEST FOR USER SPECIALS
71A5  23        0385          INC     HL              ; TARGET TO DE
71A6  56        0386          LD      D,(HL)          ; HI BYTE
71A7  23        0387          INC     HL              ; NXT INSTR
71A8  3E00      0388          LD      A,0             ; TARGET < 0005?
71AA  B2        0389          OR      D
71AB  C0        0390          RET     NZ
71AC  3E04      0391          LD      A,4
71AE  BB        0392          CP      E
71AF  D8        0393          RET     C
71B0  AF        0394          XOR     A               ; SET Z FLAG
71B1  C9        0395          RET
                0396 ;
                0397 ; INDEX INTO USER SUBROUTINE TABLE
                0398 ; PASS A, X, & Y TO/FROM USER ROUTINE AS A, B, & C.
                0399 ;
71B2  C5        0400 INUSR    PUSH    BC              ; SAVE CODES
71B3  1600      0401          LD      D,0
71B5  7B        0402          LD      A,E
71B6  83        0403          ADD     A,E
71B7  83        0404          ADD     A,E
71B8  5F        0405          LD      E,A
71B9  FD21467A  0406          LD      IY,USR0
71BD  01CE71    0407          LD      BC,USRET        ; RETURN ON STACK
71C0  C5        0408          PUSH    BC
71C1  FD19      0409          ADD     IY,DE
71C3  DD7E00    0410          LD      A,(IX+0)        ; PASS A,X, & Y
71C6  DD4602    0411          LD      B,(IX+2)
71C9  DD4E03    0412          LD      C,(IX+3)
71CC  FDE9      0413          JP      (IY)            ; AND GO
71CE  DD7700    0414 USRET    LD      (IX+0),A        ; RETURN HERE
71D1  DD7002    0415          LD      (IX+2),B        ; RETURN PARAMETERS
71D4  DD7103    0416          LD      (IX+3),C
71D7  C1        0417          POP     BC
```

```
71D8  C9        0418          RET
                0419 ;
                0420 ; PROCESS SR AND INTERRUPT RETURNS
                0421 ;
71D9  CD0475    0422 PRTI     CALL    TSPY            ; RTI
71DC  FD23      0423          INC     IY
71DE  FD7E00    0424          LD      A,(IY+0)        ; RESTORE FLAGS
71E1  DD7701    0425          LD      (IX+1),A
71E4  CDF971    0426          CALL    PRET1           ; RESTORE PC
71E7  01A504    0427          LD      BC,4A5H
71EA  C3D774    0428          JP      END1
71ED  CDF671    0429 PRTS     CALL    PRET            ; RTS...RESTORE PC
71F0  01A904    0430          LD      BC,4A9H
71F3  C3D774    0431          JP      END1
71F6  CD0475    0432 PRET     CALL    TSPY            ; SP TO IY
71F9  FD23      0433 PRET1    INC     IY              ; POINT TO LAST ENTRY
71FB  FD6E00    0434          LD      L,(IY+0)        ; RESTORE PCL
71FE  FD23      0435          INC     IY
7200  FD6600    0436          LD      H,(IY+0)        ; RESTORE PCH
7203  C30D75    0437          JP      TYSP            ; RESTORE NEW SP
                0438 ;
                0439 ; PROCESS SP EXCHANGES
                0440 ;
7206  FE90      0441 PTSS     CP      90H             ; WHICH ONE IS IT?
7208  200C      0442          JR      NZ,PTSX-$
720A  DD7E02    0443 PTXS     LD      A,(IX+2)        ; MOVE DATA FROM X...
720D  DD7704    0444          LD      (IX+4),A        ; TO SP
7210  01D904    0445          LD      BC,4D9H
7213  C3D674    0446          JP      END
7216  DD7E04    0447 PTSX     LD      A,(IX+4)        ; MOVE SP
7219  DD7702    0448          LD      (IX+2),A        ; TO X REG
721C  0ED1      0449          LD      C,0D1H
721E  C3E372    0450          JP      COMPL           ; AND MODIFY FLAGS
                0451 ;
                0452 ; PROCESS ALL REMAINING IMPLIED INSTRUCTIONS
                0453 ;
7221  4F        0454 PIMPL    LD      C,A             ; OP CODE
7222  CB39      0455          SRL     C               ; GET MSD
7224  CB39      0456          SRL     C
7226  CB39      0457          SRL     C
7228  CB39      0458          SRL     C
722A  E60F      0459          AND     0FH             ; CHECK LSD
722C  FE08      0460          CP      8
722E  FD215C7C  0461          LD      IY,IMPTBL       ; INDEX INTO TABLE
7232  CADA75    0462          JP      Z,JPIB
7235  FD21647C  0463          LD      IY,IMPTBL+8
7239  C3DA75    0464          JP      JPIB
                0465 ;
                0466 ; IMPLIED INSTR PROCESSORS
                0467 ;
723C  DD3402    0468 PINX     INC     (IX+2)          ; INX
723F  0E65      0469          LD      C,65H
7241  1813      0470          JR      IDEFLG-$
7243  DD3403    0471 PINY     INC     (IX+3)          ; INY
7246  0E69      0472          LD      C,69H
7248  180C      0473          JR      IDEFLG-$
724A  DD3502    0474 PDEX     DEC     (IX+2)          ; DEX
724D  0E55      0475          LD      C,55H
724F  1805      0476          JR      IDEFLG-$
7251  DD3503    0477 PDEY     DEC     (IX+3)          ; DEY
7254  0E59      0478          LD      C,59H
7256  DDCB018E  0479 IDEFLG   RES     1,(IX+1)        ; SET/RES Z FLG
725A  2004      0480          JR      NZ,SINTST-$     ; PER Z80 FLAGS
725C  DDCB01CE  0481          SET     1,(IX+1)
7260  DDCB01BE  0482 SINTST   RES     7,(IX+1)        ; ALSO SIGN FLG
7264  F21373    0483          JP      P,ENDIT
7267  DDCB01FE  0484          SET     7,(IX+1)
726B  C31373    0485          JP      ENDIT
726E  018504    0486 PNOP     LD      BC,485H         ; NOP
7271  C3D674    0487          JP      END
7274  CD0475    0488 PPHA     CALL    TSPY            ; PHA
7277  DD7E00    0489          LD      A,(IX+0)        ; ACCUM
727A  0E8D      0490          LD      C,8DH
727C  1808      0491          JR      DOPUSH-$        ; PUSH IT
727E  CD0475    0492 PPHP     CALL    TSPY            ; PHP
7281  DD7E01    0493          LD      A,(IX+1)        ; P REG
7284  0E91      0494          LD      C,91H
7286  C5        0495 DOPUSH   PUSH    BC
7287  FD7700    0496          LD      (IY+0),A        ; PLACE ON STACK
728A  FD2B      0497          DEC     IY              ; BUMP SP DOWN
728C  CD0D75    0498          CALL    TYSP            ; & REPLACE IT
728F  C1        0499          POP     BC
7290  C31373    0500          JP      ENDIT
7293  CDB472    0501 PPLA     CALL    DOPULL          ; PLA
7296  DD7700    0502          LD      (IX+0),A        ; DATA TO ACC
7299  CDED74    0503          CALL    SINS1           ; SET/RES S,Z FLGS
729C  CD0D75    0504          CALL    TYSP            ; RESTORE SP
729F  019504    0505          LD      BC,495H
72A2  C3D674    0506          JP      END
72A5  CDB472    0507 PPLP     CALL    DOPULL          ; PLP
72A8  DD7701    0508          LD      (IX+1),A        ; DATA TO P REG
72AB  CD0D75    0509          CALL    TYSP            ; RESTORE SP
72AE  019904    0510          LD      BC,499H
72B1  C3D674    0511          JP      END
72B4  CD0475    0512 DOPULL   CALL    TSPY            ; GET SP
72B7  FD23      0513          INC     IY              ; BUMP SP
72B9  FD7E00    0514          LD      A,(IY+0)        ; GET DATA
72BC  C9        0515          RET
                0516 ;
                0517 ; PROCESS ACC/INDEX TRANSFERS
                0518 ;
72BD  DD7E00    0519 PTAX     LD      A,(IX+0)        ; TAX
72C0  DD7702    0520          LD      (IX+2),A
72C3  0EC9      0521          LD      C,0C9H
72C5  181C      0522          JR      COMPL-$
72C7  DD7E00    0523 PTAY     LD      A,(IX+0)        ; TAY
72CA  DD7703    0524          LD      (IX+3),A
72CD  0ECD      0525          LD      C,0CDH
72CF  1812      0526          JR      COMPL-$
72D1  DD7E02    0527 PTXA     LD      A,(IX+2)        ; TXA
72D4  DD7700    0528          LD      (IX+0),A
72D7  0ED5      0529          LD      C,0D5H
72D9  1808      0530          JR      COMPL-$
72DB  DD7E03    0531 PTYA     LD      A,(IX+3)        ; TYA
72DE  DD7700    0532          LD      (IX+0),A
72E1  0EDD      0533          LD      C,0DDH
72E3  CDED74    0534 COMPL    CALL    SINS1           ; SET/RES S,Z FLGS
72E6  182B      0535          JR      ENDIT-$
                0536 ;
                0537 ; PROCESS FLAG SET/RES INSTRUCTIONS
                0538 ;
72E8  FD214C7C  0539 PFLGS    LD      IY,FLGTBL       ; TABLE START
72EC  0600      0540          LD      B,0
72EE  FEB8      0541          CP      0B8H            ; CLV SPECIAL
72F0  2002      0542          JR      NZ,PFL1-$
```

(Continued on pg. 30)

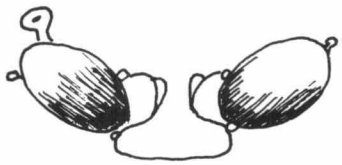
H.A.R.L.I.E. a "no programming" computer

BY J. L. LAWRENCE AND JAY SPRENKLE
Computer Science Dept. Software Lab
University of Missouri – Rolla
Rolla, Missouri 65401

It was with much interest and skepticism that the authors read the article "Build a Self-Learning No-programming Computer with your Microprocessor" by Klaus Holtz in *DDJ* #33. The article describes a computer which requires no programming, is educated almost like a human child, and is relatively inexpensive. Applications suggested for this new computer include language translation, a typewriter which refuses to type incorrectly spelled words, unbreakable secret codes, and others which are currently difficult to implement. All of this is claimed to be achievable with no programming!

The article's author describes briefly the theory behind the proposed computer as "Knowledge Processing and Infinite Dimensional Networks." It would be nice to have a reference for such an amazing theory, if a reference does exist. Rather than repeat the theory here, we merely pause to mention it. As Holtz points out, understanding of the theory is not necessary to implement his ideas.

Figure 1.



```
10 REM ****************************
20 REM *     H A R L I E          *
30 REM *                          *
40 REM ****************************
80 DIM I$(72),M1(2),M9(2),M5(2000,1),T$(72),T(72),W(72)
90 DIM A5(100,1)
100 N=2000:M%(N,0)=-9999:M5(N,1)=-9999
110 M=100:A5(M,0)=-9999:A5(M,1)=-9999
120 ! CHR$(26)
130 ! TAB(23),"----------------------"
140 ! TAB(23),"! H. A. R. L. I. E. !"
150 ! TAB(23),"----------------------"
160 !:! TAB(19),"I AM A SELF-LEARNING COMPUTER":!:!
170 !:!:I$="":INPUT "> ",I$
180 IF I$="*STOP" THEN STOP
181 IF I$="*MEM" THEN 245
182 IF I$="*CLEAR" THEN 246
190 I9=LEN(I$):IF I9=0 THEN 170
200 IF I$(I9,I9)<>" " THEN 230:REM GET RID OF TRAILING BLANKS
210 I$=I$(1,I9-1)
220 GOTO 180
230 GOSUB 250
240 GOTO 170
245 GOSUB 1310:GOTO 170
246 GOSUB 1500:GOTO 170
250 REM SENTENCE ENCODING ROUTINE
260 IF I$(1,1)="*" THEN S=1 ELSE S=0
270 IF S=1 THEN I$=I$(2,LEN(I$))
280 L9=LEN(I$):IF I$(L9,L9)="?" THEN Q=1 ELSE Q=0:REM WAS IT QUESTION
290 IF Q=1 THEN I$=I$(1,L9-1)
300 IF I$(1,2)="A*" THEN Q=2:REM IS IT AN ANSWER
305 IF Q=2 THEN S=1:REM ALWAYS STORE ANSWERS
310 IF Q<>2 THEN 330
320 I$=I$(3,LEN(I$))
330 L9=LEN(I$):J9=1:W1=-1:F2=0
340 T$="":FOR I=J9 TO L9
350 IF I$(I,I)=" " THEN 390
360 T$=T$+I$(I,I)
370 NEXT I
380 GOTO 440
390 J9=I+1
400 GOSUB 1040:REM CONVERT T$ TO T
410 P9=-1:GOSUB 890:REM SEE IF I KNOW IT ALREADY
420 W1=W1+1:W(W1)=A:IF F>0 THEN F2=F
430 GOTO 340
440 P9=-1:GOSUB 1040:GOSUB 890
450 W1=W1+1:W(W1)=A:IF F>0 THEN F2=F
460 FOR I=0 TO W1:T(I)=W(I):NEXT I
470 T(W1+1)=32
480 P9=-2:GOSUB 890:REM ADD SENTENCE
490 IF F>0 THEN F2=F
500 IF Q=1 AND F<2 THEN A9=A:REM ADDRESS OF LAST QUESTION
510 IF Q=0 THEN 610:REM NOT A QUESTION OR ANSWER
520 IF F2=0 OR F2=1 THEN 560
530 IF F2=2 THEN I$="**** I DON'T KNOW ****"
540 IF F2=3 THEN I$="**** MORE STORAGE REQUESTED ****"
550 GOTO 650
560 GOSUB 700:REM GET ANSWER
570 IF F=0 THEN 590:REM I KNOW THE ANSWER
580 GOTO 610
590 GOSUB 1190
600 GOTO 650
610 IF F=0 THEN I$="**** I KNEW THAT ****"
620 IF F=1 THEN I$="**** I WILL REMEMBER THAT ****"
630 F2=F
640 GOTO 530
650 ! I$
690 RETURN
700 REM ASSOCIATIVE NETWORK ROUTINE
710 F=0
720 IF Q=1 THEN 750:REM ANSWER NOT SET
730 M1(1)=A9:M1(2)=A:REM PUT IN NEW ANSWER TO QUESTION
740 GOTO 760
750 M1(1)=A:REM PUT IN QUESTION
760 A=0
770 G=A5(A,0):P=A5(A,1):M9(1)=G:M9(2)=P:REM FETCH MEMORY WORD
780 IF G=0 THEN 830:REM ALL ZERO
790 IF G=M1(1) THEN 860:REM QUESTION=POINTER
800 A=A+1
810 IF A5(A,0)<>-9999 THEN 770
820 F=3:GOTO 880:REM MORE MEMORY
830 IF Q=2 THEN 850:REM ANSWER KEY SET
840 F=2:GOTO 880:REM I DON'T KNOW
850 A5(A,0)=M1(1):A5(A,1)=M1(2):F=1:GOTO 880:REM STORE MATRIX REGISTER
860 IF Q=2 THEN 850:REM ANSWER KEY SET
870 A=M9(2)
880 RETURN
890 REM SERIAL NETWORK -- FIGURE 2
900 A=0:M1(2)=P9:P5=-1:F=0
```

```
91Ø P5=P5+1:M1(1)=T(P5):REM GET NEXT INPUT CHAR
92Ø G=M5(A,Ø):P=M5(A,1):M9(1)=G:M9(2)=P:REM FETCH MEMORY WORD
93Ø IF G=Ø THEN 98Ø:REM WORD IS ALL ZERO
94Ø IF G=M1(1) AND P=M1(2) THEN 1Ø1Ø:REM WORD=MATRIX REGISTER
95Ø A=A+1:REM INCREMENT ADDRESS
96Ø IF M5(A,Ø)<>-9999 THEN 92Ø:REM END OF STORAGE?
97Ø F=3:GOTO 1Ø3Ø
98Ø IF S=1 THEN 1ØØØ:REM STORAGE ENABLED
99Ø F=2:GOTO 1Ø3Ø
1ØØØ F=1:M5(A,Ø)=M1(1):M5(A,1)=M1(2):REM STORE MATRIX REG IN MEMORY
1Ø1Ø M1(2)=A:REM LOAD ADDRESS INTO POINTER
1Ø2Ø IF M1(1)<>32 THEN 91Ø:REM GATE=END CODE
1Ø3Ø RETURN
1Ø4Ø REM CONVERT T$ TO T
1Ø5Ø L=LEN(T$)
1Ø6Ø FOR I=1 TO L:T(I-1)=ASC(T$(I,I)):NEXT I
1Ø7Ø T(L)=32
1Ø8Ø RETURN
1Ø9Ø REM RETRIEVAL ROUTINE -- FIGURE 3
11ØØ P5=-1:T$=""
111Ø G=M5(A,Ø):P=M5(A,1):M9(1)=G:M9(2)=P:REM FETCH MEMORY WORD
112Ø P5=P5+1:T(P5)=G
113Ø IF P=P9 THEN 115Ø:REM POINTER = Ø(1)
114Ø A=P:GOTO 111Ø
115Ø FOR I=P5 TO Ø STEP -1
116Ø T$=T$+CHR$(T(I))
117Ø NEXT I
118Ø RETURN
119Ø REM RETRIEVE A SENTENCE INTO I$
12ØØ W1=-1:I$="":P9=-1
121Ø K9=M5(A,1):REM SKIP END OF SENTENCE MARKERS
122Ø W1=W1+1:T(W1)=M5(K9,Ø)
123Ø IF M5(K9,1)=-2 THEN 125Ø
124Ø K9=M5(K9,1):GOTO 122Ø
125Ø FOR I=W1 TO Ø STEP -1:W(W1-I)=T(I):NEXT I
126Ø FOR J=Ø TO W1
127Ø A=W(J):GOSUB 1Ø9Ø:REM RETRIEVE WORD
128Ø I$=I$+T$
129Ø NEXT J
13ØØ RETURN
131Ø REM PRINT OUT MASS MEMORY
132Ø P5=Ø
133Ø ! P5,TAB(1Ø),M5(P5,Ø),TAB(2Ø),M5(P5,1),TAB(3Ø),
134Ø IF M5(P5,Ø)>31 AND M5(P5,Ø)<128 THEN ! CHR$(M5(P5,Ø)) ELSE !
135Ø P5=P5+1
136Ø IF M5(P5,Ø)>=1 THEN 133Ø
137Ø P5=Ø:! "ASSOCIATIVE MEMORY"
138Ø ! P5,TAB(1Ø),A5(P5,Ø),TAB(2Ø),A5(P5,1),TAB(3Ø)
139Ø P5=P5+1
14ØØ IF A5(P5,Ø)>=1 THEN 138Ø ELSE RETURN
15ØØ M5(Ø,Ø)=Ø:M5(Ø,1)=Ø:A5(Ø,Ø)=Ø:A5(Ø,1)=Ø
151Ø M5(1,Ø)=Ø:M5(1,1)=Ø:A5(1,Ø)=Ø:A5(1,1)=Ø
152Ø RETURN
```

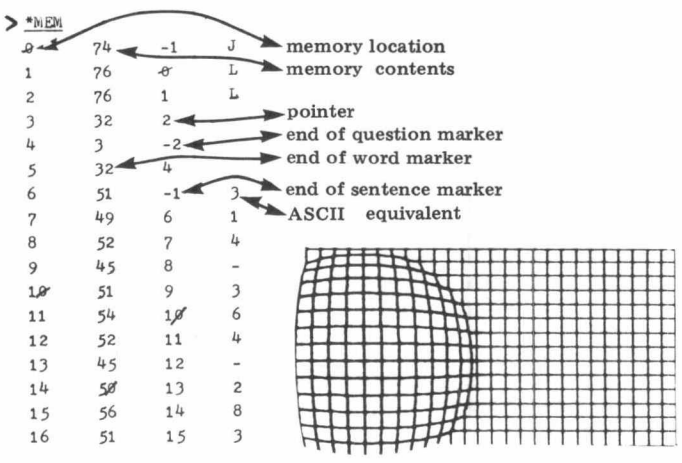
Being fully aware of the philosophical problems and implications of computer "learning" (and wishing to protect our future as programmers!), we decided to implement the proposed machine. Instead of using hardware to actually build the machine, we simulated the machine in software. Figure 1 is a listing of the program in Northstar BASIC which we ran on an Imsai 8080. Furthermore, we have run the same program on an Apple II and an 8080 with CP/M and Microsoft BASIC. The code can be vastly improved upon, but the implementation was intended to reproduce the hardware description as faithfully as possible. In the terminology of the original article, lines 890-1080 represent the Serial Network Routine, lines 1090-1180 represent the Retrieval Routine, and lines 700-880 represent the Associative Network Routine. The rest of the code is fairly obvious and is used to enhance the I/O. See figure 6 of the original article for further clarification. As for main variables, array A5 represents the associative network memory, array M5 represents the serial network memory, M1 is the matrix register, and M9 is the memory word.

Figure 2.

```
 ------------------
 ! H. A. R. L. I. E. !
 ------------------
I AM A SELF-LEARNING COMPUTER

>*JLL?
**** I DON'T KNOW ****

>A*314-364-2831
**** I WILL REMEMBER THAT ****

>*UMR?
**** I DON'T KNOW ****

>A*UNIV. OF MO. ROLLA
**** I WILL REMEMBER THAT ****

>*JCL?
**** I DON'T KNOW ****

>A*JOB CONTROL LANGUAGE
**** I WILL REMEMBER THAT ****

>UMR?
UNIV. OF MO.-ROLLA

>JLL?
314-364-2831

>*STOP
STOP IN LINE 181
READY
```

Operation of H.A.R.L.I.E. is straightforward. Figure 2 illustrates a sample run. Referring to figure 2, ">" is the prompt character indicating that H.A.R.L.I.E. is expecting a response from you. All user responses are underlined for clarity. To use H.A.R.L.I.E., the user merely types in the request/response at the keyboard. Any response preceded by "*" turns on the storage flag and causes whatever was typed to be placed in the serial network memory (array M5) if it is not already there. Any response which ends with "?" is a question which H.A.R.L.I.E. will answer if he can. If H.A.R.L.I.E. doesn't know the answer to the question, he will respond with "**** I don't know ****", as figure 2 shows. After an unsuccessful query, H.A.R.L.I.E. expects you to supply the correct answer although it is not necessary. Preceding the response with "A*" causes H.A.R.L.I.E. to remember the correct response the next time the appropriate question is asked. This is the way in which H.A.R.L.I.E. "learns."

Figure 3.



Ø	74	-1	J
1	76	Ø	L
2	76	1	L
3	32	2	
4	3	-2	
5	32	4	
6	51	-1	3
7	49	6	1
8	52	7	4
9	45	8	-
1Ø	51	9	3
11	54	1Ø	6
12	52	11	4
13	45	12	-
14	5Ø	13	2
15	56	14	8
16	51	15	3

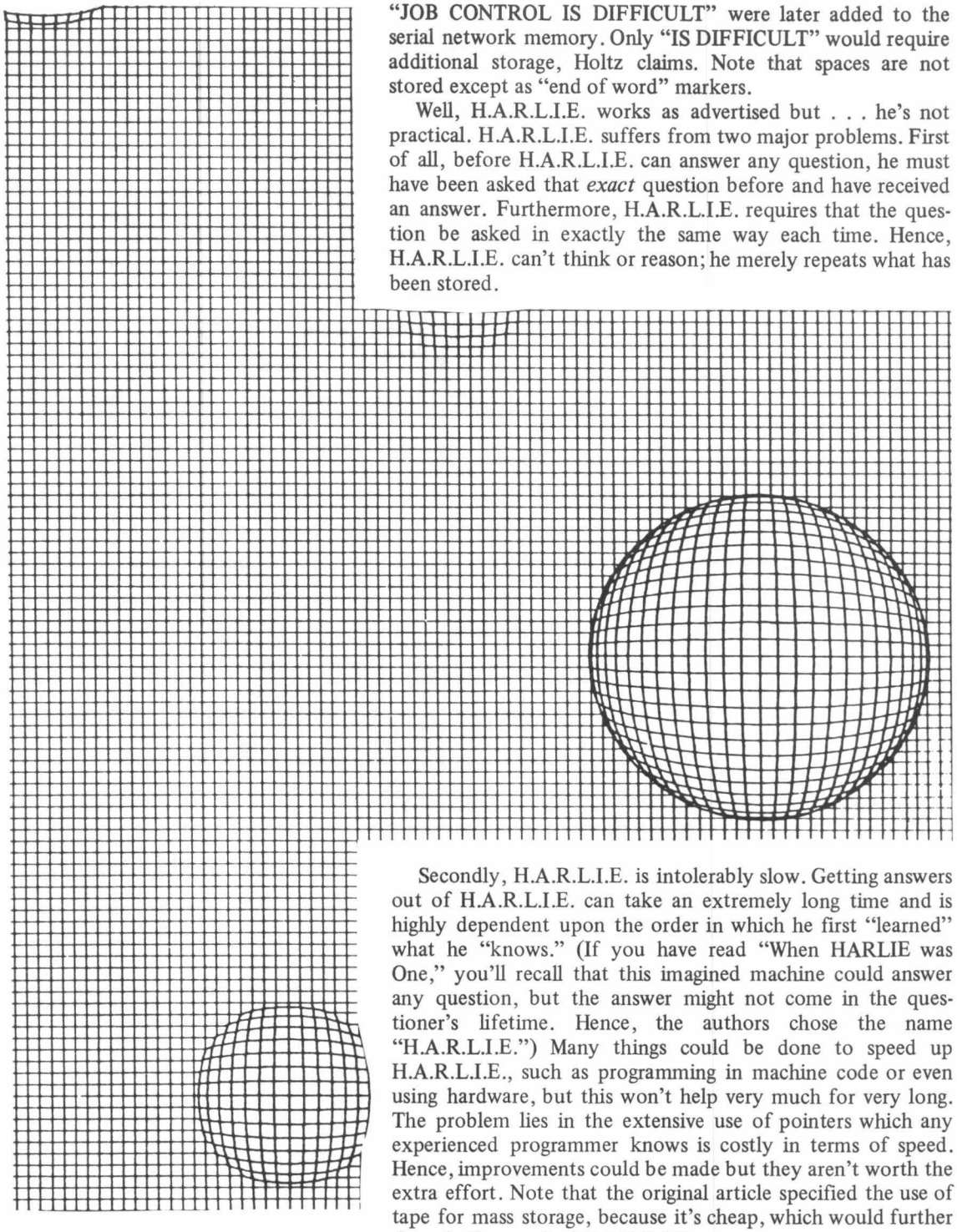
17	49	16	1
18	32	17	
19	18	-2	
20	32	19	
21	85	-1	U
22	77	21	M
23	82	22	R
24	32	23	
25	24	-2	
26	32	25	
27	78	21	N
28	73	27	I
29	86	28	V
30	46	29	.
31	32	30	
32	79	-1	O
33	70	32	F
34	32	33	
35	77	-1	M
36	79	35	O
37	46	36	.
38	45	37	-
39	82	38	R
40	79	39	O
41	76	40	L
42	76	41	L
43	65	42	A
44	32	43	
45	31	-2	
46	34	45	
47	44	46	,
48	32	47	
49	67	0	C
50	76	49	L
51	32	50	
52	51	-2	3
53	32	52	
54	79	0	O
55	66	54	B
56	32	55	
57	67	-1	C
58	79	57	O
59	78	58	N
60	84	59	T
61	82	60	R
62	79	61	O
63	76	62	L
64	32	63	
65	76	-1	L
66	65	65	A
67	78	66	N
68	71	67	G
69	85	68	U
70	65	69	A
71	71	70	G
72	69	71	E
73	32	72	
74	56	-2	8
75	64	74	@
76	73	75	I
77	32	76	

ASSOCIATIVE MEMORY

0	5	20	memory location
1	26	48	pointer to answer
2	53	77	pointer to questions

Figure 3 shows the contents of memory after the dialogue of figure 2 has occurred; refer to the original article for a description of how to follow the pointers. The point to be made here is to illustrate what Holtz means when he claims that the more information stored, the less storage required to store additional information. From figure 2, note that "JLL" and a phone number was entered. Later, "JCL" and "JOB CONTROL LANGUAGE" were also entered. Looking at figure 3, the "J" is stored only once. Suppose the sentence "JOB CONTROL IS DIFFICULT" were later added to the serial network memory. Only "IS DIFFICULT" would require additional storage, Holtz claims. Note that spaces are not stored except as "end of word" markers.

Well, H.A.R.L.I.E. works as advertised but . . . he's not practical. H.A.R.L.I.E. suffers from two major problems. First of all, before H.A.R.L.I.E. can answer any question, he must have been asked that *exact* question before and have received an answer. Furthermore, H.A.R.L.I.E. requires that the question be asked in exactly the same way each time. Hence, H.A.R.L.I.E. can't think or reason; he merely repeats what has been stored.



Secondly, H.A.R.L.I.E. is intolerably slow. Getting answers out of H.A.R.L.I.E. can take an extremely long time and is highly dependent upon the order in which he first "learned" what he "knows." (If you have read "When HARLIE was One," you'll recall that this imagined machine could answer any question, but the answer might not come in the questioner's lifetime. Hence, the authors chose the name "H.A.R.L.I.E.") Many things could be done to speed up H.A.R.L.I.E., such as programming in machine code or even using hardware, but this won't help very much for very long. The problem lies in the extensive use of pointers which any experienced programmer knows is costly in terms of speed. Hence, improvements could be made but they aren't worth the extra effort. Note that the original article specified the use of tape for mass storage, because it's cheap, which would further degrade the speed of H.A.R.L.I.E.

In conclusion, H.A.R.L.I.E. does work but he is highly impractical. Users won't be dumping accumulated "knowledge" from one tape to another as Holtz envisions, nor will knowledge be sorted in all-knowing Super Computers with this particular design, which accomplishes the task of fact retrieval in a very awkward and inefficient manner. Much better techniques exist that are currently operative in many database systems. After reading Holtz's article and this one, the reader will realize that both are meant to be taken with a healthy sense of humor and a large block of salt!

(Author's Note: H.A.R.L.I.E. is taken from the science fiction book "When HARLIE was One" by David Gerrold.)

CLEAN STARTS FOR COSMAC 1861 VIDEO OUTPUT

BY GARY H. PRICE
733 Blue Sage Drive
Sunnyvale, CA 94086

The 1861 low-resolution video chip is a versatile adjunct to the 1802 Cosmac microprocessor, but the interrupt software required for frame formatting typically slows program execution by about a factor of two. This overhead can sometimes be alleviated by enabling and disabling the 1861 under software control, as is done, for example, by Netronics ELF II TINY BASIC when using the 1861 for character output. A clean start is not assured, however, on the first frame following an 1861 re-enable unless special precautions are taken. The 1861 DISPLAY STATUS line provides a means, as will be described, by which a re-enable can be synchronized with the interrupt and DMA requests to the 1802 through which the video frame is produced.

The 1861 DISPLAY STATUS line is set low twice per frame, once during the last four horizontal scans preceding the display window and again during the last four horizontal scans of the display. The interrupt request by which the display-refresh routine is entered is set during the last half of the first of these two STATUS-low periods. Sync signals are output by the 1861 even while disabled in order to maintain frame synchronization. Thus, if the 1861 is re-enabled within the display window (as can occur with ELF II TINY BASIC), the initial series, or partial series, of DMA requests following the enable will be received during execution of code outside the display-refresh routine. Such an occurrence is accompanied by a flash of "garbage" on the display screen.

The code given in Listings 1 and 2 avoids such bad frames by assuring that the 1861 is enabled outside the display window. DISPLAY STATUS is assumed to be connected to 1802 flag line No. 1 (active low), and DISPLAY ON and DISPLAY OFF are assumed to be connected to the decoded N lines set by the 1802 I/O instructions IN,1 (69) and OUT,1 (61), respectively, as in the ELF II. Register 2 is assumed to be pointing to an available memory location for the I/O byte transfers associated with the IN and OUT instructions (e.g., the next location on the top of the stack). Two versions of the routine are given. That in Listing 1 pertains to the case in which DISPLAY STATUS is directly connected to the 1802 flag line. The version given in Listing 2 is for the ELF II with I/O-monitor board installed. In this system, the 1861 DISPLAY STATUS and SYNC signals are ORed to generate the flag input to the 1802.

Listing 1. Basic 1861 reactivation routine.

```
Label   Instruction

WAIT1   B1
        WAIT1
WAIT2   BN1
        WAIT2
        SEX,2
        SEX,2
LOOP    NOP
        NOP
        DEC,2
        IN,1
        OUT,1
        B1
        LOOP
        IN,1
```

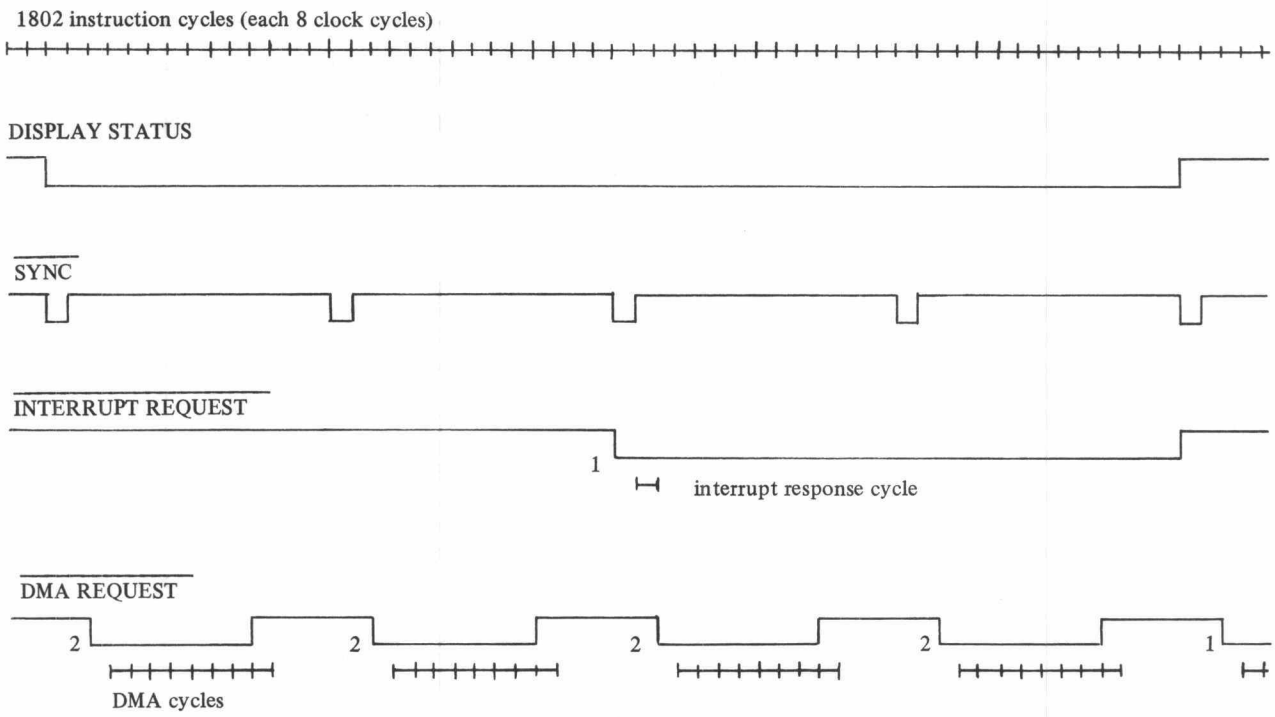
Listing 2. 1861 reactivation routine for ELF II with I/O-monitor board installed.

```
Label   Instruction

WAIT1   B1
        WAIT1
        B1
        WAIT1
WAIT2   BN1
        WAIT2
        SEX,2
LOOP    NOP
        NOP
        DEC,2
        IN,1
        OUT,1
        B1
        LOOP
        IN,1
```

The initial tests of DISPLAY STATUS preceding LOOP in the listings allow entry to the loop, the timing of which is critical, only on *initiation* of the STATUS signal. With only the first B1 in the WAIT1 loop in Listing 2, about 99 percent of the starts will be clean, but two tests in sequence are necessary in this case to prevent dropping through the hold on the SYNC pulse. In both the Listing 1 and Listing 2 versions of the routine, synchronization of the 1861 timing with the 1802 instruction fetch/execute cycle will be reestablished during the holds if it has been disrupted (e.g., by any 3-cycle instructions closely preceding entry into the routine). Once the final loop has been entered, the 1861 is enabled only during a series of brief windows between possible DMA-request bursts. Should the STATUS signal be that preceding the display window, the onset of the interrupt request is caught by one of these enable windows, and an ordinary display frame is generated by the interrupt-service routine. Upon exit from the interrupt-service routine, STATUS is high, and the 1861 is permanently (until the next disable) enabled as the loop finally is exited. Should the STATUS signal be that preceding the end of the display window, however, no interrupt is encountered during its duration, and the loop keeps the 1861 disabled during the periods that it otherwise would be initiating DMA action. Once the inhibited frame has passed, as indicated by STATUS going high, the loop is exited, and the 1861 finally is enabled permanently. The timing of the signal lines involved is shown in Figure 1.

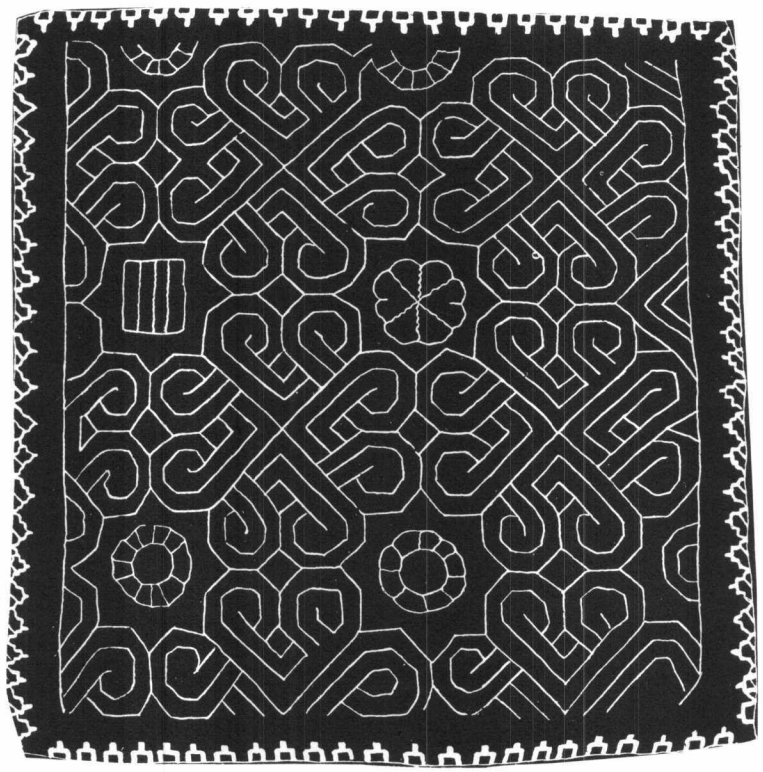
Gary Price is a physicist at SRI who is learning about microprocessors from his ELF II. He has done a hand-compilation for the 1802 of Dick Duda's Othello program from its original BASIC, and he would be happy to share this program on a Netronics-format tape. —*SR*

Figure 1. Timing of some 1861 signals and 1802 responses.



Notes:

1. Start of display window only
2. End of display window only



PCNET NOTEBOOK

AN OCCASIONAL MEMORANDUM

from dave caulkins

The PCNET Project has operated a bulletin board system* in the San Francisco Bay Area for about two years. The system we're running now is the Apple Bulletin Board System from Peripherals Unlimited. In late May/early June of '80 our ABBS experienced a (for me) novel problem; a person or persons would call the system, give a CB-like 'handle' instead of a name during logon, and then leave messages making free use of all the well known anglo-saxon expletives.

Other bulletin board systems in the area experienced similar behavior by someone using the same handle. Systems operators discussed the matter, and we all agreed that the commercial time-sharing service procedure of requiring login via password would solve the problem, but would sharply restrict the preset free access policy of BBS systems. We rejected this approach as putting too much of a damper on easy BBS access. Meanwhile back at the system the bad guys had grown more disagreeable, threatening to crash the system as well as leaving offensive messages. I was intrigued by the problem of devissing BBS security programs which would detect and discourage abusive behavior, but which would not be too much of a burden for (or better still not even be visible to) ordinary users. I coded up some security extensions to the ABBS which seem to have been effective against the bad guys; the number of offensive messages has dropped to near zero and there have been no system crashes in spite of several threats to that effect.

Two security measures seem to be particularly useful. Both are straightforward technically, but one involves some interesting psychology.

THE DIRTY WORD FILTER

This is a string parsing subroutine that detects any use of offensive words or phrases. The Dirty Word Filter is used on names submitted by users during the logon procedure as well as on almost all other strings sent by the user while he is on the system. If a dirty word or phrase is detected, the phone connection is immediately broken.

THE UNDESIRABLES LIST

The kind of person who gets his kicks from this sort of thing seems to have an ego involvement with the 'handle' he uses. He becomes the focus of attention with the added advantage of anonymity; "the mysterious DIRTY HARRY strikes again!". The undesirables list contains the names used on all objectionable messages. Whenever such a name is used on entry, the system breaks the phone connection. If new aliases are used, they are easily added to the list. It's hard to build up a reputation as a mysterious evil doer if you have to keep changing your name.

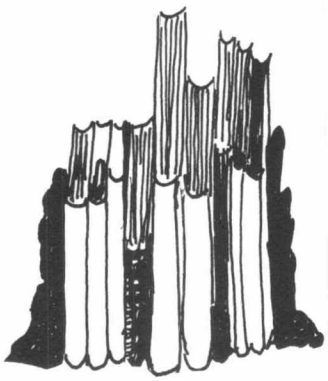
Some systems (e.g., ABBS) have the unfortunate property that if crashed the remote user is given full control of the system with the ability to alter the system program and do other anti-social things. Proper attention to detail in writing the system software can make a provoked crash very unlikely; there is a hardware fix that can make it impossible. It's called a watchdog timer. The concept is fairly simple; a hardware monostable multivibrator is built that can be reset by an output signal from the system computer. A circuit of this kind is easily built using a 555 timer and a few other parts; the whole thing should cost less than $5. The timer is set to time out in 1 to 5 seconds. A subsroutine is added to the BBS system that restarts the timer each time it is entered; this subroutine is called from enough places in the BBS so that the timer is always reset before it times out. If the system crashes for any reason the subroutine doesn't get called and the timer times out and hangs up the phone.

If all else fails your friendly local telephone company is prepared to assist in preventing your computer from being embarrassed by bad language. They have equipment which, with modest cooperation on the part of the BBS operator, can trace the number from which the dirty data was sent. Use of obscene words over the telephone turns out to be illegal, even if the languge is ASCII. So all you obscene data callers out there had best be careful. Your next visitor may be a cop with a bunch of printouts showing just what was sent from your phone and when.

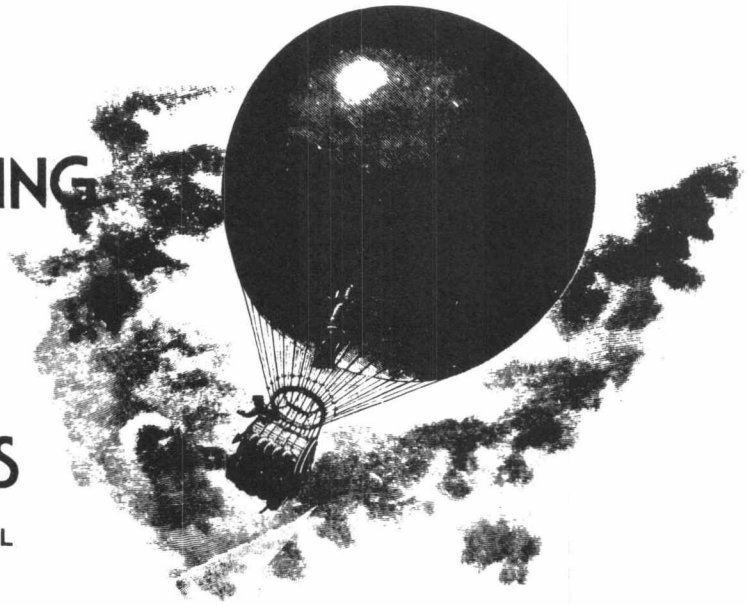
I'll be happy to share the full details of all my system security measures with other BBS operators. I'm not going to put them in this article; one doesn't provide detailed plans of the bank vault in an article on how to foil bank robbers. Write me care of DDJ and I'll arrange to supply you with the details.

You can write to Dave Caulkins at PCNET, P.O. Box E, Menlo Park, CA 94025.

** (See The Electric Phone Book on p.21 of the June/July* DDJ *for a list of them.)*



PROGRAMMING PASTIMES AND PLEASURES



©1980, BY CHARLES WETHERELL

Dataflow Computers

There is a tremendous range in speed between the fastest and the slowest computer in common use today. I don't know what the slowest commercial computer is (perhaps we should have a write-in contest?), but an 8-bit addition in 5 microseconds seems reasonable. This rate is still about the same as the addition speed of the IBM 1401 that I learned to program on back in 1964, so really this may not be so slow.

But it is not nearly fast enough for scientists who need answers to big problems. They often use a Cray-1, perhaps the fastest computer available on the market right now. A Cray-1 can do a 64-bit addition in about 15 nanoseconds. Ignoring questions about carries, the Cray-1 also does eight times as much work. But this is not the whole story. The Cray-1 sometimes can do five operations at once. So the additions could possibly go on 13,335 times as fast on a Cray-1 as on the microprocessor chip you bought at your local computer store.

How much of a difference is a factor of 13,335? Assume that some program takes exactly one minute on the Cray-1. The microprocessor would take almost 1-1/3 weeks! And if the Cray-1 program took one hour, the micro would take slightly more than 1½ years! Since Cray-1's are commonly used for time-dependent problems like weather prediction, a delay of a year and a half in turn-around could make a forecast of rain for tomorrow fairly useless.

Money and Technology

Speed is not the whole story, of course. Our hypothetical micro might sell for a couple of dollars and I could likely package it with memory and power for well under $100 in a useful configuration. The Cray-1 has a price tag of $5 megabucks. (Admittedly, the Cray-1 does have much more memory, but memory is relatively cheap.) The price ratio is 50,000 to 1, about three times my optimistic estimate of the speed ratio and perhaps ten times a more conservative speed ratio. Scientists pay heavily for the speed they need. And now many scientists are calling for gigaflop computers capable of one billion floating point operations a second. Imagine what these giants will cost.

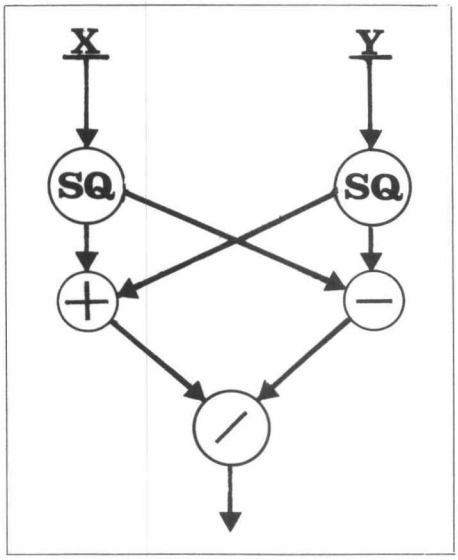
There is hope, though. The cover article of *Scientific American* for May, 1980, describes how superconducting computers built from Josephson junctions might be built. A prediction of about one nanosecond per instruction is also made, exactly what is needed for a gigaflop execution rate. But the article demonstrates, albeit indirectly, why a Josephson computer (and the Cray-1) will cost so much. To build the superfast giant, whole new areas of technology and engineering have to be developed, improved, or refined. All the learning and research cost money. In high technology manufacturing, serial number 1 of a new product is always expensive (remember the first transistor radios and hand calculators). After a hundred, a thousand, or a million items have been sold, the research cost can be spread out and drop to dollars or pennies per item.

The villain here is the necessity to use components employing the most advanced and expensive technology. What if the $2 microcomputers could somehow be connected together to solve the big problems using only reasonably standard technology, particularly using no superfast components? Then the supercomputer might not require an appropriation from Congress to buy.

An Army of Ants

How can a $2 micro beat a Cray-1? By enlisting a lot of friends. Assume that I can buy a reasonably cheap microprocessor that will do a 64-bit add in one microsecond. If 1,000 of these micros all run continuously on the same problem, they will average an operation per nanosecond. More realistically, if 5,000 micros each operate only 20% of the time because of scheduling and communication delays, the whole group will still average one nanosecond per operation. Metaphorically, an army of ants might be able to crash through the jungle as fast as an elephant *if* the ants can get organized.

Dataflow graphs are a way to organize the ants. Consider a typical calculation like $(x^2 + y^2)/(x^2 - y^2)$. In a normal computer, this would take at least five steps. But look at the dataflow graph:



Now it is clear that two computers could cooperate and do the calculation in three steps by sharing the work. If you remember that scientific calculations are much more elaborate than our simple example and that thousands of similar elements have parallel computations done on them, you can see that there is plenty of opportunity for parallel operation. Conventional computers hide this possible speed-

up because instructions must be executed sequentially even though the dataflow graph shows there is no dependence between two instructions. In fact, many programs could have entire loops or subprograms executed in parallel if their host computers could do the job.

A New Architecture

A *dataflow computer* has five main components: instruction memory, data memory, arbitration network, distribution network, and a battery of processors. In a conventional computer, instructions sit passively, waiting for the CPU to call them for execution. On a dataflow machine, each instruction takes an active part in the computation. An instruction waits for all of its operands to be ready; when they are, the instruction signals the arbitration network that it is ready for execution. The arbitration network packs the data and the instruction together and ships the package to a free processor. When the processor is through executing the instruction, the results are shipped back through the distribution network to any instructions waiting. In the graph above, the output from the left-hand square instruction would be passed to both the add and the subtract.

We can notice a number of facts about a dataflow machine. First, many instructions may be clamoring for execution at the same time; it is the responsibility of the arbitration network to sort out the chaos. There is no requirement, by the way, that the instructions be processed in order of readiness. Second, processors do not have to be synchronized at all nor need they all have the same capabilities. Data gets back to waiting instructions whenever the processor gets finished. The arbitration network must make sure instructions go only to processors that can execute them. Third, this scheme will work only if on average there are several thousand instructions ready to execute at one time and if bottlenecks, when only one or two instructions are ready, occur only rarely. Dataflow relies more than most architectures on good statistical behavior in programs, a property likely in scientific computations. Finally, the data store will be used to hold large data aggregates like records and arrays. This part of the system will require considerable engineering because it will have to be smarter than the average memory.

Dataflow is the idea of Jack Dennis at MIT and right now researchers around the world are working on ways to build dataflow machines. The promise of growing a fast, large computer out of many slow, cheap components intrigues scientists, computer engineers, and administrators who pay for computers. Here's hoping for success.

Z-80 Circuit for the H-8

BY TED T. STOWE
704 Harbor Road
Bricktown, NJ 08723

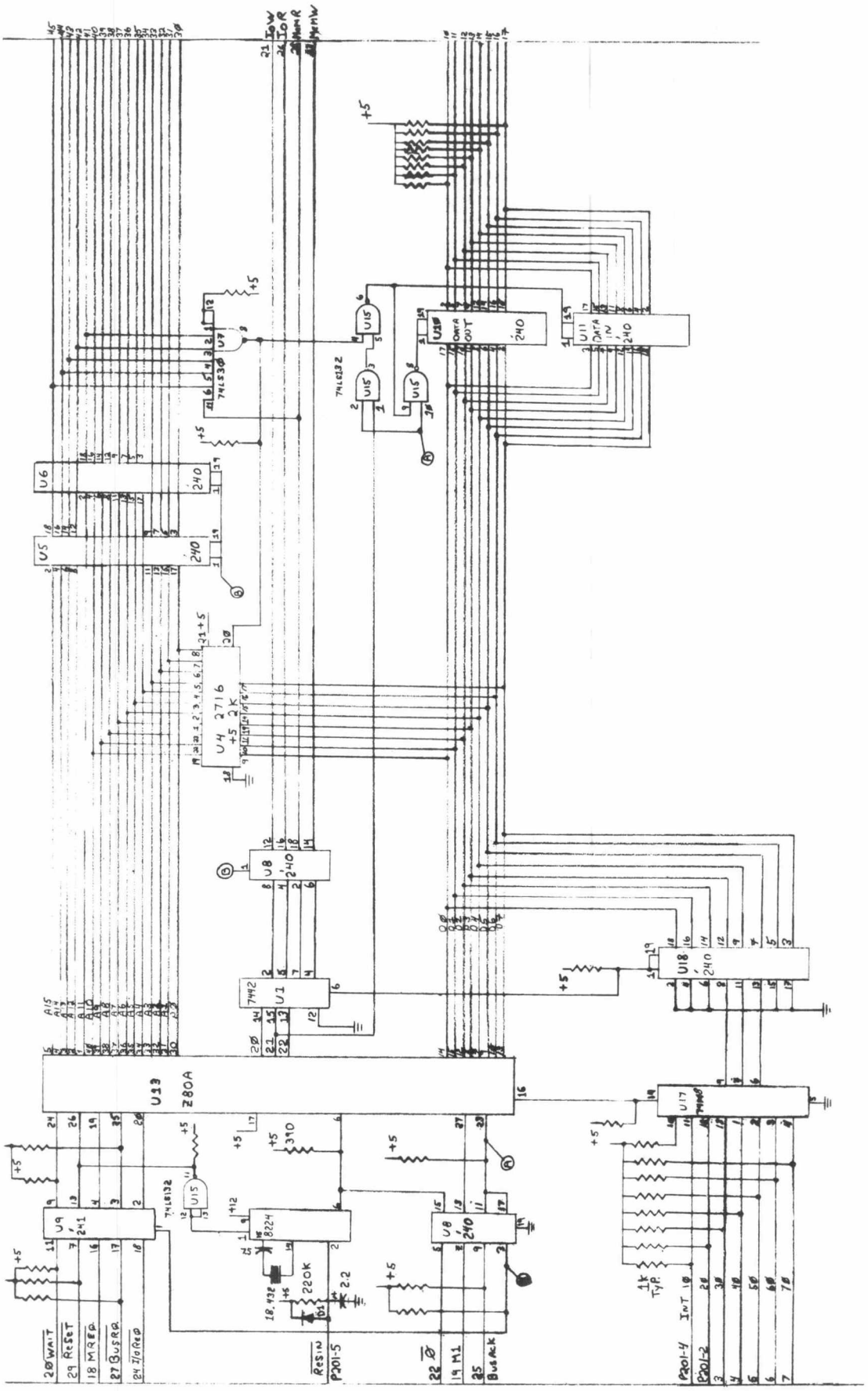
Thought you might be interested in a Z-80 circuit for the H-8 computer. I have had this circuit running for several months and have had only one minor problem, and that is that I had to go to CP/M for a Z-80 assembler as the Heath disk operating system is strictly 8080. I find in retrospect that my only gain with HDOS is faster instruction execution time, which created a problem with system utilities which sampled the real-time clock. However, the latest release of HDOS (ver 1.5) which is intended for the H-89 (Z-80) and H-8 (8080), has been updated in that area (both use the same clock speed and essentially the same monitor).

To enable the front panel single-step to function, it is necessary to remove IC108 and tie pin 8 to pin 11. This circuit has been operating with all of the Heath H-8 boards and the only problem to date that has popped up is with the latest version of the Heath 16K board, part number 85-2197-1. A minor conflict with the bank-enable exists that I haven't, as yet, had the time/inclination to resolve. If any problems/questions arise I will answer any correspondence.

(continued on next page)

IC List for Z-80 Board

DESIGNATION	REAL WORLD	HEATH PART NO.
U1	7442	443-756
U4	2716 +5	NOT AS YET
U5	74LS240	443-754
U6	74LS240	443-754
U7	74LS30	443-3
U8	74LS240	443-754
U9	74LS241	443-824
U10	74LS240	443-754
U11	74LS240	443-754
U12	8224	443-758
U13	Z-80	443-881
U15	74LS132	443-792
U17	74148	443-756
U18	74LS240	443-754


U13
Z80A
U4 2716
+5 2K
U1
7442
U8
240
U9
241
U5
240
U6
240
U7
74LS30
U15
74LS132
U10
DATA OUT
240
U11
DATA IN
240
U17
U18
240
8224
220K
390
2.2
1K TYP.
20 WAIT
29 RESET
18 MREQ
27 BUSRQ
24 I/OREQ
RESIN
P201-5
22 Ø
19 M1
25 BUSACK
P201-4
P201-2
INT 10

INDIRECT ADDRESSING FOR THE 8080 AND Z80

BY ANTHONY SKJELLUM
1695 Shenandoah Road
San Marino, CA 91108

Abstract

The 8080 and Z80 microprocessors do not provide indirect addressing modes for all instructions. This article, a response to a previous one dealing with indirect CALLs for the 6502, deals with the implementation of indirect addressing in software. Three techniques are explored: the "mini-program," as discussed in the 6502 article, the self-modifying subroutine and the self-modifying macro instruction. These techniques are applied to various instructions which lack the indirect mode. As well, the use of the one byte RST calls is suggested as a means of accessing the indirection routines in a fast manner. Finally, a specific example dealing with the Z80 indexed addressing mode is provided. The many listings are provided to act as a tutorial and an impetus to the reader to use the methods suggested in future software development.

After reading Kenneth Skier's article "Indirect Addressing for the 6502," in the January issue of *Byte*, I compiled a set of routines to handle indirect addressing for some of the cases where it is not available for the 8080 and Z80 microprocessors. While the particular case of indirect subroutine CALLs handled by Mr. Skier is easier to implemenet on 8080-type processors, several other operations are not available with the indirect addressing mode. The particular aspects dealt with here are (1) the implementation of the indirect CALLs as mentioned above, (2) the lack of indirect mode for memory loads and stores of 16 bit registers where the direct mode is provided for, (3) indirect addressing for input-output (I/O) instructions on the 8080 specifically, and (4) indirect mode for indexed addressing on the Z80. Besides the several general examples, one specific programing example has also been included to give the reader an idea of how the techniques discussed here might actually be applied. As well, the use of RST CALLs is included to suggest a fast access technique for some of the self-modifying routines included below.

Indirect Calls

Creating indirect mode for subroutine CALLs was one of the main points discussed in the previously mentioned article dealing with the 6502. This procedure is very easy to accomplish on the 8080 and Z80 microcomputers without the use of dedicated Random Access Memory locations as suggested by Mr. Skier for the 6502. Listing Ia provides a trivial routine to allow this procedure. The routine ICALL is CALLed by the main program with the HL register pair set up with the address of the target subroutine, i.e., the subroutine to be executed indirectly. The main program's CALL places the correct return address of the hardware stack where it remains until the target subroutine performs some sort of return instruction (assuming that the stack is not tampered with). The PCHL (Program counter := HL) instruction directs the program flow to the target subroutine by executing an indirect jump to the address contained in the HL register pair. This effects an indirect CALLing system.

However, suppose the subroutine requires data of some sort to be sent in the HL register pair (as is ofter the case). In such a situation this technique will not suffice. A modified version of Listing Ia (in Listing Ib) uses a dedicated word of memory to store the value of HL to be passed to the target routine. ICALL2 takes HL and places it on the stack above the main program's return address. It then loads HL with the contents of the location MEMW, which contains the correct data, and executes a RET instruction. The RET fetches the top address off the stack which is the target subroutine's address. Execution continues at the target routine as before and the additional stack usage will not affect the main program's return address, also on the stack.

Indirect Register Addressing

In the 8080-type microprocessors, the accumulator is provided with indirect addressing over the BC, DE, and HL register pairs. In fact, the eight bit general purpose registers B, C, D, E, H, and L may even participate in indirect addressing referred to the HL register pair (i.e., MOV B,M does B:=(HL)). However, no such mode is provided for the sixteen bit register pairs such as HL which IS provided with direct addressing. We will now consider several aspects of creating indirect addressing for the 8080 and Z80. Listing IIa illustrates the routine LHLI which performs an indirect load of HL from the address pointed to by HL initially. Note that this code is not self-modifying but that it uses a 4 byte buffer called SCRATCH which must reside in Random Access Memory. The function of this routine may not be openly obvious. What's being done is that the routine effectively creates a "mini-program" in the SCRATCH memory which consists of an LHLD opcode (load HL direct) a two byte operand field for the LHLD instruction and a RET instruction to give control back to the main program. (To get a visual idea of the "mini-program," see Figure 1.)

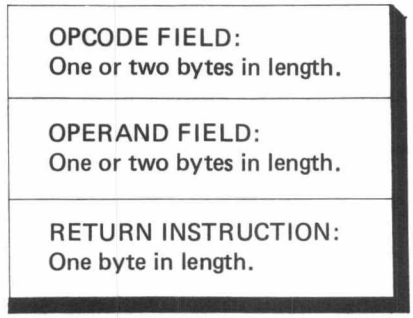


FIGURE I. The general form of a "mini-program" as presented in this article. Note that Z80 instructions may use two byte opcode fields. Two byte operand fields occur more often for both Z80 and 8080 instructions.

It is the modification of the operand field of the LHLD instruction which allows us to create the indirect load. The LHLI routine stores the address provided on entry into the operand field. Then it executes the "mini-program" with a jump (JMP) instruction. The HL pair is loaded with the contents of the provided address and then the program RETurns to the address found on the top of the hardware stack. Of course, this is the return address to the main program and execution continues there.

Listing IIb contains the routine $LHLI which performs the same function as LHLI in Listing IIa. However, because it uses the self-modifying technique, it is shorter and faster than LHLI. Essentially, the "mini-program" that was executed through LHLI is done in-line. The first SHLD (store HL direct) stores the address which the main program provided. It is stored into the operand field of the following LHLD instruction which proceeds to load HL with the contents of the given memory address. The self-modifying version looks more inviting than its predecessor. However, it should be noted that this technique is not available when programs are stored in Read Only Memory. Additionally, as pointed out by Mr. Skier, the method can be volatile and potentially cause disastrous results to a program if used incorrectly. Finally, Listing IIc is the macro for $LHLI. It provides a means for direct in-line use of the self-modifying routine directly within the main program.

Sometimes, it is also advantageous to perform indirect stores into memory. In this case, it is not practical to have the register pair also serve as the indirection pointer (i.e., when would (HL):=HL be useful?). Instead the DE pair may be used as in listings IIIa, IIIb, and IIIc, and IIId (the macro for $SHLID). The routine SHLIDE takes six bytes of program memory while the method employing the "mini-program" in SCRATCH is about three times as long and consequently slower. Analogous routines are possible for the Z80 type load-store direct instructions (LDED, LSPD, SBCD, etc.) where the two byte opcode must be accounted for when creating the SCRATCH "mini-program" as well as in the self-modifying versions.

Indirection For Input-Output

The 8080's I/O instructions are limited to direct mode with accumulator addressing. That is, the ports referenced are always constants and the data is either sent from the A register or loaded into it depending whether the instruction is an OUT or an IN. Once again it is possible to circumvent this with either self-modifying code or the SCRATCH technique which has been used above. Listing IVa presents IND, a routine which creates a "mini-program" and executes it in a fashion analogous to LHLI. IND expects the port to input from the accumulator and loads the accumulator with the returned value. Listing IVb presents $IND which is the self-modifying version of IND while Listing IVc is the macro version of IND for in-line use in the main program. (Once again the same caution must apply to the use of the self-modifying code.) Listing Va contains OUTD, a routine to effect indirect output instructions for the 8080. Again the SCRATCH buffer technique is used. This time, data is sent in the B register and the accumulator contains the port number. Listing Vb is the self-modifying version $OUTD while Listing Vc is the macro routine, OUTD, which is analogous to Listing IVc. It is important to note that Z80 users have no need for this type of coding for indirect I/O since it is provided in the Z80's instruction set. Any of the general purpose registers or the accumulator may be used for the data pointer while the C register holds the port number. Special block input and output instructions (OUTI, INIR, etc.) are also provided.

Z80 Indirect Indexing Mode

The Z80 provides eight bit signed two's complement indexed addressing on its IX and IY registers. The indexing modes involve many varieties of operations, but the basic drawback in some cases is that the index is fixed. However, it is possible to create routines similar to those previously presented to alleviate this problem. Fortunately, the index offset byte is always the third byte of the index instruction no matter which one it is. Therefore, one example of a "mini-program" and self-modifying code should serve as examples for the reader's own possible use. Listing VIa, VIb and VIc all perform an indexed MOV instruction while the second is self-modifying and the latter is the macro instruction. The following section is a specfic example which uses the macro IXMOV of Listing VIc to illustrate how these "software generated" indirect addressing modes may be used to increase program efficiency.

A Specific Example

Listing VIIa contains a portion of a LIFE program that I wrote some time ago. The routine MCOUNT was designed to lookup the number of neighbors of each life cell on the video display screen which it uses and to save the number in a one kilobyte storage area for later access in determining the fate of the next LIFE "generation." The point of including this routine here is that it uses the indexed register MOV instructions of the Z80 to access the different neighbors' memory locations relative to the cell under inspection. However, each of the eight checks requires an indexed MOV followed by a CALL (the CELL macro expansions) to the neighbor tally routine, CHECK (see Listing VIIc). It would be preferable to write a loop which performs each location check in eight iterations. However, this is not possible with the use of the direct indexed addressing mode. Thus, it is necessary to incorporate the macro instruction sequence, IXMOV, into the program in order to make the routine more efficient. This allows the neighbor checks to be done in an eight count loop instead of as a series of eight MOV-CALL (CELL macro) sequences. Listing VIIb demonstrates the recoding of the NCOUNT function to include the IXMOV macro sequence and an inner loop. The recoded version, $NCONT, uses about twenty-two less bytes than the previous routine, NCOUNT. As hoped, the routine utilizing the indirect mode macro is more efficient than its predecessor. (Note that Listing VIIc contains references used by both NCOUNT and $NCONT.)

Using Restart Vectors

I'd like to make one last point about the use of the self-modifying routines. They are usually short enough to use in one of the 8080 and Z80 type RST vectors located in page zero. If a particular type of instruction is called frequently by the main program, it may pay off in terms of throughput and memory to utilize a RST call to execute the special indirect instruction. RST instructions perform one byte CALLs, and at a saving of two bytes per CALL, this may save memory space where short, or help speed up a routine which is inherently time related.

Conclusion

In this article the implementation of indirect addressing modes for certain 8080 and Z80 instructions has been discussed. This included indirect memory addressing for sixteen bit registers (specifically HL), indirect I/O addressing, indirect subroutine referencing and indirect indexed addressing for the Z80's IX and IY registers. As well, the use of the 8080-type RST vectors was touched on as a possible code and time saver in implementing these ideas.

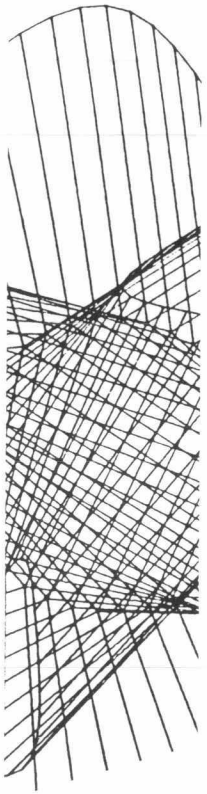
BIBLIOGRAPHY

1. *Skier, Kenneth, "Indirect Addressing for the 6502," BYTE, January 1980, pp. 118, 119.*

LISTINGS

NOTE: The listings provided with this article are written with the TDL MACRO ASSEMBLER (XITAN) and consequently use the special TDL Z80 mnemonics and pseudo-ops. Conversion to other assembler formats should be rather straightforward.

```
;               LISTING Ia.


;*********************************************************
;*                                                       *
;*      ICALL                                            *
;*                                                       *
;*      Function:       Perform indirect CALL of         *
;*                      target subroutine addressed      *
;*                      by HL on entry.                  *
;*                                                       *
;*      Expects:        HL with appropriate address      *
;*      Returns:        Subroutine CALLed.               *
;*      Consumes:       None (although subroutine may)   *
;*                                                       *
;*********************************************************
        ;
        ;
ICALL:  PCHL            ; PC:=HL;


;               LISTING Ib.


;*********************************************************
;*                                                       *
;*      ICALL2                                           *
;*                                                       *
;*      Function:               Do indirect CALL         *
;*                              as ICALL but this        *
;*                              time HL must contain     *
;*                              valid data for           *
;*                              target subroutine.       *
;*                                                       *
;*      Expects:        HL with address of target        *
;*                      subroutine.                      *
;*                      Contents of MEMW with data       *
;*                      to be in HL for the target       *
;*                      subroutine.                      *
;*      Returns:        Subroutine CALLed.               *
;*      Consumes:       HL and subroutine may consume    *
;*                      other registers                  *
;*                                                       *
;*********************************************************
        ;
ICALL2: PUSH    H               ; place address of target routine
                                ; on stack.  Later will use RET
                                ; to get there
        LHLD    MEMW            ; get proper contents of HL
        RET                     ; execute target routine
```



```
;               LISTING IIa.


;*********************************************************
;*                                                       *
;*      LHLI                                             *
;*                                                       *
;*      Function:       Load HL with the contents        *
;*                      of the address initially         *
;*                      pointed to by the HL pair.       *
;*      HL:=(HL)                                         *
;*                                                       *
;*      Expects:        HL with appropriate address      *
;*      Returns:        Contents of given address        *
;*                      back in HL.                      *
;*      Consumes:       HL,(SCRATCH),(SCRATCH+1),        *
;*                         (SCRATCH+2),(SCRATCH+3)       *
;*                                                       *
;*********************************************************
        ;
LHLI:   SHLD    SCRATCH+1               ; store the given address in
                                        ; what will be the operand
                                        ; field of the "mini-program"
                                        ; LHLD.

        LXI     H,SCRATCH               ; point to opcode memory location
        MVI     M,LOAD                  ; put in binary code for LHLD
        LXI     H,SCRATCH+3             ; point to RET location
        MVI     M,RETN                  ; put in RET instruction
        JMP     SCRATCH                 ; execute "mini-program" and then
                                        ; return to CALLing program.
RETN    =       0C9H                    ; value of RET instruction
LOAD    =       02AH                    ; value of LHLD instruction


;               LISTING IIb.


;*********************************************************
;*                                                       *
;*      $LHLI                                            *
;*                                                       *
;*      Function:       Same as LHLI: HL:=(HL)           *
;*                                                       *
;*      *** This Routine is self-modifying ***           *
;*                                                       *
;*      Expects:        HL points to appropriate         *
;*                      address.                         *
;*      Returns:        Contents memory pointed to       *
;*                      by HL on entry.                  *
;*      Consumes:       HL                               *
;*                                                       *
;*                                                       *
;*********************************************************
        ;
$LHLI:  SHLD    .+4                     ; store address to operand field
                                        ; of next LHLD
        LHLD    0                       ; load from the appropriate
                                        ; address
        RET
        ;
```

; LISTING IIc.

```
;*************************************************************
;*                                                           *
;*      MACRO: LHLI                                          *
;*                                                           *
;*      This macro places the text of the routine            *
;*      $LHLI (minus the RET instruction) directly           *
;*      into the program code.  This is feasible             *
;*      because of its shortness, but it is self-            *
;*      modifying.                                           *
;*                                                           *
;*      Expects:        HL with appropriate address          *
;*                      to load from.                        *
;*      Returns:        HL:=(HL)                             *
;*      Consumes:       HL                                   *
;*      Calls:          no other MACROS                      *
;*                                                           *
;*************************************************************

        ;
        .define LHLI    =
        [       SHLD    .+4
                LHLD    0                       ]
        ;
```

; LISTING IIIa.

```
;*************************************************************
;*                                                           *
;*      SHLIDE                                               *
;*                                                           *
;*      Function:       (DE):=HL                             *
;*                                                           *
;*      Expects:        DE with appropriate address.         *
;*                      HL with appropriate value.           *
;*      Returns:        Memory pointed to by DE set          *
;*                      equal to HL.                         *
;*      Consumes:       None                                 *
;*                                                           *
;*************************************************************
        ;       HL= value, DE= address on entry
        ;
SHLIDE: XCHG                    ; HL = address, DE = value
        MOV     M,E             ; put in low byte
        INX     H               ; point to memory for high byte
        MOV     M,D             ; put in high order byte
        DCX     H               ; restore HL to original address value
        XCHG                    ; restore HL=value, DE=address
        RET
        ;
```

; LISTING IIIb.

```
;*************************************************************
;*                                                           *
;*      XSHLID                                               *
;*                                                           *
;*      Function:       Same as SHLIDE but with              *
;*                      "mini-program" technique             *
;*                      utilized instead.                    *
;*                                                           *
;*                                                           *
;*      Expects:        DE with address, HL with value       *
;*      Returns:        (DE):=HL                             *
;*      Consumes:       (SCRATCH),(SCRATCH+1),               *
;*                      (SCRATCH+2),(SCRATCH+3)              *
;*                                                           *
;*************************************************************
        ;
XSHLID: XCHG                    ; DE=value, HL=address
        SHLD    SCRATCH+1       ; store HL in operand field of future
                                ; SHLD
        XCHG                    ; HL=value, DE=address
        PUSH    H
        LXI     H,SCRATCH       ; point to first byte of "mini-program"
        MVI     M,STOR          ; set up SHLD
        LXI     H,SCRATCH+3     ; point to where RET must go
        MVI     M,RETN          ; store RET instruction
        POP     H               ; restore value
        JMP     SCRATCH         ; perform indirect store and return
        ;
STOR    =       22H             ; value of SHLD opcode
RETN    =       0C9H            ; value of RET opcode
```

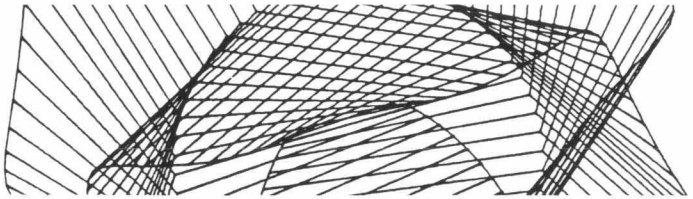
; LISTING IIIc.

```
;*************************************************************
;*                                                           *
;*      $SHLID                                               *
;*                                                           *
;*      Function:       (DE):=HL                             *
;*                                                           *
;*      *** This is self-modifying ***                       *
;*                                                           *
;*      Expects:        DE with appropriate address          *
;*                      HL with appropriate value            *
;*                                                           *
;*      Returns:        (DE) set equal to HL                 *
;*      Consumes:       none                                 *
;*                                                           *
;*************************************************************
        ;
$SHLID: XCHG                    ; HL=address, DE=value
        SHLD    .+5             ; store to operand field of
                                ; next SHLD
        XCHG                    ; HL=value, DE=address
        SHLD    0               ; perform store
        RET
        ;
```

```
;             LISTING IIId.


;*********************************************************
;*                                                       *
;*      MACRO SHLID                                      *
;*                                                       *
;*      Function is the same as $SHLID.                  *
;*                                                       *
;*      Expects:        DE with store address,           *
;*                      HL with data.                    *
;*      Returns:        (DE):=HL.                        *
;*      Consumes:       none.                            *
;*                                                       *
;*********************************************************
        ;
        .define SHLID   =
        [       XCHG                    ; DE=data, HL=address
                SHLD    .+5             ; store to field of next SHLD
                XCHG                    ; DE=address, HL=data
                SHLD    0               ; store to correct address
                                        ]
        ;
```

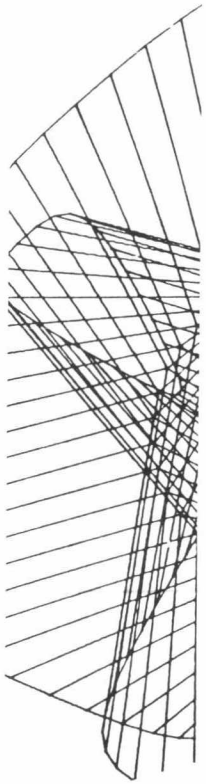
```
;             LISTING IVa.


;*********************************************************
;*                                                       *
;*      IND                                              *
;*                                                       *
;*      Function:       Get A with the contents of       *
;*                      the input port addressed by      *
;*                      A on entry.                      *
;*      Expects:        A with appropriate port #.       *
;*      Returns:        A with input value.              *
;*      Consumes:       A,(SCRATCH),(SCRATCH+1)          *
;*                      (SCRATCH+2)                      *
;*                                                       *
;*                                                       *
;*********************************************************
        ;
IND:
        PUSH    H                       ; will use HL in this routine
        LXI     H,SCRATCH               ; point to beginning of "mini-program"
        MVI     M,INCODE                ; place IN opcode
        INX     H                       ; point to next location
        MOV     M,A                     ; place port #
        INX     H                       ; point to last location
        MVI     M,RETN                  ; place RET instruction
        JMP     SCRATCH                 ; execute it.

        ;

RETN    =       0C9H                    ; value of RET opcode
INCODE  =       0DBH

        ;
```

```
;             LISTING IVb.


;*********************************************************
;*                                                       *
;*      $IND                                             *
;*                                                       *
;*      Function:       Get A with contents of input     *
;*                      port addressed by A on entry.    *
;*                                                       *
;*      *** This is self-modifying ***                   *
;*                                                       *
;*      Expects:        A with appropriate input port    *
;*                      number.                          *
;*                                                       *
;*      Returns:        A with input value from          *
;*                      specified port.                  *
;*      Consumes:       A                                *
;*                                                       *
;*********************************************************
        ;
$IND:   STA     .+4             ; store input port # to operand field
                                ; of next IN instruction
        IN      0               ; input from appropriate port
        RET
        ;
```



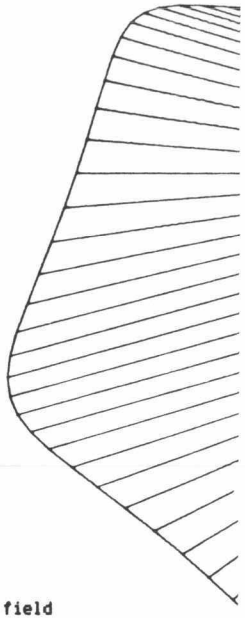
```
;             LISTING IVc.


;*********************************************************
;*                                                       *
;*      MACRO IND                                        *
;*                                                       *
;*      Function is the same as $IND, IND and this       *
;*      also uses the self-modifying technique but for   *
;*      direct in-line use in programs.                  *
;*                                                       *
;*      Expects:                A with port #.           *
;*      Returns:                A with value of input    *
;*                              port.                    *
;*      Consumes:               A                        *
;*                                                       *
;*********************************************************
        ;
        .define IND     =
        [       STA     .+4             ;; store input port # to operand
                                        ;; field of next IN instruction.

                IN      0               ;; input from appropriate port.

                                        ]
        ;
```

```
;               LISTING Va.

;*********************************************************
;*                                                       *
;*      OUTD                                             *
;*                                                       *
;*      Function:       Send data in B register          *
;*                      to port spec. by A reg.          *
;*                                                       *
;*      Expects:        A with port #, B with            *
;*                      appropriate data.                *
;*      Returns:        Data sent.                       *
;*      Consumes:       A,(SCRATCH),(SCRATCH+1)          *
;*                      (SCRATCH+2)                      *
;*                                                       *
;*                                                       *
;*********************************************************
        ;
OUTD:
PUSH    H                       ; we need HL as a pointer
LXI     H,SCRATCH               ; point to beginning of
                                ; "mini-program" area.

MVI     M,OUTC                  ; first place OUT opcode
INX     H                       ; point to next location
MOV     M,A                     ; then imbed port #
INX     H                       ; point to next location
MVI     M,RETN                  ; and finally place RET opcode
MOV     B,A                     ; place output data in A.
POP     H                       ; recover HL's correct value
JMP     SCRATCH                 ; execute "mini-program"
;
OUTC    =       0D3H            ; value of OUT opcode
RETN    =       0C9H            ; value of RET opcode
;

;               LISTING Vb.

;*********************************************************
;*                                                       *
;*      $OUTD                                            *
;*                                                       *
;*      Function:       Get A with contents of output    *
;*                      port addressed by A on entry.    *
;*                                                       *
;*      *** This is self-modifying ***                   *
;*                                                       *
;*      Expects:        Data in B and output port # in   *
;*                      the accumulator (A).             *
;*                                                       *
;*      Returns:        Data outputted to appropriate    *
;*                      port.                            *
;*      Consumes:       A                                *
;*                                                       *
;*********************************************************
        ;
$OUTD:  STA     .+5             ;; store output port # to operand
                                ;; field of next IN instruction.
        MOV     A,B             ;; get data in A
        OUT     0               ;; output from appropriate port.

        RET
        ;
```

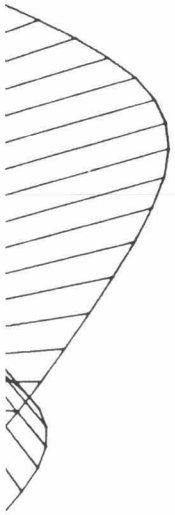


```
;               LISTING Vc.

;*********************************************************
;*                                                       *
;*      MACRO OUTD                                       *
;*                                                       *
;*      Function is the same as $OUTD, OUTD and this     *
;*      also uses the self-modifying technique but for   *
;*      direct in-line use in programs.                  *
;*                                                       *
;*      Expects:                Data in REG,  number     *
;*                              of output port in A.     *
;*      Returns:                Data outputted to        *
;*                              correct port.            *
;*      Consumes:               A                        *
;*                                                       *
;*      Note: REG is the register specified in the       *
;*      macro expression and may be B,C,D,E,H,L .        *
;*      Using A would not give practical results.        *
;*                                                       *
;*********************************************************
        ;
        .define OUTD[REG]       =
        [       STA     .+5             ;; store output port # to operand
                                        ;; field of next IN instruction.

                MOV     A,REG           ;; get data in A
                OUT     0               ;; output from appropriate port.

                                        ]
        ;

;               LISTING VIa.

;*********************************************************
;*                                                       *
;*      IXMOV   Indexed indirect move                    *
;*                                                       *
;*      Function:               Load the A register      *
;*                              With the index in A      *
;*                              on index register X.     *
;*                                                       *
;*      Expects:                A with index, X with     *
;*                              appropriate address.     *
;*      Returns:                A= (index+ X)            *
;*      Consumes:               A,(SCRATCH),(SCRATCH+1)  *
;*                              (SCRATCH+2),(SCRATCH+3)  *
;*                                                       *
;*********************************************************
        ;
IXMOV:  PUSH    H                       ; save HL pair
        LXI     H,SCRATCH               ; point to start of "mini-program"
        MVI     M,XCODE                 ; X register opcode prefix
        INX     H
        MVI     M,MOVAM                 ; stick in "MOV A,M" opcode
                                        ; which is the second byte of
                                        ; the direct index mode opcode
```

```
        INX     H
        MOV     M,A                     ; place offset
        INX     H
        MVI     M,RETN                  ; place RET instruction
        JMP     SCRATCH                 ; execute it.

        ;
RETN    =       0C9H                    ; value of RET opcode
MOVAM   =       07EH                    ; value of MOV A,M opcode
XCODE   =       0DDH                    ; value of IX register prefix.

        ;
```



```
;               LISTING VIb.


;*********************************************************
;*                                                       *
;*      $IXMOV                                           *
;*                                                       *
;*      Function:       Same as IXMOV but this is        *
;*                      Self-modifying.                  *
;*                                                       *
;*      Expects:        A with index, X with appropr.    *
;*                      address.                         *
;*      Returns:        A = (index + X)                  *
;*      Consumes:       A                                *
;*                                                       *
;*********************************************************
        ;
$IXMOV: STA     .+5                     ; store index to operand field
                                        ; of next MOV indexed instruction.

        MOV     A,0(X)                  ; load A with appropriate value.
        RET
        ;
```



```
;               LISTING VIc.


;*********************************************************
;*                                                       *
;*      MACRO IXMOV                                      *
;*                                                       *
;*      *** Self-modifying ***                           *
;*                                                       *
;*      Function:       Inline version of $IXMOV         *
;*                      but any REG may be spec.         *
;*                      to receive the indexed           *
;*                      value returned.                  *
;*      Expects:        A with offset, X with address    *
;*      Returns:        REG = (index +X)                 *
;*      Consumes:       REG                              *
;*                                                       *
;*      Note:   REG may be A,B,C,D,E,H,L                 *
;*                                                       *
;*********************************************************
        ;
```

```
.define IXMOV[REG]      =
[       STA     .+5
        MOV     REG,0(X)                        ]
;
;
;


;               LISTING VIIa.


;*********************************************************
;*                                                       *
;*      NCOUNT:                 A neighbor counter       *
;*                              algorithm used in        *
;*                              LIFEZ by A. Skjellum.    *
;*                                                       *
;*      Uses LISTING VIIc.                               *
;*                                                       *
;*      This routine assumes a FLASHWRITER video         *
;*      interface at 0D000H in memory and checks         *
;*      each location for the number of surrounding      *
;*      live "cells" in order to accomplish LIFE         *
;*      simulation game.                                 *
;*                                                       *
;*      Note:   The macro CELL expands to a six          *
;*              byte sequence each time it is called     *
;*              and loads the accumulator with           *
;*              the contents of the specific neighbor    *
;*              location surrounding the current cell.   *
;*              It then CALLs the subroutine check       *
;*              which handles the actual neighbor        *
;*              counting.                                *
;*                                                       *
;*      This version is to exemplify code which does     *
;*      not use the "software-generated" indirect mode.  *
;*      The routine $NCONT does use this technique       *
;*      and is appreciably more efficient.               *
;*                                                       *
;*********************************************************
        ;
NCOUNT: LXI     X,SCRN          ; point to video display memory
        LXI     D,KILO          ; cell count i.e. 1024
        LXI     H,RESERV        ; place neighbor number in associated
                                ; reserve memory to be checked later.
REP:    MVI     C,0             ; zero neighbor count
        CELL    UL              ; upper left corner
        CELL    UM              ; upper middle
        CELL    UR              ; upper right corner
        CELL    ML              ; middle left
        CELL    MR              ; middle right
        CELL    LL              ; lower left corner
        CELL    LM              ; lower middle
        CELL    LR              ; lower right corner
        MOV     M,C             ; store number of neighbors at cell location
                                ; on screen in the associated reserve memory.
        INX     H               ; point to next reserve location
        INX     X               ; address next cell
        DCX     D               ; decrement count
        MOV     A,E             ; verify if both D and E are zero
        ORA     D               ; and we're done if they are.
        JRNZ    REP             ; if not, continue.
        ;
        ; ... program continues ....
        ;
```

```
;               LISTING VIIb.


;*********************************************************
;*                                                       *
;*      $NCONT                                           *
;*                                                       *
;*      Uses    LISTING VIIc.                            *
;*                                                       *
;*      This example performs the same function          *
;*      as NCOUNT.                                       *
;*                                                       *
;*      This uses the self-modifying macro IXMOV         *
;*      from Listing VIc.                                *
;*                                                       *
;*      In this version, an index offset table is        *
;*      used to obtain the necessary offset values.      *
;*                                                       *
;*********************************************************

        ;
$NCONT: LXI     X,SCRN                  ; point to video display memory
        LXI     D,KILO                  ; cell count i.e. 1024
        LXI     H,RESERV                ; place neighbor number for each
                                        ; cell in an associated byte of
                                        ; memory where it can be accessed
                                        ; later.
REP:    PUSH    H                       ; we'll use HL as a temporary so
                                        ; save its other value
        LXI     B,0800H                 ; let B:=8 (loop counter) and
                                        ; C:=0, the initial neighbor count
                                        ; as before.
        LXI     H,TABL                  ; point to index table
REP1:   MOV     A,M                     ; fetch index
        IXMOV   A                       ; let A:=(index +X) ** MACRO **
        CALL    CHECK                   ; call CHECK, the actual neighbor
                                        ; counter ( in LISTING VIIc. )
        INX     H                       ; point to next index
        DJNZ    REP1                    ; repeat 8 times for all neighbors
        POP     H                       ; HL goes back to RESERV table
                                        ; value.
        MOV     M,C                     ; store neighbor count in associated
                                        ; reserve memory.
        INX     H                       ; point to next reserve location
        INX     X                       ; point to next cell location
        DCX     D                       ; decrement count
        MOV     A,E                     ; verify if D and E are both equal
                                        ; to zero.  If so, we're done.
        ORA     D                       ; check.
        JRNZ    REP                     ; if not, continue.
        ;
        ; ... program continues ...
        ;
        ;
TABL:   .BYTE   UL                      ; upper left index
        .BYTE   UM                      ; upper middle
        .BYTE   UR                      ; upper right
        .BYTE   ML                      ; middle left
        .BYTE   MR                      ; middle right
        .BYTE   LL                      ; lower left
        .BYTE   LM                      ; lower middle
        .BYTE   LR                      ; lower right
        ;
        ;
        ;
```

```
;               LISTING VIIc.


        ;
        ;*********************
        ;*                   *
        ;* Equates           *
        ;*                   *
        ;*********************
        ;
UL      =       -65
UM      =       -64
UR      =       -63
ML      =       -01
MR      =       +01
LL      =       +63
LM      =       +64
LR      =       +65
        ;
SCRN    =       0D000H          ; hardware consideration
KILO    =       1024            ; number of cells to check on screen.
STAR    =       '*'             ; value of STAR character
        ;
        ;
        ;*********************
        ;*                   *
        ;* Support           *
        ;*                   *
        ;*********************
        ;
        ; Routine: Check.
        ;
        ; Handles the actual neighbor counting.
        ;
CHECK:  CPI     STAR            ; was neighbor a star
        RNZ                     ; if not don't do anything.
        INR     C               ; count another live neighbor
        RET                     ; and return.
        ;
        ;*********************
        ;*                   *
        ;* MACRO: CELL       *
        ;*                   *
        ;*********************
        ;
        .define CELL[POSIT]     =
        [       MOV     A,POSIT(X)
                CALL    CHECK                   ]
        ;
        ;
```

THE END

(Continued from pg. 10)

```
72F2  CBAF      0543          RES     5,A
72F4  1F        0544 PFL1     RRA                     ; CHECK MSD
72F5  1F        0545          RRA
72F6  1F        0546          RRA
72F7  1F        0547          RRA
72F8  E60E      0548          AND     0EH             ; MASK MSD + 1
72FA  CB4F      0549          BIT     1,A             ; TEST FOR SET/RES
72FC  4F        0550          LD      C,A             ; FORM TABLE INDEX
72FD  FD09      0551          ADD     IY,BC
72FF  FD4E00    0552          LD      C,(IY+0)        ; GET MNEM CODE
7302  FD7E01    0553          LD      A,(IY+1)        ; GET MASK
7305  2006      0554          JR      NZ,PFL2-$
7307  2F        0555          CPL                     ; DO CLEARS
7308  DDA601    0556          AND     (IX+1)
730B  1803      0557          JR      PFL3-$
730D  DDB601    0558 PFL2     OR      (IX+1)          ; DO SETS
7310  DD7701    0559 PFL3     LD      (IX+1),A        ; AND STUFF IT
7313  0604      0560 ENDIT    LD      B,4
7315  C3D674    0561          JP      END
                0562 ;
                0563 ; PROCESS ALL INSTRUCTIONS HAVING MULTIPLE ADDRESSING
                0564 ;
7318  01D474    0565 MLTMOD   LD      BC,MLTEND       ; PUT RETURN ON STACK
731B  C5        0566          PUSH    BC
731C  7E        0567          LD      A,(HL)          ; GET INSTR TO A
731D  4F        0568          LD      C,A             ; AND TO C
731E  CB39      0569          SRL     C               ; TEST EVEN/ODD
7320  F5        0570          PUSH    AF
7321  CB39      0571          SRL     C               ; ROTATE MSD INTO LSD
7323  CB39      0572          SRL     C
7325  CB39      0573          SRL     C
7327  CB81      0574          RES     0,C             ; MASK LSB
7329  F1        0575          POP     AF
732A  FD21887C  0576          LD      IY,EVNTBL       ; JUMP PER TABLES
732E  D2DA75    0577          JP      NC,JPIB
7331  FD21987C  0578          LD      IY,ODDTBL
7335  C3DA75    0579          JP      JPIB
                0580 ;
                0581 ; MULTI-MODE INSTRUCTION PROCESSORS
                0582 ;
7338  CD1A75    0583 PASL     CALL    MODBBB          ; GET MODE,ADDR,&DATA
733B  CB27      0584          SLA     A               ; ASL
733D  CDE374    0585          CALL    SINSET          ; SET/RES S,Z,C FLGS
7340  CDD875    0586          CALL    PUTDAT          ; RESTORE DATA
7343  3E08      0587          LD      A,8
7345  C9        0588          RET
7346  CB4F      0589 PBTRL    BIT     1,A             ; BIT OR ROL?
7348  2810      0590          JR      Z,PBIT-$
734A  CD1A75    0591          CALL    MODBBB          ; ROL
734D  CDDB74    0592          CALL    CARSET          ; SET UP C FLAG
7350  17        0593 PROL1    RLA                     ; DO ROTATE
7351  CDE374    0594          CALL    SINSET
7354  CDD875    0595          CALL    PUTDAT
7357  3E9C      0596          LD      A,9CH
7359  C9        0597          RET
735A  DDCB01B6  0598 PBIT     RES     6,(IX+1)        ; BIT
735E  CD3275    0599          CALL    MODBBX
7361  C5        0600          PUSH    BC
7362  47        0601          LD      B,A             ; SAVE IT IN B
7363  DD7E00    0602          LD      A,(IX+0)        ; GET SIM ACC
7366  A0        0603          AND     B               ; AND IT
7367  CB77      0604          BIT     6,A             ; CHECK V BIT
7369  2804      0605          JR      Z,PBIT1-$
736B  DDCB01F6  0606          SET     6,(IX+1)        ; SET V BIT IF REQD
736F  CDED74    0607 PBIT1    CALL    SINS1           ; SET/RES S,Z FLGS
7372  C1        0608          POP     BC
7373  3E18      0609          LD      A,18H
7375  C9        0610          RET
7376  CD1A75    0611 PLSR     CALL    MODBBB          ; LSR
7379  CB3F      0612          SRL     A
737B  CDE374    0613          CALL    SINSET
737E  CDD875    0614          CALL    PUTDAT
7381  3E80      0615          LD      A,80H
7383  C9        0616          RET
7384  CD1A75    0617 PROR     CALL    MODBBB          ; ROR
7387  CDDB74    0618          CALL    CARSET          ; SET UP C FLG
738A  1F        0619 PROR1    RRA                     ; DO ROTATE
738B  CDE374    0620          CALL    SINSET
738E  CDD875    0621          CALL    PUTDAT
7391  3EA0      0622          LD      A,0A0H
7393  C9        0623          RET
7394  CB4F      0624 PSTXY    BIT     1,A             ; WHICH ONE?
7396  280C      0625          JR      Z,PSY-$
7398  CD3275    0626          CALL    MODBBX          ; STX
739B  DD7E02    0627          LD      A,(IX+2)
739E  CDD875    0628          CALL    PUTDAT
73A1  3EC0      0629          LD      A,0C0H
73A3  C9        0630          RET
73A4  CD3275    0631 PSY      CALL    MODBBX          ; STY
73A7  DD7E03    0632          LD      A,(IX+3)
73AA  CDD875    0633          CALL    PUTDAT
73AD  3EC4      0634          LD      A,0C4H
73AF  C9        0635          RET
73B0  CB4F      0636 PLDXY    BIT     1,A             ; WHICH ONE?
73B2  280C      0637          JR      Z,PLY-$
73B4  CD2075    0638          CALL    MODDDD          ; LDX
73B7  DD7702    0639          LD      (IX+2),A
73BA  CDED74    0640          CALL    SINS1
73BD  3E78      0641          LD      A,78H
73BF  C9        0642          RET
73C0  CD2075    0643 PLY      CALL    MODDDD          ; LDY
73C3  DD7703    0644          LD      (IX+3),A
73C6  CDED74    0645          CALL    SINS1
73C9  3E7C      0646          LD      A,7CH
73CB  C9        0647          RET
73CC  CB4F      0648 PDCY     BIT     1,A             ; WHICH ONE?
73CE  2827      0649          JR      Z,PCPY-$
73D0  CD3275    0650          CALL    MODBBX          ; DEC
73D3  3D        0651          DEC     A
73D4  CDED74    0652          CALL    SINS1
73D7  CDD875    0653          CALL    PUTDAT
73DA  3E50      0654          LD      A,50H
73DC  C9        0655          RET
73DD  CB4F      0656 PICX     BIT     1,A             ; WHICH ONE?
73DF  280D      0657          JR      Z,PCPX-$
73E1  CD3275    0658          CALL    MODBBX          ; INC
73E4  3C        0659          INC     A
73E5  CDED74    0660          CALL    SINS1
73E8  CDD875    0661          CALL    PUTDAT
73EB  3E60      0662          LD      A,60H
73ED  C9        0663          RET
73EE  CD0474    0664 PCPX     CALL    CXY             ; CPX
73F1  CDE374    0665          CALL    SINSET
73F4  3E48      0666          LD      A,48H
73F6  C9        0667          RET
73F7  DD23      0668 PCPY     INC     IX
```

```
73F9  CD0474    0669          CALL    CXY             ; CPY
73FC  DD2B      0670          DEC     IX
73FE  CDE374    0671          CALL    SINSET
7401  3E4C      0672          LD      A,4CH
7403  C9        0673          RET
7404  CD4375    0674 CXY      CALL    MODCC
7407  C5        0675          PUSH    BC
7408  47        0676          LD      B,A             ; MEM DAT INTO B
7409  DD7E02    0677          LD      A,(IX+2)        ; X (Y) INTO A
740C  90        0678          SUB     B
740D  3F        0679          CCF                     ; BORROW = 6502 /C
740E  C1        0680          POP     BC
740F  C9        0681          RET
7410  DDCB01B6  0682 PSBC     RES     6,(IX+1)        ; SBC...CLR V FLG
7414  CD1475    0683          CALL    MODAAA
7417  C5        0684          PUSH    BC
7418  47        0685          LD      B,A             ; DATA TO B
7419  DD7E00    0686          LD      A,(IX+0)        ; ACC TO A
741C  CDDB74    0687          CALL    CARSET          ; SET UP C FLAG
741F  3F        0688          CCF                     ; AND COMPLEMENT IT
7420  DDCB015E  0689 PSBC1    BIT     3,(IX+1)        ; CHECK FOR DEC MODE
7424  2003      0690          JR      NZ,PSDEC-$
7426  98        0691          SBC     A,B
7427  1802      0692          JR      PSBC2-$
7429  98        0693 PSDEC    SBC     A,B             ; DECIMAL MODE
742A  27        0694          DAA
742B  DD7700    0695 PSBC2    LD      (IX+0),A        ; RESULT TO ACCUM
742E  3F        0696          CCF                     ; BORROW = 6502 /C
742F  CDE374    0697          CALL    SINSET
7432  E6C0      0698          AND     0C0H            ; TEST XOR OF B6 & B7
7434  2808      0699          JR      Z,SBC3-$
7436  FEC0      0700          CP      0C0H
7438  2804      0701          JR      Z,SBC3-$
743A  DDCB01F6  0702          SET     6,(IX+1)        ; SET V FLAG IF XOR = 1
743E  C1        0703 SBC3     POP     BC
743F  3EAC      0704          LD      A,0ACH
7441  C9        0705          RET
7442  CD1475    0706 PORA     CALL    MODAAA          ; ORA
7445  C5        0707          PUSH    BC
7446  47        0708          LD      B,A             ; DATA TO B
7447  DD7E00    0709          LD      A,(IX+0)        ; ACC TO A
744A  B0        0710          OR      B
744B  DD7700    0711          LD      (IX+0),A        ; RESULT TO ACC
744E  CDED74    0712          CALL    SINS1
7451  C1        0713          POP     BC
7452  3E88      0714          LD      A,88H
7454  C9        0715          RET
7455  CD1475    0716 PAND     CALL    MODAAA          ; AND
7458  C5        0717          PUSH    BC
7459  47        0718          LD      B,A             ; DATA TO B
745A  DD7E00    0719          LD      A,(IX+0)        ; ACC TO A
745D  A0        0720          AND     B
745E  DD7700    0721          LD      (IX+0),A        ; RESULT TO ACC
7461  CDED74    0722          CALL    SINS1
7464  C1        0723          POP     BC
7465  3E04      0724          LD      A,4
7467  C9        0725          RET
7468  CD1475    0726 PEOR     CALL    MODAAA          ; EOR
746B  C5        0727          PUSH    BC
746C  47        0728          LD      B,A             ; DATA TO B
746D  DD7E00    0729          LD      A,(IX+0)        ; ACC TO A
7470  A8        0730          XOR     B
7471  DD7700    0731          LD      (IX+0),A        ; RESULT TO ACC
7474  CDED74    0732          CALL    SINS1
7477  C1        0733          POP     BC
7478  3E5C      0734          LD      A,5CH
747A  C9        0735          RET
747B  DDCB01B6  0736 PADC     RES     6,(IX+1)        ; ADC: CLEAR V FLAG
747F  CD1475    0737          CALL    MODAAA
7482  C5        0738          PUSH    BC
7483  47        0739          LD      B,A             ; DATA TO B
7484  DD7E00    0740          LD      A,(IX+0)        ; ACC TO A
7487  CDDB74    0741          CALL    CARSET          ; SET UP C FLG
748A  DDCB015E  0742 PADC1    BIT     3,(IX+1)        ; CHECK FOR DEC MODE
748E  2003      0743          JR      NZ,PADEC-$
7490  88        0744          ADC     A,B             ; BINARY MODE
7491  1802      0745          JR      PADC2-$
7493  88        0746 PADEC    ADC     A,B             ; DECIMAL MODE
7494  27        0747          DAA
7495  DD7700    0748 PADC2    LD      (IX+0),A        ; RESULT TO ACC
7498  CDE374    0749          CALL    SINSET
749B  E6C0      0750          AND     0C0H            ; CHECK XOR OF B6 & B7
749D  2808      0751          JR      Z,PADC3-$
749F  FEC0      0752          CP      0C0H
74A1  2804      0753          JR      Z,PADC3-$
74A3  DDCB01F6  0754          SET     6,(IX+1)        ; SET V FLG IF XOR = 1
74A7  C1        0755 PADC3    POP     BC
74A8  3E00      0756          LD      A,0
74AA  C9        0757          RET
74AB  CD1475    0758 PSTA     CALL    MODAAA          ; STA
74AE  DD7E00    0759          LD      A,(IX+0)        ; ACC TO MEM
74B1  CDD875    0760          CALL    PUTDAT
74B4  3EBC      0761          LD      A,0BCH
74B6  C9        0762          RET
74B7  CD1475    0763 PLDA     CALL    MODAAA          ; LDA
74BA  DD7700    0764          LD      (IX+0),A        ; MEM TO ACC
74BD  CDED74    0765          CALL    SINS1
74C0  3E74      0766          LD      A,74H
74C2  C9        0767          RET
74C3  CD1475    0768 PCMP     CALL    MODAAA          ; CMP
74C6  C5        0769          PUSH    BC
74C7  47        0770          LD      B,A             ; DATA TO B
74C8  DD7E00    0771          LD      A,(IX+0)        ; ACC TO A
74CB  90        0772          SUB     B
74CC  3F        0773          CCF                     ; BORROW = 6502 /C
74CD  CDE374    0774          CALL    SINSET
74D0  C1        0775          POP     BC
74D1  3E44      0776          LD      A,44H
74D3  C9        0777          RET
                0778 ;
                0779 ; END OF ALL MULTI MODE PROCESSORS
                0780 ;
74D4  81        0781 MLTEND   ADD     A,C             ; FORM MNEM CODE IN C
74D5  4F        0782          LD      C,A
74D6  23        0783 END      INC     HL              ; POINT TO NXT INSTR
74D7  22647A    0784 END1     LD      (NPC),HL
74DA  C9        0785          RET                     ; BACK TO CMND
                0786 ;
                0787 ; VARIOUS SUBROUTINES FOR TRANSLATOR
                0788 ;
                0789 ; SET Z80 C FLAG TO MATCH 6502 C FLG
                0790 ;
74DB  37        0791 CARSET   SCF
74DC  DDCB0146  0792          BIT     0,(IX+1)
74E0  C0        0793          RET     NZ
74E1  3F        0794          CCF
```

```
74E2  C9          0795          RET
                  0796 ;
                  0797 ; SET SIMULATOR S,Z,C FLAGS PER Z80 RESULTS
                  0798 ;
74E3  DDCB0186    0799 SINSET   RES     0,(IX+1)        ; RESET C FLAG
74E7  3004        0800          JR      NC,SINS1-$
74E9  DDCB01C6    0801          SET     0,(IX+1)        ; SET C PER Z80 C
74ED  DDCB018E    0802 SINS1    RES     1,(IX+1)        ; CLEAR Z FLAG
74F1  B7          0803          OR      A               ; SEE IF ACC = 0
74F2  2004        0804          JR      NZ,SINS2-$
74F4  DDCB01CE    0805          SET     1,(IX+1)        ; SET Z FLG
74F8  DDCB01BE    0806 SINS2    RES     7,(IX+1)        ; CLEAR SIGN FLAG
74FC  CB7F        0807          BIT     7,A             ; CHECK SIGN OF ACC
74FE  C8          0808          RET     Z
74FF  DDCB01FE    0809          SET     7,(IX+1)        ; SET S IF NECESSARY
7503  C9          0810          RET
                  0811 ;
                  0812 ; TRANSFER SIMULATOR SP TO/FROM IY
                  0813 ;
7504  0601        0814 TSPY     LD      B,1             ; XFER SP TO IY
7506  DD4E04      0815          LD      C,(IX+4)
7509  C5          0816          PUSH    BC
750A  FDE1        0817          POP     IY
750C  C9          0818          RET
750D  FDE5        0819 TYSP     PUSH    IY              ; XFER IY TO SP
750F  C1          0820          POP     BC
7510  DD7104      0821          LD      (IX+4),C
7513  C9          0822          RET
                  0823 ;
                  0824 ; ADDRESS MODE DECODERS...RETURN WITH DATA TO BE MODIFIED
                  0825 ; IN A, ADDRESS MODE CODE IN B, # BYTES IN C, ADDRESS OF
                  0826 ; DATA IN DE
                  0827 ;
7514  FD21FA7B    0828 MODAAA   LD      IY,AAATBL       ; GROUP 1
7518  182D        0829          JR      MOD1-$
751A  FD210A7C    0830 MODBBB   LD      IY,BBBTBL       ; GROUP 2
751E  1827        0831          JR      MOD1-$
7520  FD211A7C    0832 MODDDD   LD      IY,DDDTBL       ; GROUP 3
7524  FEB6        0833          CP      0B6H            ; LDX ZPG,Y SPECIAL
7526  2002        0834          JR      NZ,DDD1-$
7528  3E10        0835          LD      A,10H
752A  FEBE        0836 DDD1     CP      0BEH            ; LDX ABS,Y SPECIAL
752C  2019        0837          JR      NZ,MOD1-$
752E  3E18        0838          LD      A,18H
7530  1815        0839          JR      MOD1-$
7532  FD212A7C    0840 MODBBX   LD      IY,BBXTBL       ; GROUP 4 & 6
7536  FE96        0841          CP      96H             ; STX ZPG,Y SPECIAL
7538  2805        0842          JR      Z,BBX-$
753A  1F          0843          RRA                     ; EXTRA SHIFT
753B  E60C        0844          AND     0CH             ; EXTRA MASK
753D  1808        0845          JR      MOD1-$
753F  3E10        0846 BBX      LD      A,10H           ; SPECIAL
7541  1804        0847          JR      MOD1-$
7543  FD21347C    0848 MODCC    LD      IY,CCTBL        ; GROUP 5
7547  1F          0849 MOD1     RRA                     ; ROTATE MODE INTO LSD
7548  E60E        0850          AND     0EH             ; MASK THE REST
754A  0600        0851          LD      B,0             ; FORM OFFSET IN BC
754C  4F          0852          LD      C,A
754D  FD09        0853          ADD     IY,BC           ; ADD TO TABLE START
754F  FD4E00      0854          LD      C,(IY+0)        ; MODE OFFSET
7552  FD4601      0855          LD      B,(IY+1)        ; # CODE
7555  C5          0856          PUSH    BC              ; SAVE FOR LATER
7556  FD21747C    0857 GETDAT   LD      IY,GETTBL       ; JUMP TABLE
755A  C3DA75      0858          JP      JPIB
755D  23          0859 PABY     INC     HL              ; ABS,Y
755E  7E          0860          LD      A,(HL)          ; 2ND BYTE INSTR
755F  DD8603      0861          ADD     A,(IX+3)        ; PLUS Y INDEX
7562  5F          0862          LD      E,A             ; TO E
7563  23          0863          INC     HL
7564  7E          0864          LD      A,(HL)          ; 3RD BYTE INSTR
7565  CE00        0865          ADC     A,0             ; PLUS CARRY
7567  57          0866          LD      D,A             ; TO D
7568  1861        0867          JR      EXIT-$
756A  23          0868 PIMM     INC     HL              ; IMMEDIATE
756B  7E          0869          LD      A,(HL)          ; DATA TO ACC
756C  185E        0870          JR      EX1-$
756E  DD7E00      0871 PACX     LD      A,(IX+0)        ; ACCUM
7571  DDE5        0872          PUSH    IX
7573  D1          0873          POP     DE
7574  1856        0874          JR      EX1-$
7576  23          0875 PZPG     INC     HL              ; ZPG
7577  5E          0876          LD      E,(HL)          ; ADDRESS TO DE
7578  1600        0877          LD      D,0
757A  184F        0878          JR      EXIT-$
757C  23          0879 PAB      INC     HL              ; ABSOLUTE
757D  5E          0880          LD      E,(HL)          ; ADDRESS TO DE
757E  23          0881          INC     HL
757F  56          0882          LD      D,(HL)
7580  1849        0883          JR      EXIT-$
7582  DD7E02      0884 PZX      LD      A,(IX+2)        ; ZPG,X...GET X INDEX
7585  1803        0885          JR      PZ1-$
7587  DD7E03      0886 PZY      LD      A,(IX+3)        ; ZPG,Y...GET Y INDEX
758A  23          0887 PZ1      INC     HL
758B  86          0888          ADD     A,(HL)          ; ADD BASE ADDRESS
758C  5F          0889          LD      E,A             ; PUT IN E
758D  1600        0890          LD      D,0             ; ZERO PAGE, NO WRAP
758F  183A        0891          JR      EXIT-$
7591  23          0892 PZIX     INC     HL              ; (ZPG,X)
7592  7E          0893          LD      A,(HL)          ; 2ND BYTE INSTR
7593  DD8602      0894          ADD     A,(IX+2)        ; ADD X INDEX
7596  5F          0895          LD      E,A             ; TO FORM BASE ADDRESS...
7597  1600        0896          LD      D,0             ; IN ZERO PG, NO WRAP
7599  FD210000    0897          LD      IY,0
759D  FD19        0898          ADD     IY,DE           ; TRANSFER TO IY
759F  FD5E00      0899          LD      E,(IY+0)        ; EFF ADDR TO DE
75A2  FD5601      0900          LD      D,(IY+1)
75A5  1824        0901          JR      EXIT-$
75A7  23          0902 PZIY     INC     HL              ; (ZPG),Y
75A8  5E          0903          LD      E,(HL)          ; BASE ADDRESS
75A9  1600        0904          LD      D,0             ; IN ZERO PAGE
75AB  FD210000    0905          LD      IY,0
75AF  FD19        0906          ADD     IY,DE           ; TRANSFER TO IY
75B1  FD7E00      0907          LD      A,(IY+0)        ; EFF ADDRESS...
75B4  DD8603      0908          ADD     A,(IX+3)        ; PLUS Y INDEX...
75B7  5F          0909          LD      E,A             ; TO DE
75B8  FD7E01      0910          LD      A,(IY+1)
75BB  CE00        0911          ADC     A,0
75BD  57          0912          LD      D,A
75BE  180B        0913          JR      EXIT-$
75C0  23          0914 PABX     INC     HL              ; ABS,X
75C1  7E          0915          LD      A,(HL)          ; 2ND BYTE INSTR
75C2  DD8602      0916          ADD     A,(IX+2)        ; PLUS X INDEX
75C5  5F          0917          LD      E,A             ; TO E
75C6  23          0918          INC     HL              ; 3RD BYTE INSTR
75C7  7E          0919          LD      A,(HL)
75C8  CE00        0920          ADC     A,0             ; PLUS CARRY
```

```
75CA 57       0921          LD      D,A             ; TO E
75CB 1A       0922 EXIT     LD      A,(DE)          ; DATA TO ACC
75CC C1       0923 EX1      POP     BC              ; RESTORE # CODE
75CD F5       0924          PUSH    AF              ; SAVE DATA
75CE 3E03     0925          LD      A,3
75D0 A0       0926          AND     B               ; # BYTES...
75D1 4F       0927          LD      C,A             ; TO C
75D2 CB38     0928          SRL     B
75D4 CB38     0929          SRL     B               ; B=ADDR MODE
75D6 F1       0930          POP     AF              ; RESTORE DATA
75D7 C9       0931          RET
              0932 ;
              0933 ; STORE DATA TO ADDRESS FROM GETDAT
              0934 ;
75D8 12       0935 PUTDAT   LD      (DE),A
75D9 C9       0936          RET
              0937 ;
              0938 ; INDEX INTO JUMP TABLES...ENTER WITH TABLE START
              0939 ; ADDRESS IN IY AND OFFSET IN C.  CALCULATES TABLE
              0940 ; POSITION AND JUMPS TO RESULT
              0941 ;
75DA 0600     0942 JPIB     LD      B,0             ; INDIRECT JUMP PER TABLE
75DC FD09     0943          ADD     IY,BC           ; TABLE ADDRESS TO IY
75DE FD4E00   0944          LD      C,(IY+0)        ; TABLE ENTRY TO BC
75E1 FD4601   0945          LD      B,(IY+1)
75E4 C5       0946          PUSH    BC
75E5 FDE1     0947          POP     IY              ; ENTRY TO IY
75E7 FDE9     0948          JP      (IY)            ; AND GO TO IT
              0949 ;
              0950 ; ELEMENTARY MONITOR FUNCTIONS TO EXAMINE A BLOCK OF
              0951 ; MEMORY LOCATIONS (E) AND TO CHECK AND UPDATE
              0952 ; INDIVIDUAL LOCATIONS (M)
              0953 ;
75E9 11DE7A   0954 EXAM     LD      DE,GETADR       ; EXAM BLOCK OF MEM
75EC CDCE79   0955          CALL    PRTDE
75EF CD6379   0956          CALL    GETWRD          ; GET START ADDR
75F2 D5       0957          PUSH    DE
75F3 CD5479   0958          CALL    LINF
75F6 11F67A   0959          LD      DE,GETN
75F9 CDCE79   0960          CALL    PRTDE
75FC CD6379   0961          CALL    GETWRD          ; GET # BYTES
75FF D5       0962          PUSH    DE
7600 C1       0963          POP     BC
7601 1E04     0964          LD      E,4             ; CHANGE TO # LINES (/16)
7603 AF       0965          XOR     A
7604 CB38     0966 BYTLN    SRL     B
7606 CB19     0967          RR      C
7608 3001     0968          JR      NC,XXX-$
760A 3C       0969          INC     A
760B 1D       0970 XXX      DEC     E
760C 20F6     0971          JR      NZ,BYTLN-$
760E B7       0972          OR      A               ; SEE IF PARTIAL LINE
760F 2801     0973          JR      Z,YYY-$
7611 03       0974          INC     BC
7612 E1       0975 YYY      POP     HL              ; HL = START, BC = COUNT
7613 0B       0976          DEC     BC              ; ONE LESS LINE
7614 C5       0977 EXLP     PUSH    BC              ; LOOP FOR ONE LINE
7615 CD4F79   0978          CALL    CRLF
7618 CD9076   0979          CALL    LADR            ; PRINT ADDRESS AT START
761B CD9D76   0980          CALL    TWO             ; INSERT TWO SPACES
761E E5       0981          PUSH    HL              ; SAVE ADDRESS FOR ALPHA
761F 1E10     0982          LD      E,16            ; 16 ENTRIES PER LINE
7621 7E       0983 LNLP     LD      A,(HL)          ; GET BYTE FROM MEMORY
7622 CD9676   0984          CALL    PACC            ; PRINT IT HEX
7625 CDA076   0985          CALL    ONE             ; SPACE
7628 23       0986          INC     HL              ; POINT TO NEXT BYTE
7629 1D       0987          DEC     E               ; DONE WITH HEX?
762A 20F5     0988          JR      NZ,LNLP-$       ; NO, DO NEXT BYTE
762C CD9D76   0989          CALL    TWO             ; YES, SET UP FOR ALPHA
762F E1       0990          POP     HL              ; SAME STARTING ADDRESS
7630 1E10     0991          LD      E,16
7632 7E       0992 ASCII    LD      A,(HL)          ; GET BYTE
7633 FE20     0993          CP      20H             ; IS IT PRINTABLE?
7635 3804     0994          JR      C,DOT-$
7637 FE80     0995          CP      80H
7639 3802     0996          JR      C,ASOK-$
763B 3E2E     0997 DOT      LD      A,'.'           ; NO, PRINT A DOT
763D CD277A   0998 ASOK     CALL    PRNA
7640 23       0999          INC     HL              ; BUMP POINTER
7641 1D       1000          DEC     E               ; DONE WITH LINE?
7642 20EE     1001          JR      NZ,ASCII-$
7644 C1       1002          POP     BC
7645 78       1003          LD      A,B             ; SEE IF ALL DONE
7646 B1       1004          OR      C
7647 C8       1005          RET     Z               ; YES, RET TO CMND
7648 0B       1006          DEC     BC
7649 18C9     1007          JR      EXLP-$          ; ELSE DO ANOTHER LINE
              1008 ;
              1009 ; SUBSTITUTE MEMORY...ENTER BY 'M'
              1010 ;
764B 11DE7A   1011 SUBM     LD      DE,GETADR       ; ASK FOR ADDRESS
764E CDCE79   1012          CALL    PRTDE
7651 CD6379   1013          CALL    GETWRD          ; GET IT
7654 EB       1014          EX      DE,HL           ; MOVE IT TO HL
7655 CD4F79   1015 SULP     CALL    CRLF
7658 CD9076   1016          CALL    LADR            ; PRINT THE ADDRESS
765B CD9D76   1017          CALL    TWO
765E 7E       1018          LD      A,(HL)
765F CD9676   1019          CALL    PACC            ; PRINT THE CONTENTS
7662 CD9D76   1020          CALL    TWO
7665 CD167A   1021 JLP      CALL    GETCHR          ; GET INPUT
7668 FE2E     1022          CP      2EH             ; EXIT ON "."
766A C8       1023          RET     Z
766B FE20     1024          CP      20H
766D 281E     1025          JR      Z,FWD-$         ; SKIP AHEAD ON "SPACE"
766F CD8979   1026          CALL    CONV            ; CONV HEX IF POSSIBLE
7672 28F1     1027          JR      Z,JLP-$         ; NOT A HEX NUMBER
7674 E5       1028          PUSH    HL              ; SAVE ADDRESS AGAIN
7675 210000   1029          LD      HL,0            ; CLEAR FOR ROTATE
7678 CDAB79   1030 LLP      CALL    ROBYT           ; ROTATE NYBBLE INTO HL
767B CD167A   1031 KLP      CALL    GETCHR          ; GET ANOTHER CHARACTER
767E FE0D     1032          CP      0DH
7680 2807     1033          JR      Z,PUTIN-$       ; IF "CR" STUFF IT
7682 CD8979   1034          CALL    CONV            ; CONV BINARY
7685 28F4     1035          JR      Z,KLP-$         ; NOT A HEX NUMBER
7687 18EF     1036          JR      LLP-$           ; CONTINUE
7689 E5       1037 PUTIN    PUSH    HL              ; DATA TO BC
768A C1       1038          POP     BC
768B E1       1039          POP     HL              ; RETRIEVE ADDRESS
768C 71       1040          LD      (HL),C          ; STUFF THE DATA
768D 23       1041 FWD      INC     HL              ; BUMP POINTER
768E 18C5     1042          JR      SULP-$          ; ...AND CONTINUE
7690 EB       1043 LADR     EX      DE,HL           ; PRINT WORD IN HL
7691 CDB279   1044          CALL    PRTWRD
7694 EB       1045          EX      DE,HL
7695 C9       1046          RET
```

```
7696 D5          1047 PACC   PUSH   DE              ; PRINT BYTE IN A
7697 5F          1048        LD     E,A
7698 CDB679      1049        CALL   PRTBYT
769B D1          1050        POP    DE
769C C9          1051        RET
769D CDA076      1052 TWO    CALL   ONE             ; PRINT 2 SPACES
76A0 C5          1053 ONE    PUSH   BC              ; PRINT SPACE
76A1 0E20        1054        LD     C,' '
76A3 CD287A      1055        CALL   PRNC
76A6 C1          1056        POP    BC
76A7 C9          1057        RET
                 1058 ;
                 1059 ; ELEMENTARY DOS BEGINS HERE.  FILES ARE MAINTAINED ON
                 1060 ; DRIVE B ONLY AND ARE NOT CP/M COMPATIBLE.  FUNCTIONS
                 1061 ; ARE DIRECTORY LIST (D), SAVE (S), LOAD (L), KILL (K)
                 1062 ; AND INITIALIZE (I).
                 1063 ; FILE NAMES CONSIST OF A PRIMARY FILE NAME OF UP TO
                 1064 ; SIX CHARS AND AN OPTIONAL FILE TYPE OF UP TO THREE.
                 1065 ;
                 1066 ; LIST DIRECTORY
                 1067 ;
76A8 CD3A79      1068 PDIR   CALL   DIR1            ; SET UP FOR DIR SCH
76AB CD8A78      1069 PAGLP  CALL   NEWPG           ; NEXT DIR PG TO BUFF
76AE 0608        1070        LD     B,8             ; 8 ENTRIES/PAGE
76B0 DD7E00      1071 PRPAG  LD     A,(IX+0)        ; STATUS BYTE
76B3 3D          1072        DEC    A
76B4 C8          1073        RET    Z               ; QUIT IF DONE
76B5 3D          1074        DEC    A
76B6 202A        1075        JR     NZ,NEXT-$
76B8 CD4F79      1076        CALL   CRLF
76BB C5          1077 PRLIN  PUSH   BC              ; PRINT NAME FROM DIR
76BC DDE5        1078        PUSH   IX
76BE 060A        1079        LD     B,10
76C0 DD4E01      1080 PLNLP  LD     C,(IX+1)
76C3 CD287A      1081        CALL   PRNC
76C6 DD23        1082        INC    IX
76C8 10F6        1083        DJNZ   PLNLP-$
76CA DDE1        1084        POP    IX
76CC CD0A7A      1085        CALL   SPAC
76CF DD5E0B      1086        LD     E,(IX+11)       ; PRINT ADDRESS
76D2 DD560C      1087        LD     D,(IX+12)
76D5 CDB279      1088        CALL   PRTWRD
76D8 CD0A7A      1089        CALL   SPAC
76DB DD5E0F      1090        LD     E,(IX+15)       ; PRINT # RECORDS
76DE CDB679      1091        CALL   PRTBYT
76E1 C1          1092        POP    BC
76E2 111000      1093 NEXT   LD     DE,10H
76E5 DD19        1094        ADD    IX,DE           ; POINT TO NXT ENTRY
76E7 10C7        1095        DJNZ   PRPAG-$         ; AND REPEAT
76E9 18C0        1096        JR     PAGLP-$
                 1097 ;
                 1098 ; LOAD PGM FROM DRIVE B AND RUN FROM START ADDRESS
                 1099 ;
76EB CDF576      1100 LDGO   CALL   LOAD            ; GET PGM
76EE DD215D7A    1101        LD     IX,VREG         ; FOR SIMULATOR
76F2 C3C16F      1102        JP     RUN
                 1103 ;
                 1104 ; SEARCH FOR AND LOAD A FILE AT ADDR PER DIRECTORY
                 1105 ;
76F5 CDF378      1106 LOAD   CALL   ANAME           ; LOAD A FILE
76F8 CD3A78      1107        CALL   NAMSCH          ; TRY TO MATCH NAME
76FB CA9D78      1108        JP     Z,NTFND
76FE FD6EFB      1109        LD     L,(IY-5)        ; SAVE START ADDR
7701 FD66FC      1110        LD     H,(IY-4)
7704 22647A      1111        LD     (NPC),HL
7707 CDAA78      1112 LOADLP CALL   SETDAT          ; SET UP FOR LOAD
770A CD407A      1113        CALL   RDONE           ; AND READ N SECTORS
770D CDC378      1114        CALL   INCDAT
7710 20F5        1115        JR     NZ,LOADLP-$
7712 C9          1116        RET
                 1117 ;
                 1118 ; SAVE A FILE AND ADD IT TO DIRECTORY
                 1119 ;
7713 CDF378      1120 SAVE   CALL   ANAME           ; SAVE # SECTORS
7716 CD3A78      1121        CALL   NAMSCH          ; SEE IF HAVE ALREADY
7719 280A        1122        JR     Z,SAV1-$
771B CD5479      1123        CALL   LINF
771E 11BF7A      1124        LD     DE,HAVMSG       ; IF SO, INFORM & QUIT
7721 CDCE79      1125        CALL   PRTDE
7724 C9          1126        RET
7725 FD215A7B    1127 SAV1   LD     IY,TEMPS
7729 3E02        1128        LD     A,2             ; NEW ACTIVE FILE
772B FD7700      1129        LD     (IY+0),A
772E CD5479      1130        CALL   LINF
7731 11DE7A      1131        LD     DE,GETADR       ; GET RUN ADDR
7734 CDCE79      1132        CALL   PRTDE
7737 CD6379      1133        CALL   GETWRD
773A FD730B      1134        LD     (IY+11),E
773D FD720C      1135        LD     (IY+12),D
7740 CD5479      1136        CALL   LINF
7743 11F67A      1137        LD     DE,GETN         ; GET # BYTES
7746 CDCE79      1138        CALL   PRTDE
7749 CD6379      1139        CALL   GETWRD
774C AF          1140        XOR    A               ; CONVERT TO SECTORS
774D 0607        1141        LD     B,7             ; DIV BY 128
774F CB3A        1142 SAVX   SRL    D
7751 CB1B        1143        RR     E
7753 3001        1144        JR     NC,SAVY-$
7755 3C          1145        INC    A
7756 10F7        1146 SAVY   DJNZ   SAVX-$
7758 B7          1147        OR     A               ; FRACTIONAL SCTR?
7759 2801        1148        JR     Z,SAVZ-$
775B 1C          1149        INC    E               ; THEN ADD ONE
775C FD730F      1150 SAVZ   LD     (IY+15),E       ; SAVE IT
775F CD3A79      1151        CALL   DIR1            ; SET UP FOR DIR SCH
7762 CD8A78      1152 SAV2   CALL   NEWPG           ; SEARCH FOR NAME
7765 0608        1153        LD     B,8
7767 DD7E00      1154 SAV3   LD     A,(IX+0)        ; STAT BYTE
776A 3D          1155        DEC    A
776B 281F        1156        JR     Z,SAVNEW-$      ; NEW FILE
776D D602        1157        SUB    2               ; DEAD FILE?
776F 2009        1158        JR     NZ,SAV4-$
7771 FD7E0F      1159        LD     A,(IY+15)       ; WILL IT FIT?
7774 3D          1160        DEC    A
7775 DDBE0F      1161        CP     (IX+15)
7778 3821        1162        JR     C,SAVOLD-$
777A 111000      1163 SAV4   LD     DE,10H          ; CHECK NXT ONE
777D DD19        1164        ADD    IX,DE
777F DD56FD      1165        LD     D,(IX-3)        ; OLD TRACK
7782 DD5EFE      1166        LD     E,(IX-2)        ; OLD SECTOR
7785 DD4EFF      1167        LD     C,(IX-1)        ; OLD #
7788 10DD        1168        DJNZ   SAV3-$
778A 18D6        1169        JR     SAV2-$
778C 3E1A        1170 SAVNEW LD     A,26            ; NEW ENTRY
778E 0C          1171        INC    C
778F 1C          1172 NEWLP  INC    E
```

```
7790  BB        1173          CP      E               ; >26?
7791  3003      1174          JR      NC,NEW1-$
7793  14        1175          INC     D               ; IF SO, NXT TRK
7794  1E01      1176          LD      E,1
7796  0D        1177 NEW1     DEC     C
7797  20F6      1178          JR      NZ,NEWLP-$
7799  1806      1179          JR      UPDIR-$
779B  DD560D    1180 SAVOLD   LD      D,(IX+13)
779E  DD5E0E    1181          LD      E,(IX+14)
77A1  FD720D    1182 UPDIR    LD      (IY+13),D       ; PUT NEW TRK & SCTR
77A4  FD730E    1183          LD      (IY+14),E
77A7  CD2878    1184          CALL    PUTDIR          ; UPDATE DIR
77AA  CDAA78    1185 SAVER    CALL    SETDAT          ; SET UP FOR SAVE
77AD  CD437A    1186          CALL    WRONE           ; & WRITE # SECTORS
77B0  CDC378    1187          CALL    INCDAT
77B3  20F5      1188          JR      NZ,SAVER-$
77B5  C9        1189          RET
                1190 ;
                1191 ; KILL A FILE (SET INACTIVE STATUS IN DIRECTORY)
                1192 ;
77B6  CDF378    1193 KILL     CALL    ANAME           ; GET NAME
77B9  CD3A78    1194          CALL    NAMSCH          ; FIND IT
77BC  CA9D78    1195          JP      Z,NTFND
77BF  FD215A7B  1196          LD      IY,TEMPS
77C3  3E03      1197          LD      A,3             ; MAKE STAT = 03
77C5  FD360003  1198          LD      (IY+0),3
77C9  185D      1199          JR      PUTDIR-$        ; AND UPDATE DIRECTORY
                1200 ;
                1201 ; INITIALIZE DISK (WRITES BLANK DIRECTORY)
                1202 ;
77CB  11457B    1203 INIT     LD      DE,INIMSG       ; INITIALIZE DISK
77CE  CDCE79    1204          CALL    PRTDE
77D1  CD167A    1205          CALL    GETCHR          ; ASK YES OR NO
77D4  FE59      1206          CP      'Y'
77D6  C0        1207          RET     NZ
77D7  CD317A    1208          CALL    HOMB
77DA  DD216A7B  1209          LD      IX,BUFFR        ; FILL BUFF W/PATTERN
77DE  FD21EA7B  1210          LD      IY,INIMSK       ; STORED AT INIMSK
77E2  DDE5      1211          PUSH    IX
77E4  0E08      1212          LD      C,8
77E6  FDE5      1213 INLP1    PUSH    IY
77E8  0610      1214          LD      B,10H
77EA  FD7E00    1215 INLP2    LD      A,(IY+0)
77ED  DD7700    1216          LD      (IX+0),A
77F0  DD23      1217          INC     IX
77F2  FD23      1218          INC     IY
77F4  10F4      1219          DJNZ    INLP2-$
77F6  FDE1      1220          POP     IY
77F8  0D        1221          DEC     C
77F9  20EB      1222          JR      NZ,INLP1-$
77FB  DDE1      1223          POP     IX
77FD  3E03      1224          LD      A,3             ; MAKE 1ST ONE DEAD
77FF  DD7700    1225          LD      (IX+0),A
7802  016A7B    1226          LD      BC,BUFFR
7805  CD3D7A    1227          CALL    BUFS
7808  0E00      1228          LD      C,0
780A  CD3A7A    1229          CALL    TRKS
780D  0C        1230          INC     C
780E  CD377A    1231          CALL    SECS
7811  CD437A    1232          CALL    WRONE           ; 1ST DIR PAGE
7814  3E01      1233          LD      A,1             ; RESTORE PATTERN
7816  DD7700    1234          LD      (IX+0),A
7819  010219    1235          LD      BC,1902H

781C  C5        1236 WRLP     PUSH    BC              ; WRITE REST OF TRK 0
781D  CD377A    1237          CALL    SECS
7820  CD437A    1238          CALL    WRONE
7823  C1        1239          POP     BC
7824  0C        1240          INC     C
7825  10F5      1241          DJNZ    WRLP-$
7827  C9        1242          RET
                1243 ;
                1244 ; OVERWRITE CURRENT DIRECTORY PAGE
                1245 ;
7828  0610      1246 PUTDIR   LD      B,10H           ; UPDATE DIR
782A  FD7E00    1247 UPLP     LD      A,(IY+0)
782D  DD7700    1248          LD      (IX+0),A
7830  DD23      1249          INC     IX
7832  FD23      1250          INC     IY
7834  10F4      1251          DJNZ    UPLP-$
7836  CD437A    1252          CALL    WRONE           ; REWRITE DIR PAGE
7839  C9        1253          RET
                1254 ;
                1255 ; SEARCH FOR FILE NAME IN DIRECTORY.. RETURN WITH Z FLG
                1256 ; SET IF NOT FOUND, Z CLEAR AND BUFFER FILLED IF FOUND
                1257 ;
783A  CD3A79    1258 NAMSCH   CALL    DIR1            ; SET UP FOR DIR SCH
783D  CD8A78    1259 LDLP     CALL    NEWPG           ; READ PAGE OF NDX
7840  0608      1260          LD      B,8
7842  DD7E00    1261 LDLP1    LD      A,(IX+0)        ; CHECK STAT
7845  3D        1262          DEC     A
7846  2001      1263          JR      NZ,LDLP2-$
7848  C9        1264          RET
7849  3D        1265 LDLP2    DEC     A
784A  DDE5      1266          PUSH    IX
784C  FDE5      1267          PUSH    IY
784E  2013      1268          JR      NZ,LDNXT-$      ; INACTIVE?
7850  0E0A      1269          LD      C,10            ; ELSE COMP NAME
7852  FD7E01    1270 CMPLP    LD      A,(IY+1)
7855  DDBE01    1271          CP      (IX+1)
7858  2009      1272          JR      NZ,LDNXT-$
785A  DD23      1273          INC     IX
785C  FD23      1274          INC     IY
785E  0D        1275          DEC     C
785F  20F1      1276          JR      NZ,CMPLP-$
7861  180D      1277          JR      CMPOK-$         ; SAME NAME
7863  FDE1      1278 LDNXT    POP     IY              ; TRY NEXT ONE
7865  DDE1      1279          POP     IX
7867  111000    1280          LD      DE,10H
786A  DD19      1281          ADD     IX,DE
786C  10D4      1282          DJNZ    LDLP1-$
786E  18CD      1283          JR      LDLP-$          ; NEXT PAGE
7870  FDE1      1284 CMPOK    POP     IY              ; NAME MATCHES
7872  DDE1      1285          POP     IX
7874  0610      1286          LD      B,10H           ; XFER DIR NTRY TO BUFF
7876  DD7E00    1287 FILLP    LD      A,(IX+0)
7879  FD7700    1288          LD      (IY+0),A
787C  DD23      1289          INC     IX
787E  FD23      1290          INC     IY
7880  10F4      1291          DJNZ    FILLP-$
7882  11F0FF    1292          LD      DE,-10H
7885  DD19      1293          ADD     IX,DE
7887  AF        1294          XOR     A
7888  3C        1295          INC     A               ; EXIT A = 1
7889  C9        1296          RET
788A  D9        1297 NEWPG    EXX                     ; GET NXT DIR PAGE
788B  DD216A7B  1298          LD      IX,BUFFR
```

```
788F FD340E  1299          INC     (IY+14)          ; NXT SCTR
7892 FD4E0E  1300          LD      C,(IY+14)
7895 CD377A  1301          CALL    SECS
7898 CD407A  1302          CALL    RDONE            ; READ IT IN
789B D9      1303          EXX
789C C9      1304          RET
789D CD5479  1305 NTFND    CALL    LINF             ; NOT FOUND
78A0 11B07A  1306          LD      DE,NFMSG
78A3 CDCE79  1307          CALL    PRTDE
78A6 31FE7D  1308          LD      SP, RES.SYSTK-2  ; RET TO CMND
78A9 C9      1309          RET
             1310 ;
             1311 ; FILL IN DISK DATA. USES SR CALLS IN CBIOS
             1312 ;
78AA FD215A7B 1313 SETDAT  LD      IY,TEMPS         ; SET UP FOR DSK DATA
78AE FD4E0B  1314          LD      C,(IY+11)
78B1 FD460C  1315          LD      B,(IY+12)
78B4 CD3D7A  1316          CALL    BUFS             ; XFER ADDR
78B7 FD4E0D  1317          LD      C,(IY+13)
78BA CD3A7A  1318          CALL    TRKS             ; TRACK
78BD FD4E0E  1319          LD      C,(IY+14)
78C0 C3377A  1320          JP      SECS             ; SECTOR
             1321 ;
             1322 ; BUMP SECTOR/TRACK AND XFER ADDRESS FOR MULTI
             1323 ; SECTOR READ OR WRITE
             1324 ;
78C3 FD7E0B  1325 INCDAT   LD      A,(IY+11)        ; BUMP DSK DATA
78C6 C680    1326          ADD     A,80H            ; BUMP XFER ADDR
78C8 FD770B  1327          LD      (IY+11),A
78CB 3003    1328          JR      NC,INC1-$
78CD FD340C  1329          INC     (IY+12)
78D0 FD7E0E  1330 INC1     LD      A,(IY+14)
78D3 3C      1331          INC     A                ; BUMP SECTOR
78D4 FE1B    1332          CP      27
78D6 3805    1333          JR      C,INC2-$
78D8 3E01    1334          LD      A,1
78DA FD340D  1335          INC     (IY+13)          ; AND TRK IF NEEDED
78DD FD770E  1336 INC2     LD      (IY+14),A
78E0 FD7E0D  1337          LD      A,(IY+13)        ; CHECK FOR VALID TRACK
78E3 FE4D    1338          CP      77
78E5 3808    1339          JR      C,INC3-$
78E7 11CB7A  1340          LD      DE,DERMSG        ; ERROR
78EA CDD979  1341          CALL    PRTXT
78ED C1      1342          POP     BC               ; RET TO CMND
78EE C9      1343          RET
78EF FD350F  1344 INC3     DEC     (IY+15)
78F2 C9      1345          RET
             1346 ;
             1347 ; GET FILE NAME TO BUFFER
             1348 ;
78F3 FD215A7B 1349 ANAME   LD      IY,TEMPS
78F7 FDE5    1350          PUSH    IY
78F9 060A    1351          LD      B,10
78FB 3E20    1352          LD      A,' '
78FD FD7701  1353 MPTY     LD      (IY+1),A         ; CLEAR INPUT BUFFER
7900 FD23    1354          INC     IY
7902 10F9    1355          DJNZ    MPTY-$
7904 110A7B  1356          LD      DE,NAMSG         ; PROMPT FOR NAME
7907 CDCE79  1357          CALL    PRTDE
790A FDE1    1358          POP     IY
790C FDE5    1359          PUSH    IY
790E CD167A  1360 NAMLP    CALL    GETCHR
7911 FE0D    1361          CP      ODH              ; END?
7913 2822    1362          JR      Z,NAMIN-$
7915 FE2E    1363          CP      '.'              ; DELIMITER?
7917 2807    1364          JR      Z,FTYP-$
7919 FD7701  1365          LD      (IY+1),A         ; PUT IT IN BUFFER
791C FD23    1366          INC     IY
791E 18EE    1367          JR      NAMLP-$
7920 FDE1    1368 FTYP     POP     IY
7922 FDE5    1369          PUSH    IY
7924 FD7707  1370          LD      (IY+7),A
7927 FD23    1371          INC     IY
7929 CD167A  1372 TYPLP    CALL    GETCHR
792C FE0D    1373          CP      ODH
792E 2807    1374          JR      Z,NAMIN-$
7930 FD7707  1375          LD      (IY+7),A
7933 FD23    1376          INC     IY
7935 18F2    1377          JR      TYPLP-$
7937 FDE1    1378 NAMIN    POP     IY
7939 C9      1379          RET
             1380 ;
             1381 ; SET UP FOR DIRECTORY SEARCH
             1382 ;
793A FD215A7B 1383 DIR1    LD      IY,TEMPS         ; SET UP FOR DIR SCH
793E 016A7B  1384          LD      BC,BUFFR
7941 CD3D7A  1385          CALL    BUFS
7944 0E00    1386          LD      C,0              ; TRACK 0
7946 FD710D  1387          LD      (IY+13),C
7949 FD710E  1388          LD      (IY+14),C
794C C33A7A  1389          JP      TRKS
             1390 ;
             1391 ; VARIOUS ROUTINES USED BY TRANSLATOR, MONITOR, AND DOS
             1392 ;
794F 0E0D    1393 CRLF     LD      C,ODH            ; PRINT CRLF
7951 CD287A  1394          CALL    PRNC
7954 0E0A    1395 LINF     LD      C,0AH
7956 C3287A  1396          JP      PRNC
7959 0E20    1397 PROM     LD      C,' '            ; PRINT ' >'
795B CD287A  1398          CALL    PRNC
795E 0E3E    1399          LD      C,'>'
7960 C3287A  1400          JP      PRNC
             1401 ;
             1402 ; GET HEX DATA IN DE. ROTATE CONTINUOUSLY UNTIL A
             1403 ; 'CR' IS INPUT. LSB IN E. SET Z FLG AND RET ON 'SPACE'
             1404 ;
7963 CD167A  1405 GETWRD   CALL    GETCHR           ; GET BINARY DATA
7966 FE20    1406          CP      ' '              ; RET Z IF SPACE
7968 C8      1407          RET     Z
7969 EB      1408          EX      DE,HL
796A 210000  1409          LD      HL,0             ; CLEAR HOLDING REG
796D CD8979  1410          CALL    CONV             ; CHANGE TO HEX
7970 28F1    1411          JR      Z,GETWRD-$       ; INVALID CHAR
7972 CDAB79  1412          CALL    ROBYT            ; ROTATE INTO HL
7975 CD167A  1413 WRDLP    CALL    GETCHR           ; GET NEXT CHAR
7978 FE0D    1414          CP      ODH              ; TERMINATOR
797A 280A    1415          JR      Z,WRDON-$
797C CD8979  1416          CALL    CONV
797F 28F4    1417          JR      Z,WRDLP-$
7981 CDAB79  1418          CALL    ROBYT
7984 18EF    1419          JR      WRDLP-$
7986 B7      1420 WRDON    OR      A                ; CLEAR Z FLAG
7987 EB      1421          EX      DE,HL            ; DATA INTO DE
7988 C9      1422          RET
             1423 ;
             1424 ; CONVERT ASCII DATA IN A TO HEX. SET Z FLG IF INVALID
```

```
                        1425 ;
798C  FE30      1426 CONV   CP      '0'             ; CONVERT TO BINARY
798E  380C      1427        JR      C,BAD-$
7990  FE3A      1428        CP      3AH
7992  380F      1429        JR      C,GOOD-$
7994  FE41      1430        CP      41H
7996  3804      1431        JR      C,BAD-$
7998  FE47      1432        CP      47H
799A  3807      1433        JR      C,GOOD-$
799C  0E3F      1434 BAD    LD      C,'?'
799E  CD287A    1435        CALL    PRNC            ; NOTIFY BAD INPUT
79A1  AF        1436        XOR     A               ; SET Z FLAG
79A2  C9        1437        RET
79A3  D630      1438 GOOD   SUB     30H
79A5  FE0A      1439        CP      10
79A7  3802      1440        JR      C,GOOD1-$
79A9  D607      1441        SUB     7
79AB  FEFF      1442 GOOD1  CP      0FFH            ; CLEAR Z FLAG
79AD  C9        1443        RET
                        1444 ;
                        1445 ; ROTATE BYTE IN A INTO HL VIA L
                        1446 ;
79AE  29        1447 ROBYT  ADD     HL,HL           ; ROTATE A INTO HL
79AF  29        1448        ADD     HL,HL           ; ROTATE HL 4 BITS LEFT
79B0  29        1449        ADD     HL,HL
79B1  29        1450        ADD     HL,HL
79B2  B5        1451        OR      L               ; PUT NEW DIGIT INTO L
79B3  6F        1452        LD      L,A
79B4  C9        1453        RET
                        1454 ;
                        1455 ; PRINT FOUR DIGIT VALUE IN DE
                        1456 ;
79B5  7A        1457 PRTWRD LD      A,D
79B6  CDB779    1458        CALL    PRBIN
79B9  7B        1459 PRTBYT LD      A,E             ; PRINT VALUE IN E
79BA  F5        1460 PRBIN  PUSH    AF
79BB  CB0F      1461        RRC     A
79BD  CB0F      1462        RRC     A
79BF  CB0F      1463        RRC     A
79C1  CB0F      1464        RRC     A
79C3  CDC479    1465        CALL    PRB1
79C6  F1        1466        POP     AF
79C7  E60F      1467 PRB1   AND     0FH
79C9  C690      1468        ADD     A,90H
79CB  27        1469        DAA
79CC  CE40      1470        ADC     A,40H
79CE  27        1471        DAA
79CF  1859      1472        JR      PRNA-$
                        1473 ;
                        1474 ; PRINT STRING POINTED TO BY DE.  USES PRTXT
                        1475 ;
79D1  FDE5      1476 PRTDE  PUSH    IY              ; PRNT PER DE
79D3  D5        1477        PUSH    DE
79D4  FDE1      1478        POP     IY
79D6  CDD979    1479        CALL    PRTXT
79D9  FDE1      1480        POP     IY
79DB  C9        1481        RET
                        1482 ;
                        1483 ; PRINT ASCII STRING POINTED TO BY IY.  CHECK FOR
                        1484 ; SPECIALS:  '1' PRINTS AS BYTE POINTED TO BY HL, '2'
                        1485 ; PRINTS (HL+1), '!' DOES NOT PRINT, '^' TERMINATES.
                        1486 ;
79DC  1E00      1487 PRTXT  LD      E,0             ; PRNT UP TO 255 CHRSY

79DE  FD7E00    1488 PRNLP  LD      A,(IY+0)
79E1  FE5E      1489        CP      '^'             ; TERMINATOR?
79E3  C8        1490        RET     Z
79E4  FE21      1491        CP      '!'             ; NO PRINT?
79E6  281F      1492        JR      Z,DONXT+1-$
79E8  D5        1493        PUSH    DE
79E9  2A627A    1494        LD      HL,(CPC)
79EC  23        1495        INC     HL
79ED  FE32      1496        CP      '2'             ; PRINT (HL+1)?
79EF  2007      1497        JR      NZ,PRN1-$
79F1  23        1498        INC     HL
79F2  5E        1499        LD      E,(HL)
79F3  CDB679    1500        CALL    PRTBYT
79F6  180E      1501        JR      DONXT-$
79F8  FE31      1502 PRN1   CP      '1'             ; PRINT (HL)?
79FA  2006      1503        JR      NZ,DOALPH-$
79FC  5E        1504        LD      E,(HL)
79FD  CDB679    1505        CALL    PRTBYT
7A00  1804      1506        JR      DONXT-$
7A02  4F        1507 DOALPH LD      C,A             ; ALL OTHER CHARACTERS
7A03  CD287A    1508        CALL    PRNC
7A06  D1        1509 DONXT  POP     DE
7A07  FD23      1510        INC     IY
7A09  1D        1511        DEC     E
7A0A  20D2      1512        JR      NZ,PRNLP-$
7A0C  C9        1513        RET
                        1514 ;
7A0D  C5        1515 SPAC   PUSH    BC              ; PRINT 6 SPACES
7A0E  0606      1516        LD      B,6
7A10  0E20      1517        LD      C,' '
7A12  CD287A    1518 SPLP   CALL    PRNC
7A15  10FB      1519        DJNZ    SPLP-$
7A17  C1        1520        POP     BC
7A18  C9        1521        RET
                        1522 ;
                        1523 ; CONSOLE I/O
                        1524 ;
7A19  CD2B7A    1525 GETCHR CALL    KBIN            ; GET CHAR & TEST
7A1C  FE03      1526        CP      3               ; ^C...
7A1E  CA037E    1527        JP      Z,CPWRM         ; CP/M WARM START
7A21  FE18      1528        CP      18H             ; ^X (CAN)...
7A23  2005      1529        JR      NZ,PRNA-$
7A25  31FE7D    1530        LD      SP,.RES.SYSTK-2 ; FOR RET TO CMND
7A28  3E23      1531        LD      A,'#'
7A2A  4F        1532 PRNA   LD      C,A             ; PRINT CHAR IN A
                        1533 ;
                        1534 ; JUMPS TO EXTERNAL I/O...MODIFY AS NEEDED
                        1535 ;
7A2B  C30C7E    1536 PRNC   JP      CONOUT          ; PRINT CHAR IN C
7A2E  C3097E    1537 KBIN   JP      CONIN           ; GET SINGLE CHAR IN A
7A31  C3067E    1538 KBST   JP      CONST           ; TEST INPUT STATUS
7A34  C3187E    1539 HOMB   JP      HOME            ; DRIVE HOME
7A37  C31B7E    1540 DRVS   JP      SETDRV          ; SET DRIVE #
7A3A  C3217E    1541 SECS   JP      SETSEC          ; SET SECTOR #
7A3D  C31E7E    1542 TRKS   JP      SETTRK          ; SET TRK #
7A40  C3247E    1543 BUFS   JP      SETBUF          ; SET XFER ADDRESS
7A43  C3277E    1544 RDONE  JP      READ            ; READ A SECTOR
7A46  C32A7E    1545 WRONE  JP      WRITE           ; WRITE A SECTOR
                        1546 ;
                        1547 ; JUMP TABLE FOR LINKS TO Z80 ROUTINES...FILL IN AS REQD
                        1548 ; FIRST TWO INITIALIZED TO CONSOLE I/O VIA ACC
                        1549 ;
7A49  C32B7A    1550 USR0   JP      KBIN            ; 6502 USER ROUTINES
```

```
7A49  C3277A    1551 USR1    JP      PRNA
7A4C  C90000    1552 USR2    DEFB    0C9H,0,0
7A4F  C90000    1553 USR3    DEFB    0C9H,0,0
7A52  C90000    1554 USR4    DEFB    0C9H,0,0
                1555 ;
                1556 ; 6502 BREAK VECTOR...INIT TO GO TO COMMAND LEVEL
                1557 ;
7A55  0000      1558 IPOINT  DEFW    0
                1559 ;
                1560 ; TEMPORARIES FOR TRANSLATOR
                1561 ;
7A57  01        1562 NSTEPS  DEFB    1              ; # STEPS TO EXECUTE
7A58  00        1563 NBYTES  DEFB    0              ; # BYTES IN INSTR
7A59  000A      1564 DLYCON  DEFW    0A00H          ; TIME DELAY FOR # STEP
7A5B  0000      1565 BREAK   DEFW    0              ; BREAKPOINT ADDRESS
                1566 ;
                1567 ; SIMULATED 6502 CPU REGISTERS...INIT TO THESE VALUES
                1568 ;
7A5D  00        1569 VREG    DEFB    0              ; ACCUMULATOR
7A5E  00        1570         DEFB    0              ; STATUS (P) REG
7A5F  00        1571         DEFB    0              ; X INDEX REG
7A60  00        1572         DEFB    0              ; Y INDEX REG
7A61  FF        1573         DEFB    0FFH           ; STACK POINTER
7A62  0002      1574 CPC     DEFW    200H           ; PROGRAM COUNTER
7A64  0002      1575 NPC     DEFW    200H           ; LOCATION OF NXT INSTR
                1576 ;
                1577 ; VARIOUS MESSAGES FOR SYSTEM
                1578 ;
7A66  4152454D  1579 INTRO   DEFM    'AREMKAY SIM/65 TRANSLATOR^^
      4B415920
      53494D2F
      36352054
      52414E53
      4C41544F
      525E
7A80  20435043  1580 HEADR   DEFM    ' CPC.. INS.. A.... P....'
      2E2E2049
      4E532E2E
      20412E2E
      2E2E2050
      2E2E2E2E
7A98  20582E2E  1581         DEFM    ' X.... Y.... SP... NPC..'
      2E2E2059
      2E2E2E2E
      2053502E
      2E2E204E
      50432E2E
7AB0  46494C45  1582 NFMSG   DEFM    'FILE NOT FOUND^^
      204E4F54
      20464F55
      4E445E
7ABF  46494C45  1583 HAVMSG  DEFM    'FILE EXISTS^^
      20455849
      5354535E
7ACB  42414420  1584 DERMSG  DEFM    'BAD TRACK/SECTOR #^^
      54524143
      4B2F5345
      43544F52
      20235E
7ADE  2020454E  1585 GETADR  DEFM    '  ENTER START ADDRESS: ^^
      54455220
      53544152
      54204144
      44524553
      533A205E
7AF6  2020454E  1586 GETN    DEFM    '  ENTER NO. BYTES: ^^
      54455220
      4E4F2E20
      42595445
      533A205E
7B0A  2020454E  1587 NAMSG   DEFM    '  ENTER FILNAM.TYP: ^^
      54455220
      46494C4E
      414D2E54
      59503A20
      5E
7B1F  2020454E  1588 MLTMSG  DEFM    '  ENTER # STEPS: ^^
      54455220
      23205354
      4550533A
      205E
7B31  2020454E  1589 BKPMSG  DEFM    '  ENTER BRKP ADDR: ^^
      54455220
      42524B50
      20414444
      523A205E
7B45  2020494E  1590 INIMSG  DEFM    '  INITIALIZE? (Y/N) ^^
      49544941
      4C495A45
      3F202859
      2F4E2920
      5E
                1591 ;
                1592 ; DOS BUFFERS AND DATA
                1593 ;
>7B5A           1594 TEMPS   DEFS    10H            ; DISK ENTRY BUFFER
>7B6A           1595 BUFFR   DEFS    80H            ; DIRECTORY PAGE BUFFER
7BEA  01202020  1596 INIMSK  DEFB    1,20H,20H,20H,20H,20H,20H,20H
      20202020
7BF2  20202020  1597         DEFB    20H,20H,20H,20H,20H,1,0,0
      20010000
                1598 ;
                1599 ; DATA TABLES FOR TRANSLATOR BEGIN HERE
                1600 ;
                1601 ; ADDRESSING MODE TABLES
                1602 ;
7BFA  0C1A      1603 AAATBL  DEFW    1A0CH          ; (ZPG,X)
7BFC  040E      1604         DEFW    0E04H          ; ZPG
7BFE  0006      1605         DEFW    0600H          ; IMM
7C00  060B      1606         DEFW    0B06H          ; ABS
7C02  0E1E      1607         DEFW    1E0EH          ; (ZPG),Y
7C04  0822      1608         DEFW    2208H          ; ZPG,X
7C06  122F      1609         DEFW    2F12H          ; ABS,Y
7C08  102B      1610         DEFW    2B10H          ; ABS,X
7C0A  0000      1611 BBBTBL  DEFW    0              ; INVALID
7C0C  040E      1612         DEFW    0E04H          ; ZPG
7C0E  0215      1613         DEFW    1502H          ; ACC
7C10  060B      1614         DEFW    0B06H          ; ABS
7C12  0A26      1615         DEFW    260AH          ; ZPG,Y
7C14  0822      1616         DEFW    2208H          ; ZPG,X
7C16  122F      1617         DEFW    2F12H          ; ABS,Y
7C18  102B      1618         DEFW    2B10H          ; ABS,X
7C1A  0006      1619 DDDTBL  DEFW    0600H          ; IMM
7C1C  040E      1620         DEFW    0E04H          ; ZPG
7C1E  0000      1621         DEFW    0              ; INVALID
7C20  060B      1622         DEFW    0B06H          ; ABS
```

(Continued on pg. 40)

(Continued from pg. 38)

```
7C22  0A26      1623          DEFW    260AH           ; ZPG,Y
7C24  0822      1624          DEFW    2208H           ; ZPG,X
7C26  122F      1625          DEFW    2F12H           ; ABS,Y
7C28  102B      1626          DEFW    2B10H           ; ABS,X
7C2A  040E      1627 BBXTBL   DEFW    0E04H           ; ZPG
7C2C  060B      1628          DEFW    0B06H           ; ABS
7C2E  0822      1629          DEFW    2208H           ; ZPG,X
7C30  102B      1630          DEFW    2B10H           ; ABS,X
7C32  0A26      1631          DEFW    260AH           ; ZPG,Y
7C34  0006      1632 CCTBL    DEFW    0600H           ; IMM
7C36  040E      1633          DEFW    0E04H           ; ZPG
7C38  0000      1634          DEFW    0               ; INVALID
7C3A  060B      1635          DEFW    0B06H           ; ABS
                1636 ;
                1637 ; BRANCH MASK TABLE
                1638 ;
7C3C  2680      1639 MSKTBL   DEFW    8026H           ; BPL
7C3E  1E80      1640          DEFW    801EH           ; BMI
7C40  2E40      1641          DEFW    402EH           ; BVC
7C42  3240      1642          DEFW    4032H           ; BVS
7C44  0E01      1643          DEFW    010EH           ; BCC
7C46  1201      1644          DEFW    0112H           ; BCS
7C48  2202      1645          DEFW    0222H           ; BNE
7C4A  1602      1646          DEFW    0216H           ; BEQ
                1647 ;
                1648 ; FLAG SET/RESET CODES
                1649 ;
7C4C  3501      1650 FLGTBL   DEFW    0135H           ; CLC
7C4E  B101      1651          DEFW    01B1H           ; SEC
7C50  3D04      1652          DEFW    043DH           ; CLI
7C52  B904      1653          DEFW    04B9H           ; SEI
7C54  4140      1654          DEFW    4041H           ; CLV
7C56  0000      1655          DEFW    0000            ; INVALID
7C58  3908      1656          DEFW    0839H           ; CLD
7C5A  B508      1657          DEFW    08B5H           ; SED
                1658 ;
                1659 ; JUMP TABLES TO IMPLIED INSTRUCTIONS
                1660 ;
7C5C  7E72      1661 IMPTBL   DEFW    PPHP            ; IMPLIED INSTR TABLE
7C5E  A572      1662          DEFW    PPLP
7C60  7472      1663          DEFW    PPHA
7C62  9372      1664          DEFW    PPLA
7C64  5172      1665          DEFW    PDEY
7C66  C772      1666          DEFW    PTAY
7C68  4372      1667          DEFW    PINY
7C6A  3C72      1668          DEFW    PINX
7C6C  D172      1669          DEFW    PTXA
7C6E  BD72      1670          DEFW    PTAX
7C70  4A72      1671          DEFW    PDEX
7C72  6E72      1672          DEFW    PNOP
                1673 ;
                1674 ; JUMP TABLE TO ADDR MODE PROCESSORS
                1675 ;
7C74  6A75      1676 GETTBL   DEFW    PIMM            ; INSTR MODE TABLE
7C76  6E75      1677          DEFW    PACX
7C78  7675      1678          DEFW    PZPG
7C7A  7C75      1679          DEFW    PAB
7C7C  8275      1680          DEFW    PZX
7C7E  8775      1681          DEFW    PZY
7C80  9175      1682          DEFW    PZIX
7C82  A775      1683          DEFW    PZIY
7C84  C075      1684          DEFW    PABX
7C86  5D75      1685          DEFW    PABY
                1686 ;
                1687 ; JUMP TABLES TO MULTI-MODE PROCESSORS
                1688 ;
7C88  3873      1689 EVNTBL   DEFW    PASL            ; JUMPS TO PROCESSORS
7C8A  4673      1690          DEFW    PBTRL
7C8C  7673      1691          DEFW    PLSR
7C8E  8473      1692          DEFW    PROR
7C90  9473      1693          DEFW    PSTXY
7C92  B073      1694          DEFW    PLDXY
7C94  CC73      1695          DEFW    PDCY
7C96  DD73      1696          DEFW    PICX
7C98  4274      1697 ODDTBL   DEFW    PORA
7C9A  5574      1698          DEFW    PAND
7C9C  6874      1699          DEFW    PEOR
7C9E  7B74      1700          DEFW    PADC
7CA0  AB74      1701          DEFW    PSTA
7CA2  B774      1702          DEFW    PLDA
7CA4  C374      1703          DEFW    PCMP
7CA6  1074      1704          DEFW    PSBC
                1705 ;
                1706 ; MNEMONIC ASCII CODES (PACKED INTO 2 BYTES)
                1707 ;
7CA8  8304      1708 MNMTBL   DEFW    0483H           ; MNEMONIC TABLE
7CAA  C405      1709          DEFW    05C4H
7CAC  6C06      1710          DEFW    066CH
7CAE  6308      1711          DEFW    0863H           ; BCC
7CB0  7308      1712          DEFW    0873H
7CB2  B108      1713          DEFW    08B1H
7CB4  3409      1714          DEFW    0934H
7CB6  A909      1715          DEFW    09A9H
7CB8  C509      1716          DEFW    09C5H
7CBA  0C0A      1717          DEFW    0A0CH
7CBC  4B0A      1718          DEFW    0A4BH
7CBE  C30A      1719          DEFW    0AC3H
7CC0  D30A      1720          DEFW    0AD3H
7CC2  830D      1721          DEFW    0D83H           ; CLC
7CC4  840D      1722          DEFW    0D84H
7CC6  890D      1723          DEFW    0D89H
7CC8  960D      1724          DEFW    0D96H
7CCA  B00D      1725          DEFW    0DB0H
7CCC  180E      1726          DEFW    0E18H
7CCE  190E      1727          DEFW    0E19H
7CD0  A310      1728          DEFW    10A3H           ; DEC
7CD2  B810      1729          DEFW    10B8H
7CD4  B910      1730          DEFW    10B9H
7CD6  F215      1731          DEFW    15F2H           ; EOR
7CD8  C325      1732          DEFW    25C3H           ; INC
7CDA  D825      1733          DEFW    25D8H
7CDC  D925      1734          DEFW    25D9H
7CDE  B029      1735          DEFW    29B0H           ; JMP
7CE0  722A      1736          DEFW    2A72H
7CE2  8130      1737          DEFW    3081H           ; LDA
7CE4  9830      1738          DEFW    3098H
7CE6  9930      1739          DEFW    3099H
7CE8  7232      1740          DEFW    3272H
7CEA  F039      1741          DEFW    39F0H           ; NOP
7CEC  413E      1742          DEFW    3E41H           ; ORA
7CEE  0141      1743          DEFW    4101H           ; PHA
7CF0  1041      1744          DEFW    4110H
7CF2  8141      1745          DEFW    4181H
7CF4  9041      1746          DEFW    4190H
```

(Continued on pg. 42)

(Continued from pg. 40)

```
7CF6  EC49   1747          DEFW   49ECH          ; ROL
7CF8  F249   1748          DEFW   49F2H
7CFA  894A   1749          DEFW   4A89H
7CFC  934A   1750          DEFW   4A93H
7CFE  434C   1751          DEFW   4C43H          ; SBC
7D00  A34C   1752          DEFW   4CA3H
7D02  A44C   1753          DEFW   4CA4H
7D04  A94C   1754          DEFW   4CA9H
7D06  814E   1755          DEFW   4E81H
7D08  984E   1756          DEFW   4E98H
7D0A  994E   1757          DEFW   4E99H
7D0C  3850   1758          DEFW   5038H          ; TAX
7D0E  3950   1759          DEFW   5039H
7D10  7852   1760          DEFW   5278H
7D12  0153   1761          DEFW   5301H
7D14  1353   1762          DEFW   5313H
7D16  2153   1763          DEFW   5321H
             1764 ;
             1765 ; ADDRESS MODE PRINT TABLE
             1766 ;
7D18  20243120 1767 AMDTBL  DEFM   ' $1   !'     ; ADDR MODE TABLE...REL
      20202021
7D20  20232431 1768         DEFM   ' #$1  !'     ; IMMEDIATE
      20202021
7D28  20243231 1769         DEFM   ' $21  !!'    ; ABS
      20202121
7D30  20243120 1770         DEFM   ' $1   !'     ; ZPG
      20202021
7D38  20202020 1771         DEFM   '      '      ; IMPLIED
      20202020
7D40  20412020 1772         DEFM   ' A    '      ; ACCUM
      20202020
7D48  20282431 1773         DEFM   ' ($1,X)!'    ; (ZPG,X)
      2C582921
7D50  20282431 1774         DEFM   ' ($1),Y!'    ; (ZPG),Y
      292C5921
7D58  20243120 1775         DEFM   ' $1,X  !'    ; ZPG,X
      58202021
7D60  20243120 1776         DEFM   ' $1,Y  !'    ; ZPG,Y
      59202021
7D68  20243231 1777         DEFM   ' $21,X!!'    ; ABS,X
      2C582121
7D70  20243231 1778         DEFM   ' $21,Y!!'    ; ABS,Y
      2C592121
7D78  20282432 1779         DEFM   ' ($21)!!'    ; INDIRECT
      31292121
             1780 ;
             1781 ; THAT'S ALL!
             1782 ;
             1783          END

ERRORS=0000
```

```
                0001 ; LOADER TO ALLOW ZX65 TO EXECUTE UNDER CP/M
                0002 ;
>0020           0003 SIZE    EQU     32              ; # K SYSTEM MEMORY
>8000           0004 TOP     EQU      RES.SIZE*1024
>6F00           0005 BASE    EQU      RES.TOP-1100H  ; BASE OF ZX65
>0F00           0006 LNGTH   EQU     0F00H           ; LNGTH OF SYSTEM
                0007 ;
>0100           0008         ORG     100H            ; FOR CP/M
0100  211001    0009         LD      HL,TEMP
0103  11006F    0010         LD      DE,BASE
0106  01000F    0011         LD      BC,LNGTH
0109  EDB0      0012         LDIR                    ; SYSTEM TO HIGH MEM
010B  C3006F    0013         JP      BASE            ; GO EXECUTE IT THERE
010E  00        0014         NOP
010F  00        0015         NOP
0110  00        0016 TEMP    DEFB    0
                0017 ;
                0018         END

ERRORS=0000
```

"ZXLD" . . . Auxiliary loader required to boot main program from CP/M.

THE END

Letters

RULES NOT *ALWAYS* MADE TO BE BROKEN

Dear Editor:

If any of your readers are thinking about hanging a dot-matrix line printer onto their microcomputer system, I have a friendly warning to pass on to them. "Pay close attention to your manufacturer's recommendations, or know the risk you're taking if you ignore them."

For example, North Star recommends the Anadex DP-8000 connected to the parallel interface of their Horizon. But comparing printer specs, I chose to save a few bucks by building a Heath H14 line printer for the serial interface.

I saved some money at $625.00 plus shipping, plus an additional $82.98 at Heathkit Electronic Center in Seattle when the H14 flunked its initial power-on tests. They replaced two defective CMOS IC's and repaired three open foil breaks in the five volt supply on the PC board at no charge, but they detected erroneous installation of seven transistors. (Considering the obvious textual errors in the documentation, which would you believe – the text, or the pictorials? I guessed wrong and followed the pictorial. Customer error!)

My WH14 printer tested perfectly at 4800 baud under HDOS in Seattle. It went ape at 4800 baud on the Horizon after I got it home. A quick phone call to my friendly Horizon dealer divulged the fact that North Star DOS does not test for handshaking signals! (The Heath manual advises to run no faster than 110 baud without handshaking.)

So now I have a 110 baud line printer dawdling along while the 4MHZ Z-80A and I are twiddling our respective thumbs! Does anybody out there want to trade an in-warranty Anadex DP-8000 printer for an in-warranty Heath WH14 plus some extra cash?

Sincerely,
John R. Dye

4807 Fifteenth Ave. S.E.
Lacey, WA 98503

GROWING PAINS IN THE SOFTWARE MARKET

Dear Suzanne;

On seeing my letter published in the April, 1980 issue, I realize that my expression of my frustration at the marketing methods of the CP/M products could easily be misconstrued as condemning these products or their marketer. This was not my intent and I would like to publically retract the poorly written letter.

I have every reason to believe that the packages offered by Lifeboat Associates are very valuable items. I believe that Lifeboat does a skilled job developing some of these products for wider use.

My problem is that as a relative newcomer to data processing with a great perceived need for business systems not yet available, I have devoted at least half my working efforts for several years in this direction. I find that to implement a new complex package that I need more support than I am able to obtain by telephone during limited daytime hours from New York City. I have scoured the dealers of the Northwestern United States and occasionally elsewhere and can find virtually no one who can demonstrate the CP/M software packages in operation. When I have inquired why this obviously desirable material is not offered, they tell me that they have to purchase these from Lifeboat at list price and that they can not afford to do this. Hence they stick with the sytems which are promoted through dealerships.

I believe that both the CP/M producers and Lifeboat Associates have the full right to market their product anyway they please. However, for this product to serve the needs to a person like myself who is capable and willing to pay for them, they have to offer some type of distributed customer service perhaps comparable to Sears or Penneys. Both of these latter companies have customer service departments in my city. I can go and get almost any type of assistance in operating or implementing any product ordered through the catalog. Software packages are complex items especially when run on existing machinery.

The purpose of my comment was to alert the marketers to the fact that their valuable material was not serviceable in the hands of a remote user.

Most sincerely,
Robert C. Luckey, M.D.

Richland, WA

Dr. Luckey is entirely correct in his claim that many software products – as well as hardware products – are not adequately serviced once they are bought. It was once excusable because of the "newness" of the micro industry; however, that excuse no longer rings true. Vendors of both hardware and software products can and must *increase the way they respond to the buyer after the product is sold. If not, they will simply go under. Only the best deserve to survive in this market; and that is not only a function of those who sell the best product, but of those who maintain their product and serve the customer after the product is sold. Automobile manufacturers who had no warranties or follow-through, once the auto was sold, would quickly go out of business because nobody would buy their products. It should be this way with software and hardware.* Dr. Dobb's *encourages you to send both your praise and horror stories to us.* – *SR*

ANIMAL

Dear Suzanne,

Regarding Charles Wetherell's 'famous programs' in *DDJ* #42, the smallest program that will run the longest time is something like

```
HERE  JUMP  HERE
```

which will run for as long as you let it, besides being portable to any computer I know of.

As far as 'virus' programs, I know of one story: a large development center of a large computer manufacturer (I won't tell you which) had a game called 'Animal' in which one had to guess which animal the program was talking about by making questions to the program and analyzing the answers. What the player did not know, however, was that while the game was being played, the program accessed the user's directory and copied itself into every file (in this machine files are composed of elements, so one can have multiple elements in a file) it could get hold of. In a while, virtually every file of that computer had its own copy of 'Animal,' and I was told that the management held several secret meetings to find a way of getting rid of the 'animal' without ever acknowledging that it existed!

Albertus Caesar
22 The Paddock
Chalfont St. Peter
Bucks SL9 QJQ
England

CORRECTION

Dear Suzanne Rodriguez,

I'm afraid I made a few mistakes in the letter published in *DDJ* #45. The BASIC demonstration program should have read—

```
145 PRINT
150 PRINT "X" = "; CY, C\256
160 PRINT "Y = "; CY, C MOD 256
```

Also, in the article on page 36 there is a reference to a routine on track zero to initialize the 16 ports at 0CF00H. This routine is on the disks from OSI, it is not on the disks referred to in the article. The problem then is not that the ports are overwritten, it is that they are never initialized to begin with.

A late but happy Easter.

Yours truly,
L. Barker
Chicago, Il.

MORE ON KEYPLUS

Dear *DDJ*:

We would like to thank Lt. Ronald Nicol for his favorable review of our Keyplus Integrated Utility Package for the TRS-80 in *DDJ* #45. However, there are two points that need clarification.

Lt. Nicol states "The documentation that comes with the software is excellent with the exception that the chart of graphic symbols is inaccurate." Not true; we triple-checked and the chart is 100% accurate.

Later on Lt. Nicol remarks that all utilities, even ones not being used, must be resident in memory. The Lt. believes this to be a drawback. In all fairness, it should be pointed out that the tape loads in just 20 seconds and requires less than 800 bytes of space; furthermore, utilities may be enabled or disabled in just two key strokes! Isn't this better than fooling around with several tapes each with a separate utility on it?

One final point, a disk-based version of Keyplus is available for $19.95. Disk Keyplus contains several enhancements.

Sincerely,
Steven Wexler
SJW, Inc.
P.O. Box 438
Huntingdon Valley, PA 19006

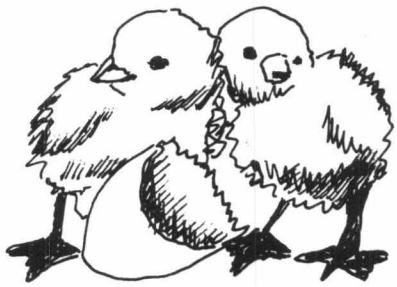
TINY-C ON PHIDECK

Dear *DDJ*:

The May Special Issue – The C Programming Language– struck a responsive cord at this location. A few of your readers might be interested to know that I have recently been licensed by Mr. Gibson to market an implementation of tiny-c (TM) for Digital Group Phideck systems. For the sum of $25 I will supply a tape and documentation. No overlays are used, so any version of Phimon can use tiny-c. A minimum of 26K, including the 2K set aside for Phimon, is necessary. The cassette tape has been prepared with a Phideck readable copy of tiny-c on one side and an audio copy on the flip side in case the Phideck version cannot be loaded. I also include several tiny-c utilities which I have developed. All of this, following Mr. Gibson's lead in the tiny-c Owners Manual, is accompanied by many pages of documentation.

Any prospective user is urged to purchase the Owners Manual from tiny-c associates. Anyone wishing to know more details can write to me.

Yours truly,
Charles Hemminger
DIGITAL GROUP Phideck Version
20 Harrison Ave.
Northampton, MA 01060



A DIFFERENT ERROR

Dear *Dr. Dobb's*:

Bruce Thomson's program in *DDJ* #45:

```
10 REM ** THIS PROGRAM IS TO POINT OUT A PROBLEM **
20 PRINT "COMPUTERS NEVER MAKE MISTAKES"
30 LET X = X + 1
40 IF X = 10 THEN 60
50 GOTO 10
60 REM ** LETS STOP PRINTNG THIS JUNK **
70 END
```

I thought the problems with this program were:

1. LETS should be LET'S
2. X was never initialized

not that transfers were made to REM statements in two places.

Sincerely,
Mark D. Senn
400 N. River Rd., #904
West Lafayette, IN 47906

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 48 | September 1980 | Volume 5, Issue 8

A REFERENCE JOURNAL FOR USERS OF SMALL COMPUTERS

A Runtime Library for the Small c Compiler 4
Ron Cain

Hardcopy Device Driver Programs for CP/M 34
Steve Willoughby

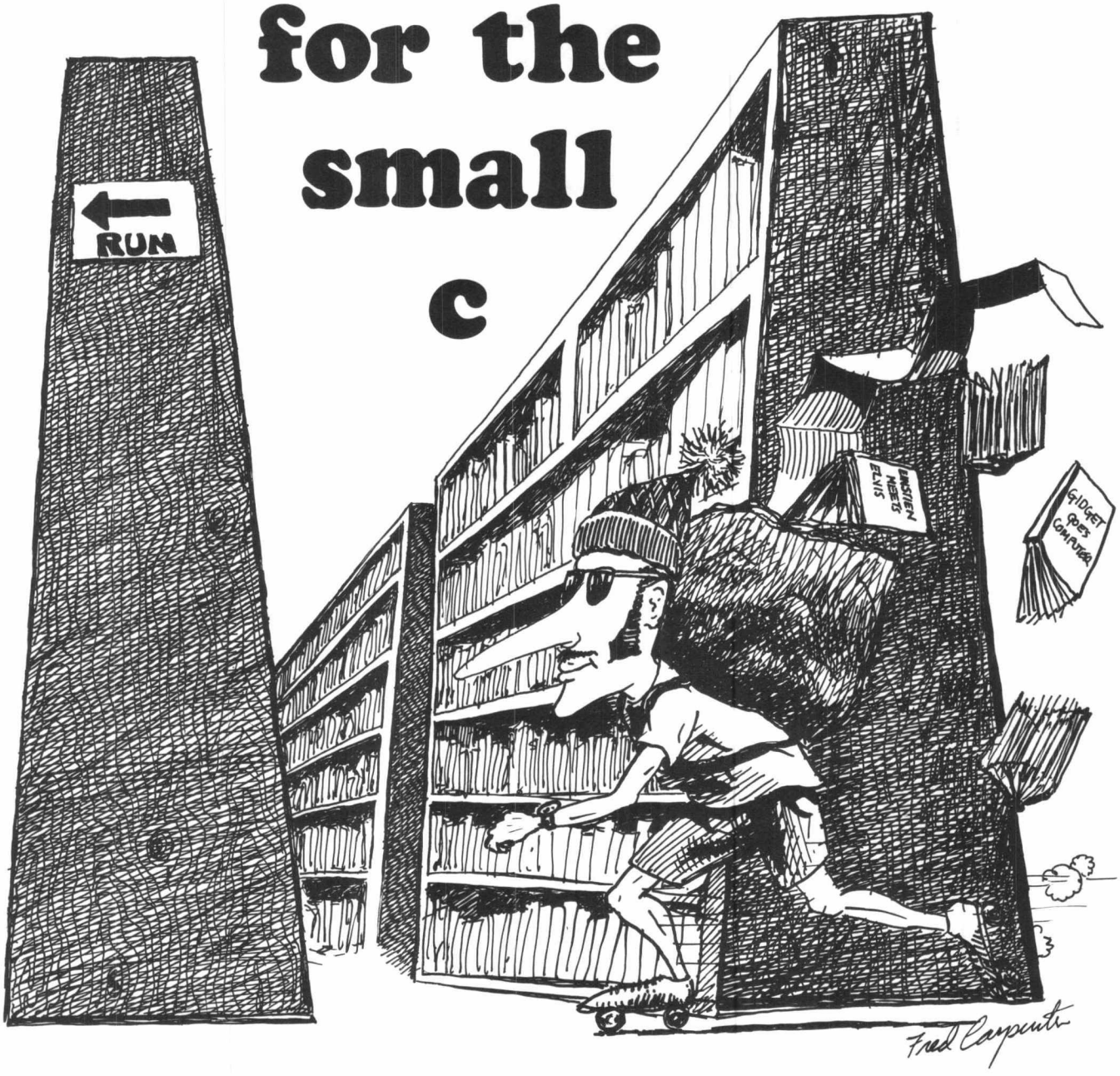
A Note on 6502 Indirect Addressing 26
W.D. Maurer

The Love Program 38
Gary Gaugler

RUNTIME LIBRARY

BY RON CAIN
811 Eleventh Ave.
Redwood City, Ca. 94063

for the small c



COMPILER

The May 1980 issue of Dr. Dobb's Journal (DDJ #45), *highlighting the C Programming Language, was one of the most popular issues ever. Foremost in popularity among the articles was "A Small C Compiler for the 8080's," by Ron Cain. Due to popular demand, we are now publishing Cain's "Runtime Library for the Small C Compiler."*

The small C compiler for 8080's published in *DDJ* #45, made reference to a runtime library. Although the listing of that runtime library did not appear in that issue due to the sheer mass of code of the compiler itself, there has been sufficient demand (and there is now sufficient room) for it.

Actually, the library consists of two distinct modules of code. It contains: (1) the arithmetic and logical routines which are called from the code generated by the compiler, and (2) the I/O routines likely to be called by the user.

The two modules are largely independent of one another, and it is conceivable you might choose to put them in separate files. Such a split would be exceedingly easy to accomplish. All the routines from "ccgchar" to "cccmpbcde" are the arithmetic and logical routines. All the routines after that are related to I/O.

Perhaps it would be helpful to discuss how the compiler generates code to use the arithmetic and logical routines within the library. Suppose you have the following piece of code:

```
int ramword;
char rambyte;
samp (value) int value; (return (ramword / value – rambyte);)
```

The compiler would generate the following code:

```
samp:
                LHLD ramword
                PUSH H
                LXI H, 4
                DAD SP
                CALL ccgint
                POP D
                CALL ccdiv
                PUSH H
                LDA rambyte
                CALL ccsxt
                POP D
                CALL ccsub
                RET
ramword: DS 2
rambyte: DS 1
```

You will notice that, as I mentioned in the previous article, the compiler is very cautious in its use of registers. For all the binary operators, it always gets the first operand into the primary register (HL), puts it on the stack, gets the second operand (also into HL), pops the first operand off the stack into the secondary register (DE), and then does the operation.

In the example above, it first evaluates "ramword" into HL with a simple 16-bit load. Seeing the " / ", it pushes HL onto the stack, and evaluates "value" into HL. Since "value" is defined as an argument passed to this routine, the compiler has to generate the address of where "value" is on the stack. At the time of the call to "samp", the stack would have looked like:

```
SP + 0 –> LSB of return address
SP + 1 –> MSB of return address
SP + 2 –> LSB of "value"
SP + 3 –> MSB of "value"
```

However, after pushing "ramword", the stack would have looked like:

```
SP + 0 –> LSB of ramword
SP + 1 –> MSB of ramword
SP + 2 –> LSB of return address
SP + 3 –> MSB of return address
SP + 4 –> LSB of "value"
SP + 5 –> MSB of "value"
```

Therefore, the pair of lines of code:

```
LXI H, 4
DAD SP
```

would generate the address of "value" in HL. To get the contents of "value", the compiler generates a call into the runtime library to the routine "ccgint". Looking at the listing of "ccgint" (the third routine in the library), you'll see it loads the HL register with the 16-bit integer pointed to by HL. Therefore, upon return from "ccgint", the compiler knows HL contains the contents of "value". Since it now has both operands for the operation, it pops into DE the operand for "ramword" it previously put on the stack, and calls another library routine "CCdiv" to perform the divide. The routine "ccdiv" returns the quotient in HL, which the compiler dutifully puts on the stack upon seeing the "–". Since "rambyte" is defined as an 8- bit entity, the compiler evaluates it with a simple 8-bit load intruciton. Since it also assumes all operands to be 16 bits, it sign extends "rambyte" into a full 16-bit value in HL by calling the library routine "ccsxt". Nearly done then, the compiler pops the previously evaluated quotient off the stack, calls another routine to do the subtract, and exits. Any routine calling the routine "samp" would then have the results expected in the HL register.

It's really not very complicated, and after looking at it for a moment you can see why the compiler can be so simple. All binary operators work in the same fashion. The few utility routines (like "ccgchar"and "ccgint") work in a very predictable way, and inevitably return a 16-bit value in HL. In fact, the routine "ccgint" is the compiler's universal routine for indirect references, which you see in code like:

```
int * ptr;
samp ( )   return ***** ptr;
```

which would generate:

```
samp:
                LHLD ptr
                CALL ccgint
                CALL ccgint
                CALL ccgint
                CALL ccgint
                CALL ccgint
                RET
ptr:            DS 2
```

Naturally, I would expect that many people would like to improve and expand their own runtime libraries. One simple improvement noticed by Walt Bilofsky (and since implemented in my own version of the compiler), lies in the fact that all the binary library routines (ccdiv, ccsub, ccxor, etc.) are always invoked from the generated code by:

```
POP D
CALL ccxxxx
```

The obvious move is to modify the compiler to not generate the "POP D" instruction, and to modify the library routines to contain their own "POP D" instruction. Let's trace such a modification down as an exercise:

In the case of the divide operator, look at the listing of the compiler in the other issue. In the C routine "heir 9( )", you'll see where it parses the "/" character and performs the division. Where you see the two lines of code:

```
pop( );
div( );
```

simply remove the "pop( )", and the compiler will no longer output the "POP D" instruction prior to the "CALL ccdiv". Then, in the routine "ccdiv"in the library, simply insert the three lines of code:

```
POP B          ;unstack return address
POP D          ;unstack operand into DE
PUSH B         ;restack return address
```

at the beginning, and you will have improved your version of the compiler. The number of "POP D"instructions generated is considerable, and the compiled code will be made smaller by this simple modification, However, shorter code is not necessarily faster code. The extra "POP B" and "PUSH B" instructions in the library routine represent 32-bits worth of memory reference, and memory reference instructions are the slowest. So you can make your own version "optimized" (I use the term loosely) either for speed or for space.

Another improvement to the compiler can be made to the routine "ccpchar". If you'll glance at the routine "putstk" in the compiler listing, you'll notice it has the line of code:

```
if (typeobj==cchar) call ("ccpchar");
        else call ("ccpint");
```

Anytime the compiler wishes to store a single character at the address contained in DE, it calls "ccpchar". Looking at the routine "ccpchar" in the runtime library, you'll notice it consists of the code:

```
ccpchar:    MOV  A, L
            STAX D
            RET
```

Those familiar with the 8080 will realize the call instruction takes more room than the routine itself. Therefore, an obvious improvement for both code compaction as well as speed would be to make the compiler drop this code inline. Changing that one line of code in "putstk" to:

```
if (typeobj==cchar)  ol ("MOV A, L"): ol ("STAX D") ;}
        else call ("ccpint");
```

would accomplish this with a minimum of effort.

In fact, any or all of the routines in the library could be executed inline if desired simply by finding where the compiler generates the instruction "call("ccxxxx")" and replacing that with calls to the routine "ol" for all the necessary instructions. The compiler began life putting the code inline, and only after it became obvious which way the wind was blowing (toward another 8K memory board) did I change it to generate this threaded code.

Other mods to the runtime library might include the addition of floating-point routines. I am in the process of adding Charles Falconer's floating-point routines to my own library. In time, either he or I will probably make the compiler smart enough to generate calls to these routines to handle floating-point numbers. Until then, I will simply call them as external routines from the program I write. Perhaps you have math routines in ROM, or an AMD 9511 chip, or some other favorite set of math routines. The runtime library is as good a place as any to put them if you expect to use them frequently. String comparers, string movers, random number generators; these are all good candidates for the libarary.

Apart from the arithmetic and logical routines, I mentioned the runtime libarary also contains the I/O routines. My version supports Polymorphic's 8810/8813 disk operating system. I know this is not the most popular choice, and I can hear the grousing already about the need to convert to CP/M. But console yourself with the fact that I probably would not have gone to the effort of writing the compiler if I'd been able to buy one of the already-existing CP/M C compilers.

The I/O routines I chose to support are by no means the most efficient or universal set, but they run under UNIX, and as I'll explain later, there were reasons.

For your own system, there are only a few routines you will need to support. The runtime library listed here has considerably more than these, but I use the other routines only at times when it becomes necessary to step outside the normal UNIX interface. The routines which the compiler needs (and which do obey UNIX conventions) are as follows:

1. fopen (sname, mode) char *sname, *mode;

This routine will attempt to open the filename pointed to by "sname" according to the character pointed to by "mode". If "mode" points to an ASCII 'r', it will attempt to open for reading. If it points to an ASCII 'w', it will attempt to open for writing. If the routine can successfully open the file for the specified mode, it will return a nonzero integer value, otherwise it will return a zero. The filename obeys the convention for strings by ending in a null byte (a zero), but other than that is free to look like a filename specific to the operating system. For example, I might attempt "fopen("<1> file.tx", "w");" where a CP/M user might try "fopen ("B:file. txt", "w");". On my system, I enforce that no more than one file be open for output on a given drive at a time. A CP/M user would probably not need this precaution.

The integer returned by this routine will reflect the "channel number" for this particular file. Subsequent reads and writes will all refer to this number. You may choose to implement it either as a simple integer (from 1 to N depending on the number of channels you'll allow) or an an address of some form of file control block. UNIX returns a pointer to a file structure.

2. fclose (iochannel) int iochannel;

This routine will close the file specified by "iochannel", which is the channel number returned from fopen( ). If that requires flushing internal system buffers to complete the file (say, if characters had been buffered in memory instead of on disk), then this is done. There is no success or failure indication returned from this routine.

3. getc(iochannel) int iochannel;

This routine will return the next sequential character from the file specified by "iochannel" (which, once again, is the channel number returned by the fopen( ) for this file. If an error occurs or an end of file is encountered, a -1 is returned. A zero character is legal.

4. putc(c,iochannel) char c; int iochannel;

This routine will write the given character "c" to the file specified by "iochannel". The character itself is returned if the write was successful, otherwise a -1 is returned.

5. gets (buff) char *buff;

This routine inputs characters from the keyboard and stores them in the buffer pointed to by "buff". The carriage-return or newline character terminating the string is replaced by a null. My version allows simple editing with deletes and control-X's, but you can make yours do whatever you like.

6. getchar( )

This routine returns the next available character from the keyboard. If no character is available, it will wait until there is.

7. putchar(c) char c;

This routine will print the character "c" onto the user's terminal.

The routines given above are probably the minimum number you will want to support. They are all the compiler needs, so for starters, they are probably enough. Adding routines later is not difficult. In fact, you may have noticed the final part of the runtime library was written in C to provide a front-end for some I/O routines I already had. I wrote the necessary routines compiled them and appended the compiled code to the library. Total time: maybe 15 minutes.

There are many ways to improve and expand this runtime library. I already mentioned a few relating to the arithmetic operators, but I'd like to make a few suggestions about the I/O routines.

Though many alternatives exist, two which seem immedi-

ately obvious are (1) make it smaller or (2) make it bigger.

An example of the former was sent to me by Gene Cotton of Fountain Valley who wanted to put together a minimal configuration for his system. His approach is straight-forward and surprisingly lucid. He deleted all my I/O routines, and added an innocuous routine:

```
;       CP/M SUPPORT ROUTINE
:       bdos (DE,C)
;       char *DE; int C;
;       returns H=B, L=A per CPM manual
bdos:   POP     H       ;hold return address
        POP     B       ;get bdos function number
        POP     D       ;get DE register argument
        PUSH    D       ; now put them
        PUSH    B       ;   back on
        PUSH    H       ;     the stack
        JMP     5       ;now off the CPM bdos
```

As CP/M users can see, this is merely a front end for the call to location 5 which they have come to know and love. It takes two arguments which it puts into DE and C respectively. Hence, a routine to get a character from the keyboard becomes:

```
getchar( )          (return bdos (1, 1);)
                    To print a character becomes:
putchar(c) char c;  bdos (c, 2); return c;}
```

In addition to making a tiny runtime library, Gene further surrounded the code with a "#ASM ... #ENDASM" construct (as described in the compiler article), called it "C. LIB", and begins his C programs with:

```
#include C.LIB
```

This means the runtime library is inserted right into the file being generated. A nice touch. The only penalty (if you want to call it that) lies in the need for the user's program to do its own bdos( ) calls to open files, read files, and close files. If you're already familiar with the way your own operating system works and would prefer to continue to work with it, I would suggest a similar approach. The Code Works (who are now distributing the little compiler for CP/M) independently invented a nearly identical routine to simplify the interface.

However, if you want to make your I/O libarary independent of your operating system, and add some bells and whistles in the process, I would like to lay out some guidelines for expansion.

From the very beginning, since C does not inherently possess a knowledge of an operating system, people have been writing their own I/O interfaces to their system. Seeing this as a serious threat to the portability of the language, the users eventually came up with a "standard" I/O package.

Rather than reliving this learning experience in the hobby field, I suggest all implementations and improvements aim toward this standard. For those who don't know it, I'll summarize the important routines as described in an article called "A New Input-Output Package" by D. M. Ritchie.

Programs expecting to do I/O usually begin with the line of code:

```
#include <stdio.h>
```

This file contains many vital I/O definitions and parameters. Among them (and worth including in all implementations) are:

```
#define stdin some-unique-value
#define stdout some-unique-value
#define stderr some-unique-value
#define EOF -1
#define NULL 0
#define BUFSIZ 512
```

These definitions provide system-wide agreement regarding certain terms, and you will see these in UNIX programs.

The definitions stdin, stdout, and stderr are the first glimpse of one of the powers of UNIX. They are essentially default "channel numbers" for the keyboard, the console screen, and the error device, respectively. They do not need to be opened with an "fopen( )", and they can be used in place of a channel number. Therefore the instruction "getc (stdin)" would return a single character from the keyboard, while "putc (c, stdout)" would print the character "c" on the console. This is not an unusual convention, but the power lies in the ability of a user at the keyboard to redirect these default channel numbers to disk files (or any other peripheral). Hence, if a program is written to input from stdin and output to stdout, it will interact with the user terminal in its normal mode of operation. However, if the user invokes that program with some syntax like "%foobar > output < input arg arg arg ... arg", he will effectively have opened the file "output" to receive all writes to stdout, and the file "input" to provide bytes to reads from stdin. Without modifying the program, it has been changed to accept input from and send output to disk files. The channel for stderr cnanot be redirected, however, and writing to it will always send the output to the user's terminal.

The definitions "NULL" and "EOF" provide names for some common values (zero and −1, respectively) likely to be returned by routines. BUFSIZ is merely the size of single I/O block (a disk sector in practice) and lets the user setup buffers of the appropriate size if needed. For floppy disk sytems, BUFSIZ should be defined as 256.

The routines which follow are the most important of the I/O library. Most are mentioned in the book *The C Programming Language* by Brian Kernighand and Dennis Ritchie. To anyone considering using the C language and especially those writing their own I/O libraries, I would suggest this book. For those routines too detailed to discuss here, I will make references to it.

1. fopen (filename, type) char *filename, *type;
   As described above, this routine opens a file for I/O and returns either a nonzero integer for success or the value NULL for failure. In addition to 'r' for "read" and 'w' for "write", the argument "type" may point to an ASCII 'a' for "append". My version does not support 'a'.
2. freopen (Filename, type, iochannel) char *filename, *type; int iochannel;
   This routine closes the file specified by iochannel, and then reopens it according to the new specifications of filename and type. The value returned is either the new channel number of NULL.
3. getc (iochannel) int iochannel:
   This routine, as described above, returns the next sequential byte from the specified file. An EOF is returned for an end of file or error.
4. fgetc (iochannel).
   Just like getc (iochannel).
5. putc (c, iochannel) char c; int iochannel;
   This routine, as described above, writes the byte "c" to the given file. The character is returned if successful, otherwise EOF is returned.
6. fputc(c, iochannel).
   Just like putc (c, iochannel).
7. fclose (iochannel) int iochannel;
   This routine closes the given file. No specific value is returned.
8. fflush(iochannel)
   This routine empties any system-buffered characters to

the file. It guarantees they are transferred to the output device.

9. exit (errcode) int errocode:

This routine cleans up whatever is needed, and exits to the operating system, returning the value in "errcode".

10. feof (iochannel)

This routine returns a nonzero value if the given file is at the end of file.

11. ferror (iochannel)

This routine returns a nonzero value if an error has occurred while accessing the given file.

12. getchar( )

This routine is effectively "return getc(stdin);".

13. putchar(c) char c:

This routine is effectively "return putc (c, stdout);".

14. gets (buffer) char *buffer;

This routine inputs a string from stdin. The newline (or carriage return) character is replaced by a null. It is the user's responsiblity to guarantee the buffer is big enough. It returns either its argument (address of buffer) for success or NULL for an end of file.

15. fgets (buffer, count, iochannel) char *buffer; int count, iochannel;

This routine reads up to "count" characters into the given buffer from the specified file. A newline (or carriage return) character terminates the string and automatically is followed by a null. It returns either the address of the buffer for success of NULL if an error occurs.

16. puts (buffer)char *buffer;

Writes the null-terminated string from the buffer to stdout. A newline (or carriage return) is appended.

17. fputs (buffer, iochannel) char *buffer; int iochannel;

This routine writes the null-terminated string to the specified file. No newline is appended. No value is returned.

18. ungetc(c, iochannel) char c; int iochannel;

This routine offends my sense of fair play. It "pushes" the given character back out the given input file so the next read will accept it. Only one character may be pushed back.

19. rewind (iochannel) int iochannel;

This routine causes the specified file to be rewound to the beginning. It is usually used only on input files, since rewinding an output file essentially discards all previous output.

20.

printf(format, a1, a2, . . . an)
fprintf (format, a1, a2, . . . an)
scanf (format, a1, a2, . . . an)
fscanf (format, a1, a2, . . . an)
fread (ptr, size, n, iochannel)
fwrite (ptr, size, n, iochannel)
system (string)
atof (string)
cfree (ptr)
calloc (n, size)
strcat (s1, s2)
strcmp (s1, s2)
strcpy (s1, s2)
strlen (s)
isalpha (c)
isupper(c)
islower (c)
isdigit (c)
isspace (c)
toupper (c)
tolower (c)

All these are described in the manual. They are all nice to have but are more suited to a set of utility programs (maybe kept in a separate file) than they are to a runtime library.

There are a few other routines not mentioned, but not all of them are supported even on UNIX systems, so they have been omitted. Not all of the routines above need to be supported, but at least you have an idea of what a full implementation would entail.

As I mentioned, my version supports only the minimum number of these routines, and it does not know about stdin, stdout, or stderr. Actually, the version I am currently working on in my spare time will be very close to everything I have described above. As I mentioned before, it will contain Charles Falconer's floating point routines.

To make C programs run in an environment more similar to UNIX, I am making the compiler (via the routine header ( )) begin all programs by generating a call to a routine named "enter" which will do all necessary system housekeeping, will pull arguments off my system command line into a construct like the ARGV, ARGC from UNIX (which I won't describe here, but which is described in detail in the manual) and will call "main ( )". When "main ( )" returns, it will return control to the routine "exit" which will close all files left open before returning to my operating system.

In time, I may not even return to the operating system. Consider a small UNIX-style shell (command line interpreter) which takes the place of the operating system (but which uses all the hooks available). With a powerful I/O library and the ablity to redirect stdin and stdout, it seems obvious that the notion of a micro-UNIX is not far fetched.

Interested? Well, at the moment, you have all the tools available that I do: (1) A compiler, (2) a standard I/O library, and (3) an already existing operating system to use as an example. For starters, that ain't half bad.

LISTING

```
;*****************************************************
;                                                    *
;       Runtime Library for Small C Compiler         *
;                                                    *
;                 by Ron Cain                        *
;                                                    *
;                                                    *
;*****************************************************
;
;
;Fetch a single byte from the address in HL and
; sign extend into HL
ccgchar: MOV    A,M
;Put the accum into HL and sign extend through H.
ccsxt:  MOV     L,A
        RLC
        SBB     A
        MOV     H,A
        RET
;Fetch a full 16-bit integer from the address in HL
;into HL
ccgint: MOV     A,M
        INX     H
        MOV     H,M
        MOV     L,A
        RET
;Store a single byte from HL at the address in DE
ccpchar: MOV    A,L
        STAX    D
        RET
;Store a 16-bit integer in HL at the address in DE
ccpint: MOV     A,L
        STAX    D
        INX     D
        MOV     A,H
        STAX    D
        RET
;Inclusive "or" HL and DE into HL
ccor:   MOV     A,L
        ORA     E
        MOV     L,A
        MOV     A,H
        ORA     D
        MOV     H,A
        RET
;Exclusive "or" HL and DE into HL
ccxor:  MOV     A,L
        XRA     E
        MOV     L,A
        MOV     A,H
        XRA     D
        MOV     H,A
        RET
;"And" HL and DE into HL
ccand:  MOV     A,L
        ANA     E
        MOV     L,A
```

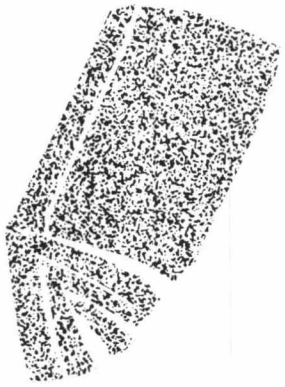
```
        MOV     A,H
        ANA     D
        MOV     H,A
        RET
;
;In all the following compare routines, HL is set to 1 if the
;condition is true, otherwise it is set to 0 (zero).
;
;Test if HL = DE
cceq:   CALL    cccmp
        RZ
        DCX     H
        RET
;Test if DE != HL
ccne:   CALL    cccmp
        RNZ
        DCX     H
        RET
;Test if DE > HL (signed)
ccgt:   XCHG
        CALL    cccmp
        RC
        DCX     H
        RET
;Test if DE <= HL (signed)
ccle:   CALL    cccmp
        RZ
        RC
        DCX     H
        RET
;Test if DE >= HL (signed)
ccge:   CALL    cccmp
        RNC
        DCX     H
        RET
;Test if DE < HL (signed)
cclt:   CALL    cccmp
        RC
        DCX     H
        RET

;Common routine to perform a signed compare
; of DE and HL
;This routine performs DE - HL and sets the conditions:
;       Carry reflects sign of difference (set means DE < HL)
;       Zero/non-zero set according to equality.
cccmp:  MOV     A,E
        SUB     L
        MOV     E,A
        MOV     A,D
        SBB     H
        LXI     H,1     ;preset true condition
        JM      cccmp1
        ORA     E       ;"OR" resets carry
        RET
cccmp1: ORA     E
        STC             ;set carry to signal minus
        RET
;
;Test if DE >= HL (unsigned)
ccuge:  CALL    ccucmp
        RNC
        DCX     H
        RET
;
;Test if DE < HL (unsigned)
ccult:  CALL    ccucmp
        RC
        DCX     H
        RET
;
;Test if DE > HL (unsigned)
ccugt:  XCHG
        CALL    ccucmp
        RC
        DCX     H
        RET
;
;Test if DE <= HL (unsigned)
ccule:  CALL    ccucmp
        RZ
        RC
        DCX     H
        RET
;
;Common routine to perform unsigned compare
;carry set if DE < HL
;zero/nonzero set accordingly
ccucmp: MOV     A,D
        CMP     H
        JNZ     ccucmp1
        MOV     A,E
        CMP     L
ccucmp1: LXI    H,1
        RET


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

;Subtract HL from DE and return in HL
ccsub:  MOV     A,E
        SUB     L
        MOV     L,A
        MOV     A,D
        SBB     H
        MOV     H,A
        RET
;Form the two's complement of HL
ccneg:  CALL    cccom
        INX     H
        RET
;Form the one's complement of HL
cccom:  MOV     A,H
        CMA
        MOV     H,A
        MOV     A,L
        CMA
        MOV     L,A
        RET
;Multiply DE by HL and return in HL
;(signed multiply)
ccmult: MOV     B,H
        MOV     C,L
        LXI     H,0
ccmult1: MOV    A,C
        RRC
        JNC     ccmult2
        DAD     D
ccmult2: XRA    A
        MOV     A,B
        RAR
        MOV     B,A
        MOV     A,C
        RAR
        MOV     C,A
        CRA     B
        RZ
        XRA     A
        MOV     A,E
        RA
        MOV     E,A
        MOV     A,D
        RAL
        MOV     D,A
        ORA     E
        RZ
        JMP     ccmult1
;Divide DE by HL and return quotient in HL, remainder in DE
;(signed divide)
ccdiv:  MOV     B,H
        MOV     C,L
        MOV     A,D
        XRA     B
        PUSH    PSW
        MOV     A,D
        ORA     A
        CM      ccdeneg
        MOV     A,B
        ORA     A
        CM      ccbcneg
        MVI     A,16
        PUSH    PSW
        XCHG
        LXI     D,0
ccdiv1: DAD     H
        CALL    ccrdel
        JZ      ccdiv2
        CALL    cccmpbcde
        JM      ccdiv2
        MOV     A,L
        ORI     1
        MOV     L,A
        MOV     A,E
        SUB     C
        MOV     E,A
        MOV     A,D
        SBB     B
        MOV     D,A
ccdiv2: POP     PSW
        DCR     A
        JZ      ccdiv3
        PUSH    PSW
        JMP     ccdiv1
ccdiv3: POP     PSW
        RP
        CALL    ccdeneg
        XCHG
        CALL    ccdeneg
        XCHG
        RET
;Negate the integer in DE
;(internal routine)
ccdeneg: MOV    A,D
        CMA
        MOV     D,A
        MOV     A,E
        CMA
        MOV     E,A
        INX     D
        RET
;Negate the integer in BC
;(internal routine)
ccbcneg: MOV    A,B
        CMA
        MOV     B,A
        MOV     A,C
        CMA
        MOV     C,A
        INX     B
        RET

;Rotate DE left one bit
;(internal routine)
ccrdel: MOV     A,E
        RAL
        MOV     E,A
```

```
        MOV     A,D
        RAL
        MOV     D,A
        ORA     E
        RET
;Compare BC to DE
;(internal routine)
cccmpbcde: MOV  A,E
        SUB     C
        MOV     A,D
        SBB     B
        RET
;
;I/O subroutines
;
;Input a character into HL
getchar:        CALL    WHO     ;Get one
        JMP     ccsxt   ;Sign extend
;
;Output a character (putchar(char))
putchar:        LXI     H,2
        DAD     SP      ;index past return
        MOV     A,M     ;get char
        CALL    WH1
        JMP     ccsxt
;
;Test for a character(return in HL if found,but don't dequeue)
CHRDY:  LHLD    KBIG
        LDA     KBIP
        SUB     L
        JZ      CHRDY1
        MOV     A,M
CHRDY1: JMP     ccsxt
;
;Disk file I/O
;
FDENUM  EQU     4       ;Number files supportec at once
FDESIZE EQU     310     ;Number bytes required for each file
;
;Offsets into unit tables
FDISKADD EQU    45
FSECTCNT EQU    47
FBYTEOFF EQU    49
FBUFF   EQU     50
;
;Open a file
;Call:  FOPEN(direction,unit,size,nameptr)
;       direction = 1 to read, 2 to write
;       unit = 1, 2, 3, ...(# units avail)
;       size = max size (ignored)
;       nameptr = addr of asciz filename
;Returns: 0=good open, -1=open failure
;
FOPEN:  LXI     H,2
        DAD     SP      ;ptr to arg list
        LXI     D,FOPARGS
        MVI     C,8
FOP1:   MOV     A,M
        STAX    D
        INX     H
        INX     D
        DCR     C
        JNZ     FOP1    ;transfer args to usable area
        CALL    FOPFDE  ;get unit's block
        JC      FOPERR
        MOV     A,M     ;icle?
        ORA     A
        JNZ     FOPERR
        XRA     A
        STA     NFDIR   ;force directory load
        LHLD    FILENAME
        XCHG
        LXI     H,FFILE
        MVI     C,0FFH
FOPEN1: LDAX    D       ;Get filename
        MOV     M,A
        INX     H
        INX     D
        INR     C
        ORA     A
        JNZ     FOPEN1
        MOV     A,C
        ORA     A
        JZ      FOPERR  ;Null
        DCX     H
        MVI     M,0DH   ;append <CR>
        LHLD    FDE
        INX     H
        XCHG            ;DE = FDE addr
        LXI     H,FFILE
        LXI     B,'TX'
        MVI     A,60H
        CALL    Owrto
        DB      'Gfid'
        PUSH    PSW
        LDA     FTYPE
        DCR     A
        JNZ     FOPWRITE
;Read open
        POP     PSW
        JC      FOPERR
        JMP     FOPOK
;Write open
FOPWRITE: POP   PSW
        JNC     FOPERR
        LXI     H,0300H
        CALL    cccmp   ;was it error 300?
        JNZ     FOPERR
;did not exist...good.
        LHLD    FDE
        INX     H
        MOV     A,M
        ANI     3       ;get drive #
        MOV     D,A
        MVI     E,0
        LXI     H,FDEBUFF
        LXI     B,FDESIZE
        MVI     A,(FDENUM AND 0FFH)
FOPWRT1:        PUSH    PSW
        MOV     A,M
        CPI     2
        JNZ     FOPWRT2
        INX     H
        MOV     A,M
        DCX     H
        ANI     3
        CMP     D       ;any other writes on this drive?
        JNZ     FOPWRT2
        INR     E
FOPWRT2: DAD    B
        POP     PSW
        DCR     A
        JNZ     FOPWRT1
        MOV     A,E
        ORA     A
        JNZ     FOPERR  ;yes...fail
;File looks ok
FOPOK:  LHLD    FDE
        INX     H
        INX     H
        MOV     A,M
        INR     A
        INR     A
        INR     A
        ANI     01FH
        MOV     C,A
        MVI     B,0
        DAD     B
        XCHG            ;DE = @FDA
        LHLD    FDE
        LXI     B,FDISKADD
        DAD     B
        MVI     C,4
FOPOK1: LDAX    D
        MOV     M,A     ;copy FDA & DNS
        INX     H
        INX     D
        DCR     C
        JNZ     FOPOK1
        XRA     A
        MOV     M,A     ;clear byte offset
FOPOK2: INX     H
        MVI     M,0
        DCR     A
        JNZ     FOPOK2  ;clear buffer
        LHLD    FDE
        LXI     B,FDISKADD
        DAD     B
        MOV     E,M
        INX     H
        MOV     D,M
        XCHG
        LXI     D,350
        MOV     A,H
        CMP     D
        JNZ     FOPOK3
        MOV     A,L
        CMP     E
FOPOK3: JNC     FOPERR  ;At end of disk
        LHLD    FDE
        LDA     FTYPE
        MOV     M,A     ;declare open
        LXI     H,0
        RET
FOPERR: LXI     H,-1    ;open error
        RET
;
;Find the unit block for unit in FUNIT
;carry set on error
FOPFDE: LDA     FUNIT
        MVI     B,(FDENUM AND 0FFH)
        LXI     D,FDESIZE
        LXI     H,FDEBUFF
FOPFDE1: DCR    A
        JZ      FOPFDE2
        DAD     D
        DCR     B
        JNZ     FOPFDE1
        STC             ;error
        RET
FOPFDE2:        SHLD    FDE
        STC
        CMC
        RET
;
;Close a file
;Call:  FCLOSE(unit)
;no returned value
FCLOSE: LXI     H,2
        DAD     SP
        MOV     A,M     ;get unit
        STA     FUNIT
        CALL    FOPFDE
        RC
        MOV     A,M
        CPI     2
        CZ      FCLSWT  ;write need special close
        LHLD    FDE
        MVI     M,0
        RET
;
;Close an output file
;carry set if error
FCLSWT: LHLD    FDE
        LXI     B,FBYTEOFF
        DAD     B
        MOV     A,M
        ORA     A
        JZ      FCLSWT2
        LXI     H,FZERO
        SHLD    FMEM    ;set ptr to...
        MVI     M,0     ;a known null word
```

```
        CALL    FWRTB   ;empty system buffer
        JNC     FCLSWT
FCLSWT2:        LHLD    FDE
        LXI     B,FSECTCNT
        DAD     B
        MOV     E,M
        INX     H
        MOV     D,M
        DCX     H
        MOV     A,E
        ORA     D
        STC
        CMC
        RZ              ;nothing written
        PUSH    D       ;DNS
        LHLD    FDE
        INX     H
        INX     H
        MOV     A,M
        INR     A
        INR     A
        INR     A
        ANI     01FH
        MOV     C,A
        MVI     B,0
        DAD     B
        INX     H
        INX     H       ;HL = @DNS
        POP     D
        MOV     M,E
        INX     H
        MOV     M,D     ;put in FDE
        LHLD    FDE
        INX     H
        MVI     A,1
        CALL    Gvrtn
        DB      'Gfid'
        RET
;
;Read from a file
;Call:  FREAD(memaddr,bytes,unit)
;       memaddr = beginning addr of user buffer
;       bytes = positive number of bytes to read
;Returns: -1 = error in read
;         else returns number of bytes successfully read
;       (ie 0 = end of file)
FREAD:  CALL    FXARGS
        LXI     H,0
        SHLD    FXFER   ;preset xfer cnt
        CALL    FOPFDE
        JC      FXFERR
        MOV     A,M
        DCR     A
        JNZ     FXFERR
;Main read loop
FREAD1: LHLD    FCNT
        MOV     A,H
        ORA     L
        JZ      FXOK    ;done
        MOV     A,h
        ORA     A
        JNZ     FREAD3
;less than 256
        CALL    FREADB  ;read a byte
        JC      FXFERR
        JMP     FREAD1  ;and loop till done
;More than 256 means we go direct to user memory
FREAD3: LHLD    FDE
        LXI     B,FBYTEOFF
        DAD     B
        MOV     A,M
        ORA     A
        JZ      FREAD5
        CALL    FREADB  ;we have some in system buff first
        JC      FXFERR
        JMP     FREAD1
FREAD5: LHLD    FDE
        LXI     B,FSECTCNT
        DAD     B
        MOV     E,M
        INX     H
        MOV     D,M     ;DE = # sectors left
        LDA     FCNT+1  ;get msb # bytes
        MVI     H,0     ;HL = # sectors requested
        MOV     A,H
        CMP     D
        JNZ     FREAD6
        MOV     A,L
        CMP     E

FREAD6: JC      FREAD7
        XCHG            ;user wants too many
FREAD7: MOV     A,H
        ORA     L
        JZ      FXOK    ;if zero...done
        PUSH    H       ;save # to read
        MOV     A,L     ;(# < 256)
        PUSH    PSW
        LHLD    FDE
        INX     H
        MOV     A,M
        ANI     3
        MOV     C,A     ;unit
        MVI     B,1     ;read cmd
        LXI     D,FDISKADD-1
        DAD     D
        MOV     E,M
        INX     H
        MOV     D,M     ;DE = disk addr
        LHLD    FMEM    ;HL = mem addr
        XCHG
        POP     PSW
        CALL    Dio     ;read
        POP     H       ;# attempted
        JC      FXFERR
;good read
        XCHG            ;DE = # sectors read
        LHLD    FDE
        LXI     B,FDISKADD
        DAD     B
        MOV     A,E
        ADD     M
        MOV     M,A
        INX     H
        MOV     A,D
        ADC     M
        MOV     M,A     ;bump ahead disk addr
        INX     H
        MOV     A,M
        SUB     E
        MOV     M,A
        INX     H
        MOV     A,M
        SBB     D
        MOV     M,A     ;count off # sectors
        LHLD    FMEM
        MOV     B,E
        MVI     C,C     ;make # bytes
        DAD     B
        SHLD    FMEM    ;bump ahead mem addr
        LHLD    FXFER
        DAD     B
        SHLD    FXFER   ;& # bytes read
        MOV     A,B
        CMA
        MOV     B,A
        MOV     A,C
        CMA
        MOV     C,A
        INX     B
        LHLD    FCNT
        DAD     B
        SHLD    FCNT    ;count down # bytes
        JMP     FREAD1  ;loop
;
;Read a single byte into user's memory
;carry set if error
FREADB: LHLD    FDE
        LXI     B,FBYTEOFF
        DAD     B
        MOV     A,M
        ORA     A
        JNZ     FRDBYT
;nothing in system buffer
        DCX     H
        MOV     A,M
        DCX     H
        ORA     M       ;any left?
        JZ      FRDBX   ;no
        DCX     H
        MOV     D,M
        DCX     H
        MOV     E,M     ;DE = disk addr
        PUSH    D
        LHLD    FDE
        INX     H
        MOV     A,M
        ANI     3
        MOV     C,A     ;unit
        MVI     B,1     ;read cmd
        LXI     D,FBUFF-1
        DAD     D       ;internal buff addr
        XCHG
        POP     H
        MVI     A,1     ;1 sector
        CALL    Dio
        RC              ;error
        LHLD    FDE
        LXI     B,FDISKADD
        DAD     B
        INR     M
        INX     H
        JNZ     FRDB2
        INR     M       ;bump disk addr

FRDB2:  INX     H
        MOV     A,M
        ORA     A
        JNZ     FRDB3
        DCR     M       ;decrement sector count
        INX     H
FRDB3:  DCR     M
        XRA     A
;accum = offset to next byte
FRDBYT: LHLD    FDE
        LXI     B,FBUFF
        DAD     B
        MOV     C,A
        MVI     B,0
        DAD     B
        MOV     A,M     ;get byte
        LHLD    FMEM
        MOV     M,A     ;put in user memory
        INX     H
        SHLD    FMEM
        LHLD    FXFER
        INX     H
        SHLD    FXFER
        LHLD    FCNT
        DCX     H
        SHLD    FCNT
        LHLD    FDE
        LXI     B,FBYTEOFF
        DAD     B
        INR     M       ;bump offset (0 means empty)
FRDBX:  STC
        CMC
        RET
;
;Read/write okay
FXOK:   LHLD    FXFER   ;get # bytes done
        RET
FXFERR: LXI     D,-1    ;error exit
        RET
```

```
;
;Transfer the arguments from the stack
FXARGS: LXI     H,4
        DAD     SP
        LXI     D,FXALIST
        MVI     C,6
FXALP:  MOV     A,M
        STAX    D
        INX     H
        INX     D
        DCR     C
        JNZ     FXALP
        RET
;
;
;Write to a file
;Call:  FWRITE(memaddr,bytes,unit)
; (see FREAD for explanation...they are same)
;
FWRITE: CALL    FXARGS
        LXI     H,0
        SHLD    FXFER
        CALL    FOPFDF
        JC      FXFERR
        MOV     A,M
        CPI     2
        JNZ     FXFERR
;Main writing loop
FWRITE1:        LHLD    FCNT
        MOV     A,H
        ORA     L
        JZ      FXOK    ;done
        MOV     A,H
        ORA     A
        JNZ     FWRITE3
;less than 256
FWRITE2:        CALL    FWRTB   ;write a byte
        JC      FXFERR
        JMP     FWRITE1
;more than 256 means it goes direct to user
FWRITE3:        LHLD    FDE
        LXI     B,FBYTEOFF
        DAD     B
        MOV     A,M
        JNZ     FWRITE2 ;empty internal buffer first
        LHLD    FDE
        LXI     B,FDISKADD
        DAD     B
        MOV     E,M
        INX     H
        MOV     D,M
        MOV     A,E
        CMA
        MOV     E,A
        MOV     A,D
        CMA
        MOV     D,A
        INX     D       ;-FDA
        LXI     H,349
        DAD     D       ;349-FDA = # sectors till end
        XCHG
        LDA     FCNT+1
        MOV     L,A
        MVI     H,0     ;make HL = # requested
        MOV     A,H
        CMP     D
        JNZ     FWRITE4
        MOV     A,L
        CMP     E
FWRITE4:        JC      FWRITE5
        XCHG            ;can't do this many
FWRITE5:        MOV     A,H
        ORA     L
        JZ      FXOK    ;no room at all
        PUSH    H
        MOV     A,L
        PUSH    PSW
        LHLD    FDE
        INX     H
        MOV     A,M
        ANI     3
        MOV     C,A     ;unit
        MVI     B,0     ;write cmd
        LXI     D,FDISKADD-1
        DAD     D
        MOV     E,M
        INX     H
        MOV     D,M     ;DE = FDA
        LHLD    FMEM
        XCHG
        POP     PSW
        CALL    Dio
        POP     D
        JC      FXFERR
;good write
        LHLD    FDE
        LXI     B,FDISKADD
        DAD     B
        MOV     A,E
        ADD     M
        MOV     M,A
        INX     H
        MOV     A,D
        ADC     M
        MOV     M,A
        INX     H
        MOV     A,E
        ADD     M
        MOV     M,A
        INX     H
        MOV     A,D
        ADC     M
        MOV     M,A
        LHLD    FMEM
        MOV     B,E
        MVI     C,0     ;make # bytes
        DAD     B
        SHLD    FMEM
        LHLD    FXFER
        DAD     B
        SHLD    FXFER
        MOV     A,E
        CMA
        MOV     B,A
        MOV     A,C
        CMA
        MOV     C,A
        INX     B
        LHLD    FCNT
        DAD     B
        SHLD    FCNT
        LHLD    FDE
        LXI     B,FDISKADD
        DAD     B
        MOV     E,M
        INX     H
        MOV     D,M
        XCHG
        LXI     D,350
        MOV     A,H
        CMP     D
        JNZ     FWRITE6
        MOV     A,L
        CMP     E
FWRITE6:        JC      FWRITE1 ;still room on disk
        JMP     FXOK    ;disk full, but no more, please
;
;Write a single byte from user memory
;carry set if error
FWRTB:  LHLD    FDE
        LXI     B,FBYTEOFF
        DAD     B
        PUSH    H
        MOV     C,M
        INX     H
        MVI     B,0
        DAD     B       ;index to place to put it
        XCHG
        LHLD    FMEM
        MOV     A,M
        STAX    D       ;copy to system buffer
        INX     H
        SHLD    FMEM
        LHLD    FXFER
        INX     H
        SHLD    FXFER
        LHLD    FCNT
        DCX     H
        SHLD    FCNT
        POP     H
        INR     M       ;bump offset
        JZ      FWRTB1  ;opps, filled buffer
FWRTBX: STC
        CMC
        RET
;Need to write system buffer to disk
FWRTB1: INX     H
        PUSH    H       ;@buff
        LHLD    FDE
        INX     H
        MOV     A,M
        ANI     3
        MOV     C,A     ;unit
        MVI     B,0     ;write cmd
        LXI     D,FDISKADD-1
        DAD     D
        MOV     A,M
        INX     H
        MOV     H,M
        MOV     L,A     ;get the disk addr
        POP     D       ;DE = mem addr, HL = disk addr
        MVI     A,1
        CALL    Dio
        RC              ;error
        LHLD    FDE
        LXI     B,FDISKADD
        DAD     B
        INR     M       ;bump NDA
        INX     H
        JNZ     FWRTB3
        INR     M
FWRTB3: INX     H
        INR     M       ;& DNS
        INX     H
        JNZ     FWRTB4
        INR     M
FWRTB4: INX     H
        INX     H       ;HL = @ buff
        XRA     A
FWRTB5: MVI     M,0
        INX     H
        INR     A
        JNZ     FWRTB5  ;clear buffer after every write
        LHLD    FDE
        LXI     B,FDISKADD
        DAD     B
        MOV     E,M
        INX     H
        MOV     D,M
        XCHG
        LXI     D,350
        MOV     A,H
        CMP     D
        JNZ     FWRTB6
        MOV     A,L
        CMP     E
FWRTB6: JC      FWRTBX  ;still room
        LXI     H,0
        SHLD    FCNT    ;full...no more can be done
        LHLD    FDE
        LXI     B,FBYTEOFF
        DAD     B
        MVI     M,0     ;and declare buffer empty
```

```
        JMP     FWRTBX
;
;File I/O storage area
;Following are the arguments
FOPARGS:
FILENAME: DS    2       ;Addr file name
FNUMB:  DS      2       ;Size on open (unused)
FXALIST:
FUNIT:  DS      2       ;Unit #
FCNT:
FTYPE:  DS      2       ;Direction of transfer
FMEM:   DS      2       ;Current memory address
;Start of local storage
FDE:    DS      2       ;Addr of current unit block
FXFER:  DS      2       ;# bytes done
FFILE:  DS      40      ;filename buffer
FZERO:  DS      1       ;known null word for closing
;
;Unit block formats:
;       +0  = type of open (0 = free, 1 = read, 2 = write)
;       +1  = Poly FDE block (max 44 bytes)
;       +45 = FDA (also NDA) next disk addr for xfer
;       +47 = DNS sector count (reads decrements, write inc's)
;       +49 = offset to next byte in internal buffer
;       +50 = internal buffer (256 bytes)
FDEBUFF:        DS      FDESIZE * FDENUM

;
;
        DS      10      ;fudge factor
;
;Following are some routines written in C to provide a UNIX compatible
;front-end for the already existing I/O routines.
;
;small-c compiler rev 1.0
;/*                                      */
;/*   Patch to poly I/O routines                 */
;/*                                      */
;#define NULL 0
;#define eol 13
;fopen(sname,mode)
fopen:
;       char *sname,*mode;
;{      int xfermode,unit;
        PUSH B
        PUSH B
;       if(*mode=='w')xfermode=2;
        LXI H,6
        DAD SP
        CALL ccgint
        CALL ccgchar
        PUSH H
        LXI H,119
        POP D
        CALL cceq
        MOV A,H
        ORA L
        JZ cc10001
        LXI H,2
        DAD SP
        PUSH H
        LXI H,2
        POP D
        CALL ccpint
;               else xfermode=1;
        JMP cc10002
cc10001:
        LXI H,2
        DAD SP
        PUSH H
        LXI H,1
        POP D
        CALL ccpint
cc10002:
;       unit=0;
        LXI H,0
        DAD SP
        PUSH H
        LXI H,0
        POP D
        CALL ccpint
;       while(unit<4)
cc10003:
        LXI H,0
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,4
        POP D
        CALL cclt
        MOV A,H
        ORA L
        JZ cc10004
;               if(FOPEN(xfermode,++unit,-1,sname)==0)
        LXI H,2
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,2
        DAD SP
        PUSH H
        CALL ccgint
        INX H
        POP D
        CALL ccpint
        PUSH H
        LXI H,1
        CALL ccneg
        PUSH H
        LXI H,14
        DAD SP
        CALL ccgint
        PUSH H
        CALL FOPEN
        XCHG
        LXI H,6
        DAD SP
        SPHL
        XCHG
        PUSH H
        LXI H,0
        POP D
        CALL cceq
        MOV A,H
        ORA L
        JZ cc10005
;                       return unit;
        LXI H,0
        DAD SP
        CALL ccgint
        POP B
        POP B
        RET
;       return NULL;
cc10005:
        JMP cc10003
cc10004:
        LXI H,0
        POP B
        POP B
        RET
;       }
;fclose(unit)
fclose:
;       int unit;
;{      return FCLOSE(unit);
        LXI H,2
        DAD SP
        CALL ccgint
        PUSH H
        CALL FCLOSE
        POP B
        RET
;}
;getc(unit)
getc:
;       int unit;
;{      char buff[1];int flag;
        DCX SP
        PUSH B
;       flag=FREAD(buff,1,unit);
        LXI H,0
        DAD SP
        PUSH H
        LXI H,4
        DAD SP
        PUSH H
        LXI H,1
        PUSH H
        LXI H,11
        DAD SP
        CALL ccgint
        PUSH H
        CALL FREAD
        POP B
        POP B
        POP B
        POP D
        CALL ccpint
;       if((flag==0)|(flag==-1))return -1;
        LXI H,0
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,0
        POP D
        CALL cceq
        PUSH H
        LXI H,2
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,1
        CALL ccneg
        POP D
        CALL cceq
        POP D
        CALL ccor
        MOV A,H
        ORA L
        JZ cc10006
        LXI H,1
        CALL ccneg
        INX SP
        POP B
        RET
;       return(buff[0]&255);
cc10006:
        LXI H,2
        DAD SP
        PUSH H
        LXI H,0
        POP D
        DAD D
        CALL ccgchar
        PUSH H
        LXI H,255
        POP D
        CALL ccand
        INX SP
        POP B
        RET
;       }
;putc(c,unit)
putc:
;       char c;int unit;
;{      char buff[1];int flag;
        DCX SP
        PUSH B
;       buff[0]=c;
        LXI H,2
        DAD SP
        PUSH H
        LXI H,0
```



```
        POP D
        DAD D
        PUSH H
        LXI H,9
        DAD SP
        CALL ccgchar
        POP D
        CALL ccpchar
;       flag=FWRITE(buff,1,unit);
        LXI H,0
        DAD SP
        PUSH H
        LXI H,4
        DAD SP
        PUSH H
        LXI H,1
        PUSH H
        LXI H,11
        DAD SP
        CALL ccgint
        PUSH H
        CALL FWRITE
        POP B
        POP E
        POP B
        POP D
        CALL ccpint
;       if((flag==0)|(flag==-1))return -1;
        LXI H,0
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,0
        POP D
        CALL cceq
        PUSH H
        LXI H,2
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,1
        CALL ccneg
        POP D
        CALL cceq
        POP D
        CALL ccor
        MOV A,H
        ORA L
        JZ cc10007
        LXI H,1
        CALL ccneg
        INX SP
        POP B
        RET
;               else return(c&255);
        JMP cc10008
cc10007:
        LXI H,7
        DAD SP
        CALL ccgchar
        PUSH H
        LXI H,255
        POP D
        CALL ccand
        INX SP
        POP B
        RET
cc10008:
;       }
        INX SP
        POP B
        RET
;gets(buff)
gets:
;       char buff[];
;{
;       int k;
        PUSH B
;       k=0;
        LXI B,0
        DAD SP
        PUSH H
        LXI H,0
        POP D
        CALL ccpint
;       while((buff[k]=getchar())!=eol)
cc10009:
        LXI H,4
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,2
        DAD SP
        CALL ccgint
        POP D
        DAD D
        PUSH H
        CALL getchar
        POP D
        CALL ccpchar
        PUSH H
        LXI H,13
        POP D
        CALL ccne
        MOV A,H
        ORA L
        JZ cc10010
;               {
;               if(buff[k]==24){k=0;putchar(24);}
        LXI H,4
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,2
        DAD SP
        CALL ccgint
        POP D
        DAD D
        CALL ccgchar
        PUSH H
        LXI H,24
        POP D
        CALL cceq
        MOV A,H
        ORA L
        JZ cc10011
        LXI H,0
        DAD SP
        PUSH H
        LXI H,0
        POP D
        CALL ccpint
        LXI H,24
        PUSH H
        CALL putchar
        POP B
;               else if(buff[k]==127)
        JMP cc10012
cc10011:
        LXI H,4
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,2
        DAD SP
        CALL ccgint
        POP D
        DAD D
        CALL ccgchar
        PUSH H
        LXI H,127
        POP D
        CALL cceq
        MOV A,H
        ORA L
        JZ cc10013
;                       {if(k>0)
        LXI H,0
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,0
        POP D
        CALL ccgt
        MOV A,H
        ORA L
        JZ cc10014
;                               {k=k-1;
        LXI H,0
        DAD SP
        PUSH H
        LXI H,2
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,1
        POP D
        CALL ccsub
        POP D
        CALL ccpint
;                               putchar(127);
        LXI H,127
        PUSH H
        CALL putchar
        POP B
;                               }
;                       }
cc10014:
;               else
        JMP cc10015
cc10013:
;                       {
;                       putchar(buff[k]);
        LXI H,4
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,2
        DAD SP
        CALL ccgint
        POP D
        DAD D
        CALL ccgchar
        PUSH H
        CALL putchar
        POP B
;                       k=k+1;
        LXI H,0
        DAD SP
        PUSH H
        LXI H,2
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,1
        POP D
        DAD D
        POP D
        CALL ccpint
;                       }
cc10015:
cc10012:
;               }
        JMP cc10009
cc10010:
;       buff[k]=0;
        LXI H,4
        DAD SP
        CALL ccgint
        PUSH H
        LXI H,2
        DAD SP
        CALL ccgint
        POP D
        DAD D
        PUSH H
        LXI H,0
        POP D
```

```
        CALL ccpchar
;       putchar(eol);
        LXI H,13
        PUSH H
        CALL putchar
        POP B
;       return buff;
        LXI H,4
        DAD SP
        CALL ccgint
        POP B
        RET
;}

;0 errors in compilation.
;
;       >>>>>  end of runtime library  <<<<<
```

ENDING

Thoughts on Small Systems and Monitors

(mostly the SYM-1)

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

I define a "small" system as one that is owned and used by only one person. This still allows more than a ten-fold range in cost and complexity, but the common ground is the man/machine interaction that can be (for the human being) a profound educational experience. My first system was a KIM-1, the creation of Don McLaughlin while he was at MOS Technology. This early SBC launched the new 6502 (in the dim and distant past of 1976!) and, together with brilliant descriptive manuals, played a major role in its popularization. It was—belatedly—emulated by the makers of other microprocessors, and of course surpassed by more complex systems of the multiboard type. As a learning tool it still has few (perhaps no) equals. All of my previous communications, in *DDJ* and elsewhere, were worked out on my KIM.

One of its invaluable assets is the complete and quite well-commented listing of the code of its 2K ROM monitor, a simple "operating system", rich in information on how to program. Like other users I soon became aware that many of its routines were callable by new programs, although this had not been clearly foreseen by whoever wrote the monitor programs. By now, this ought to be recognized as one of the major functions of every monitor: to provide a "library" of efficiently-coded routines accessible to newly-created main programs. In a ROM, the need to squeeze many operations into a limited space will often cause brevity of code to be given priority over maximum speed of execution. Also, subprograms will tend to be special-purpose instead of general-purpose. You can't have everything!

However, one service that ought to be included with the documentation of a monitor listing is a list of user-accessible subroutines, so that users need not waste time searching them out. Such a list may be longer than the number of RTS instructions in the monitor, since a subroutine may have several useful entry points (including some undreamed of by the person who wrote it!) The use of more than one exit point, which can sometimes enhance both code efficiency and timing, is often stigmatized as "unstructured." That's a word I have come to dislike. Everything has structure. The distinction that matters is *intelligibility*, and this is not necessarily lost by using two (or even more) exit points. What must be avoided is excess of any kind, including excessive rigidity and complexity.

The Upgrading/Customization Problem. Everyone must sooner or later outgrow a system as confining as an SBC, and the question becomes: upgrade to what? More crucial still, for what? Perusing the articles and ads in microcomputer journals is bewildering. So many competing systems, add-on peripherals, interfaces, languages, complex softwares, and above all uses! I long deferred any upgrading decision, in the hope that major innovations would soon appear. My concept of my own needs gradually crystallized: a system that could sense, measure, analyze, and act on the outside in a way similar—but superior-to the way I do and so do my work for me.

While "dedicated" microprocessors are already automating a variety of instruments, that's not quite what I want. There's a colossal wastage built in to the scientific (and doubtless also the business) world, caused by non-standardization, deliberate avoidance of modularity/interchangeability, and planned obsolescence.

Except at the leading edge, where state-of-the-art can only be had by custom design, "packaged" systems dominate scientific laboratories. Each has its built-in electronics, recorders, etc., so tied together that if one element obsolesces, the whole thing must be thrown away, brushed-aluminum chassis and all. These genies simultaneously serve and enslave their users, who become button-pushers with no comprehension of or power to modify the mechanisms.

The concept that has always appealed to me is more that of the "hi-fi" enthusiasts, where a system consists of several interacting but independently-replaceable components, from many competing sources. True competition, however abhorrent to industrial giants, makes possible a maximum of progressive change at the minimum cost to society. This eccentric, unbundled-components point of view enabled me to resist the lure of the increasingly powerful packaged systems that have entered the market in recent years. However flexible, they must be designed for some least-common-denominator purpose and tend to allocate large resources to things like BASIC interpreters (for me a turn-off). They are like low-cost smoothly-paved roads leading to where everyone else wants to go.

My first upgrade was a Synertek SYM-1. It is even more flexible and I/O-oriented than the KIM. Even future-oriented, since major ICs are socketed and one hopes that upgrading from the 6502 to the 6516 will be easy. The SYM is a masterwork, complex and not free from flaws, that encourages and even compels its user to explore unbeaten paths. It works best only for someone willing to make the effort to under-

stand it.

One thing the SYM can do is interface with a TTY or CRT terminal, via RS-232 at baud rates from 110 to 4800 (well below the potential of a 6502 CPU). It is obvious that a CRT—not a TTY—is the primary upgrading element of an expanded system. The Commodore PET built one in, while the Apple II, TRS-80, etc., built in only an ASCII keyboard and allow (really require) interfacing to a TV monitor. These systems tend to have less than the 24 x 80 character display of stand-alone CRT terminals, but to some extent compensate this by providing graphics, and sometimes color. Many of their limitations will soon be transcended (probably first by the Apple III). Conventional CRT terminals will soon face superior price/performance competition, and their too-high price is already falling. I therefore decided to buy one, the Televideo 920-B, that seemed a fair value at its current near-$800 mail-order price. It lacks the graphics capability of the 920-C, but I don't mind that since my mind is a symbol-oriented algebraic, not a geometric one. Of course, so-called "printer graphics" of the TTY type are possible. The 920-B is an "intelligent" terminal controlled by an Intel 8035 micro-processor, with a 32K ROM and sockets for 8 high-speed 2114 static RAMs (allowing storage of an "alternate" 24 x 80 character page). Competitive pressure has now made all intelligent terminals so good that I have no idea whether the 920-B is one of the "best," but it is very satisfactory. I like its eleven "function" keys, that send a 3-character message to the computer: 01-XX-OD, where XX ranges from 40 to 4A with the F keys alone, and from 60 to 6A if the shift key is also depressed, for a total of 22 different commands. The user decides how the computer shall interpret them. It also has its own RS-232 interface to a printer, so that the user can print exactly what he sees on the screen.

When I connected the 920-B to the SYM-1, the log-in command (S, J, 1, CR) from the SYM's onboard keyboard did not work. A study of the monitor listing led me to modify the vector in the monitor RAM at $A622 from its set value of $A7 to $71. This worked, proving that having a listing—and being able to read it—is a good thing, and also that the redundancy of *dual* control has merit.

The slowness of 4800-baud serial transmission (that seems to fill the screen at about 80 characters/second) is a shade annoying to a lover of speed. The 920-B can work at 9600 baud (very good for serial), but the SYM designers were not thinking in those terms. Although I've known the theory of communication from books, this first real-life encounter with it has been educational. One sympathizes with the problems of the pioneers trying to drive messages through long transmission lines—losses, noise, errors and the like—and sees the why of parity and checksums. The SYM monitor recognizes the low probability of transmission errors by just ignoring parity. The 920-B doesn't (unless the user so desires) since it might anytime get hooked to a telephone line! It could easily connect via a modem to a timeshared system such as MicroNet.

Comments on the SYM monitor program. Like many other KIM-trained machine-code programmers, I am keenly aware of the beauty of both code-efficiency and time-efficiency. Also of higher virtues of clarity, simplicity, modularity, and generality. Hardly ever are all of these wholly compatible.

The writers of the SYM monitor were required to pack as much power and versatility as possible into a 4K-byte space (in a socketed 2332 ROM). The coding gives priority to byte-saving devices, sacrificing timing. They had to complete the task quickly, at the cost of incomplete testing. Version 1.0 had an error in its KIM-format audio-cassette program, corrected in a new version 1.1 ROM (that nonetheless had the log-in-to-CRT flaw mentioned above). Its high-speed audio-cassette interface worked (in my hands) only with Microsette or Data Sound data tape.

The SYM monitor has a command language, similar in kind to that of KIM but far more complex, and designed to be easily user-expansible. I explored the possibility of re-interpreting the keyboard-command keys of the KIM in my EDITHA program (*DDJ* #25; although there are only 4 keys, anything one may desire can be done by implementing key *sequences*. However, the SYM expects to interact with an ASCII keyboard, so that a very large number of single-key commands could be implemented. Those that are recognized by the monitor in its command-entry mode are: B, C, D, E, F, G, L1, L2, LP, M, R, S1, S2, SD, SP, V, and W. Six of the 2-character commands (all but SD) deal with loading and storing of programs with audio cassette or paper tape. The others are useful in program-writing, viewing, running, and debugging (too esoteric to go into detail). A command is acknowledged by displaying it and printing a space. The monitor then shifts to a hex-digit entry mode, that can accept (and print) up to 3 sets of 4 hex digits, delimited by a comma (or hyphen), to serve as numerical parameters for the command. Pressing the CR (carriage return) key then causes execution if the command is valid, i.e., interpretable by the monitor. Only the M command is defined for all 4 parameter possibilities (0, 1, 2, or 3). If an undefined number of parameters has been keyed-in, control shifts to an "unrecognized syntax" vector. An unrecognized command character shifts control to a vector for that.

The computer elite (and some not-so-elite) will naturally view a mere command language as an antiquated anachronism, when so many wondrous high-level languages exist! I do some reading in the HLL domain but see nothing irresistably great there. A language can imprison the mind. True, you can encode complex logic in symbols that may seem simple, and shift the arduous task of programming and antibugging machine code to an abler mind than your own. But errors are instructive, tutelage can dim vision, and few gains are not balanced by (often unseen) losses.

A Preliminary Monitor-Enhancement Program. The "unrecognized command" mode is free-wheeling and congenial for machine-linguists. In its command-entry state, the SYM monitor is sent any ASCII character it will not recognize. It echoes this, and a space, then awaits 1, 2, or 3 four-digit hex parameters (more could be had, but I prefer to limit complexity). I reserve the last one keyed in for the address of the program to be run. Therefore it is akin to the monitor's own G (for GO) command, except that the user can provide parameters for the program to use. The unrecognized-command vector is reset to the address of a simple 23-byte interpreter (INPRET, cf. listings) that copies the second keyed-in parameter to zero-page to serve as an indirect address for program instructions, then jumps-indirect to the last meter to shift control to the program, whose address is the *real* command. The program in the listings is moderately complex but not optimized (as much of my earlier work was). It is an exercise in interaction between the SYM-1 and the 920-B, a sketch of something useful that may become even more so, perhaps even worth putting into an E-PROM. I was driven to write it because the SYM monitor expects to work with "dumb" peripherals (or even with only its own keyboard and hex display, in a quite ingenious way). There is no provision for teamwork with an intelligent terminal, and its TTY-style programs for displaying, modifying, and moving memory are restrictive.

My new programs implement only some of the possible intercommunication between the SYM and the 920-B. While this specific system may now be a unique one, it may arouse some interest in the teamwork concept, that is feasible for any SBC working with any intelligent terminal. The main program, DISPAG (at $0C00 in the listings), is implemented by the command sequence: T XYY, ZZZZ, C00 CR (where CR is the carriage-

return key). Any unrecognized character other than T works as well, since it will shift control to the vector preset in $A66D-E, that in the example is the address of INPRET ($0D00). ZZZZ is the starting address of the block of memory to be displayed, e.g., C00 will display the program itself (note that leading zeroes need not be keyed in.) YY is the hex number of *lines* to be displayed, that must range from 1 to $18. Any other value will yield an error message and return to monitor control. X controls the display format, and need not be keyed in if the "unispace" format is desired; any hex digit from 1 to F will command the "lister" format. This operation is controlled by subroutined LNFORM, that could be modified to allow other formats.

The display will therefore be from 1 to 24 lines of bytes (as hex-digit pairs) of memory. The first four (packed) hex digits are the address of the first byte in the line, separated by one space from the byte sequence . In the unispace format, each byte is followed by one space and each line has exactly 16 bytes. In the lister format, every opcode is preceded by an extra space, so that the *instruction* sequence is more legible. This is useful only for programs, and is meaningless for data tables. To avoid splitting one instruction between two successive lines, the lister format allows up to 18 bytes on one line. It uses my old BYTCNT program, all-too-familiar to *DDJ* readers! Parenthetically, I note that one more 6502 byte-count program, not very efficient, has been published (BYTE, May 1980, p. 190).

When all the lines are displayed, control shifts to the NEXCHR program. This frees the 920-B keyboard for any of its own programmed operations. To return to the SYM monitor, the CR key must be pressed twice in succession. In this mode the display can be edited in any way desired. Wholly new progams can be created on the CRT screen. NEXCHR merely echoes everything the 920-B sends to it, a mode that persists unless a double-CR or a RUBOUT signal is received. This last is a bit dangerous, since it shifts control to a progam sequence (starting at LNREAD) that can wreck the SYM memory if used carelessly. Its purpose is to transfer a sequence of hex-bytes from the CRT screen (really the 920-B memory) to a specified region of the SYM memory. It uses most of the code in the listings, and was the trickiest to write.

For the 920-B to SYM transfer to work correctly, the revised (or brand-new) program on the screen must obey the following restrictions:

1. Each line must start with 4 characters, followed by a space. These will be read but not used in any way. The reason for this is to make it unnecessary to alter the address information that DISPAG has written, when an existing program is being revised (or just moved to another memory location.) It is an inconvenience when a new program is being constructed, unless the user wishes to keep track of addresses, but at this early stage I prefer not to augment the program complexity.
2. The lowest program line must be followed by a completely blank line (unless it is at the very bottom of the screen display, in which case a blank line will be automatically created by the read operation). This is needed to terminate the transfer.
3. The line just above the first program line must also start with 4 characters and a space (again read but not used), and then must have exactly 4 hex characters that specify the SYM RAM address to which the program is to be moved. E.g., 0200 will cause transfer to start at that location in the SYM RAM. The cursor must be to the *right* of that address on the screen when the RUBOUT key is pressed. Since the top line provides the only address information, this must be correct!

It would be possible—perhaps desirable—to include various anti-goof guards (such as a requirement that the RUBOUT key be pressed twice in succession to initiate the transfer) but only use-experience can truly dictate how a program of this type ought to be refined. A few guards do exist. Either a non-hex character or an incomplete hex byte (only one digit) will cause return to monitor control.

The program lines need not be limited to 16 bytes, or be written in any particular format, though the lister format is the most legible. When the transfer has been completed, control shifts to the VERIF program, that will use the DISPAG logic to display the revised or new program that is now in the SYM memory, and again shift to the NEXCHR mode. This allows further revision, and another transfer. This can be kept up until the user is satisfied that the program is ready to be tested. In my experience, it has proved desirable to test tricky logic in the form of subroutines, even when it will ultimately be coded in-line to optimize time-and code- efficiency. Debugging a complex in-line program can be difficult! There are times when I wonder whether really adequate testing is even possible. And is a poorly-tested program really debugged?

The Game of ROM-Creation. Even the minority of programmers who have read the ROMs created by someone else do not fully appreciate the problem until they toy with the idea of creating their own. It's the chasm between critic an artist, or rather between a builder of sand-castles and a sculptor in marble.

My coding would surely seem very sand-castly to the writer(s) of the SYM monitor, who make heavy use of the stack and hardly touch zero-page—while I haven't even one overt stack-save, and transfer information freely via the on-chip registers, and use zero-page with wild abandon. The last is possible because the monitor graciously cedes zero-page to users, using instead its own half-page of RAM staring at $A600. The only zero-page locations it uses are the eight of RAM starting at $A600. The only zero-page locations it uses are the eight from F8-FF. Its stack usage is compelled by a seemingly iron rule that registers must be saved and restored by all subroutines, except when information is to be returned in the accumulator, the status register, or both. The 6502 instruction set allows a 6-byte code sequence for stack-saving (16 cycles) and for restoring (20 cycles) its four major registers PAXY. However, the SYM monitor frequently saves bytes by using a JSR SAVER (3 bytes but 103 cycles!) as the first instruction of a subroutine. SAVER is an intricate 48-byte program that rearranges the stack so that, when it has returned, PAXY is left on the stack. The subroutine that has used SAVER exits by a JMP to one of three entry-points of a program that will restore all registers (29 cycles), only AXY (41 cycles), or only XY (55 cycles) before its RTS termination. This allows three options with only six more bytes added to the subroutine, but the timing penalty is high (I think it's why serial transmission can't exceed 4800 baud.) The philosophic contrast shows in my 62 bytes of code from $0D2F in the listings, that contain 13 different entry points to a single RTS (and 2 others, ESCOUT and TWOCHR, that are potential entry points), averaging less than 5 code bytes per subroutine. This economy is possible only because nothing is saved. Panegyrists of discipline might also frown on the anarchy of my subroutine LNFORM, that uses the Y index for "parameter-passing" and PAX in a self-centered way. Its calls to the monitor subroutines OUTBYT and SPACE, on the other hand, involve no fewer than 28 stack save/restore instructions, plus the musical-chairs stack manipulation of the monitor's SAVER subroutine, for a total of 39 stack operations! Since LNFORM only needs to protect the X and Y registers from alteration by OUTBYT, OUTBYT does a lot of work to svae LNFORM the 8 bytes it would need to do it itself. The *entire* KIM-1 monitor has only *ten* stack pull/push opcodes, but does more saving in zero-page—simpler but with the possibility of conflict, that is of course a powerful argument for using the stack.

One thing is for sure. ROMs tend to use a lot of subroutines. The KIM monitor has 115 JSRs, and it's the dominant opcode. The SYM monitor has 269 JSRs (in twice the ROM space). Knowers of the 6502 will expect that the LDA will be a close second (as it is in the KIM and probably in the SYM, that does not have a static analysis of opcode usage frequency). That is a powerful argument for speeding up subroutine calls, as the lonely genius who designed the Signetics 2650 years ago foresaw. I have no idea who he was (though I am sure the gender is right), but I hope the story of the 2650 will someday be written, since foresight is to me the most miraculous of all human qualities. I hardly need to expatiate, for readers of *DDJ*, on the virtues of subroutines: easy debugging, easy rewriting, and above all availability to any program.

Re-reading my main program (from $0C00 to 0C73), I see that I created seven subroutines, without quite knowing why. All are used exactly once and could have been coded in-line. Why not? LNFORM CVHEX, and RNOSCP are complex enough so that the need for tailorability was perhaps subconsciously obvious. The others are moronic drudgery, and may just indicate something like battle-fatigue. It's wrong to assume that everything constructed by a frail human being is a gem of purest ray serene! We get tired, we get fed-up, we don't feel like recalculating relative-branch offsets, so we do it the easy way. This kind of feeling must have pervaded the SYM monitor-writer(s) near the end of the task. Michelangelo too, when he put the finishing touches on the Sistine Chapel, at an age not far from my own.

In fact, the subroutine-library concept that I lauded at the start of this communication is elusive. Who knows what will be useful to many other workers? No monitor has a bytecount routine, but we now know it's useful because so many have been created. The SYM provides a block-move routine, that many KIM users had to write for themselves. However, its byte-search program can only find single bytes (my EDITHA program can also find sequences of two or three bytes). What length of byte-string is likely to be needed? ROM-design is bedeviled by dilemmas, and is analogous to the design of the CPU itself, likewise frozen in silicon.

One essential of ROM-design is mastery of the CPU instruction set. The SYM monitor shows great skill in complex assembler logic, and is usually very well coded, but there are spots where recoding would save both bytes and time. That kind of thing is rare in the work of Jim Butterfield, who "thinks in 6502", so I assume that the SYM writer(s) learned on a different kind of machine. One gets "conditioned" to a set one uses all the time—much like one of the human languages—so there'll be touches of awkardness if one shifts to a less familiar set (or language), recognizable by a "native speaker." There's also the "personal style" factor, but some styles command admiration even though they differ from one's own.

A ROM also has an "architecture," as a CPU does, and this may be true of any program (even my rough sketch in the listings). The SYM ROM is basically a command-language interpreter. Having read the low-level "instruction," it first branches to one of the four "legal" parameter-blocks (0, 1, 2, or 3 parameters). Each of these checks the "legal" command-characters and, if one is found, shifts control to the proper execution coding. If not, control shifts to the "unrecognized-command" vector, that defaults to an error-message (very simple and clever). This is something more than a set of isolated routines filed so they can be retrieved by keying the right code, since the ensemble serves a common purpose. Perhaps the only component of my new program that merits addition to a monitor is the old BYTCNT routine; all else is system-dependent.

I am reminded of a brief—but highly didactic—exposition of the principles of system design (chapters 1 and 3 in the book *Systems Concepts*, edited by R. F. Miles, Wiley-Interscience, 1973). A "system" is defined as a large, complex, man-made set of concepts and/or elements used to satisfy some human need. All components are integrated to yield a set of optimum outputs from a set of given inputs (some of them random, so that the exact performance at any instant is unpredictable). It works semi-automatically: machines always perform some of the functions and human beings always perform other functions. The design goal is to maximize "effectiveness" for a given cost—that implies trade-offs. I now shift from paraphrase to a key quotation: "Because it is generally impossible to find a single number which realistically represents the effectiveness of a complex system, there is a good deal of subjectiveness, as represented by judgment, as well as objectiveness, as represented by analysis, in systems engineering." The system is an entity such that "... optimization of each subsystem independently will not lead in general to a system optimum, and ... improvement of a particular subsystem actually may worsen the overall system." Food for thought there! Less arguable is the concept that "no system can be all things to all people, all of the time" so that "the fundamental mission of a system should not be jeopardized, nor its fundamental objectives significantly compromised, in order to accommodate events of extremely low probability" since that would make the system "too complex to be workable." This means that someone must guess right the probability of future events! Wow! In the saying *per aspera ad astra*, only the *aspera* are certainties.

A Brief Afterthought on FORTH. To me, FORTH has been the most tantalizing of the existing HLLs. Perhaps the word is infuriating, since the FORTH enthsiasts—like the Rosicrucians or the initiates to the ancient Eleusynian mystery—won't tell you what it is. In comparison, my own much more miniscule programs come with a surfeit of explanatory comment (tinseled with allusions). When I glanced at a listing of FORTH, its most striking quality was the virtual absence of comment. Adam Osborne recently observed (*InfoWorld* 2(8):7, 1980) that the success of an HLL depends less on its intrinsic merit thn on hhow hard it's pushed. Whatever the demerits of BASIC—and they are legion—being unexplained is not one of them. Dozens of books expound it in great detail, and some are brilliant. Where is the book that describes how FORTH works, from the ground up, in a painstakingly detailed, translucent an vivid way?

```
Main Program, 3 sequences: DISPAG, NEXCHR, VERIF. Normal entry
     from Monitor via INPRET (cf. text for details.)

ØC2Ø 2Ø 4D 83  DISPAG JSR CRLF   (carriage return + line feed)
     A6 Ø1             LDX ZBAS+1 (base page Hi in X reg.)
     A5 ØØ             LDA ZBAS   ("    "  Lo in Acc.)
     2Ø F4 82          JSR OUTXAH (print HiLo base address)
     2Ø 42 83          JSR SPACE  (print one space)
     A Ø Ø             LDY # Ø    (zero Y index for this line)
     D A Ø ØØ          JSR LNFORM (print one line of bytes)
     F 2Ø 9Ø ØC                   (# of bytes printed, Y to Acc.)
    12 98              TYA        (clear carry for add)
    13 18              CLC
    14 65 ØØ           ADC ZBAS   (add to base address Lo)
    16 85 ØØ           STA ZBAS   (store new Lo value)
    18 9Ø Ø2           BCC NOPINC (carry clear, no Hi change)
    1A E6 Ø1           INC ZBAS+1 (increment Hi address)
```

```
  1C  CE 4E A6  NOPINC DEC P1L    (decrement line-count)
  1F  D0 DF            BNE DISPAG (if not 0, do next line)

0C21  20 58 8A  NEXCHR JSR INTCHR (get char. from term. kybd.)
   4  C9 0D            CMP #$0D   (a CR?)
   6  D0 07            BNE RUBTES (if not, go test a RUBOUT)
   8  20 58 8A         JSR INTCHR (get next character)
   B  C9 0D            CMP #$0D   (a second CR?)
   D  F0 44            BEQ EXIT   (if yes, back to monitor)
   F  C9 7F     RUBTES CMP #$7F   (a RUBOUT? be careful!)
  31  D0 EE            BNE NEXCHR (no, go get another char.)

 (memory move from terminal memory to SYM memory now begins)

0C33  20 73 0D  LNREAD JSR RNOSCP (move one line as ASCII chars.)
   6  B0 3B            BCS EXIT   (error, not an even #)
   8  30 2E            BMI VERIF  (line all spaces, verify move)
   A  20 B3 0C         JSR CVHEX  (convert ASCII to hex bytes)
   D  B0 34            BCS EXIT   (error, non-hex conversion)
   F  A6 07            LDX ZBAS+7 (load # of chars. moved)
  41  E0 05            CPX #5     (more than four?)
   3  B0 05            BCS MVRAM   (if yes, move bytes to RAM)
   5  20 A4 0D         JSR STORAD (if four, use as base address)
   8  F0 E9            BEQ LNREAD (and read next line)

0C4A  A4 06      MVRAM LDY ZBAS+6 (load # of bytes to be moved)
   C  88         WYDEC DEY        (decrement #)
   D  B5 07            LDA ZBAS+7,X (pick up top byte in buffer)
   F  91 00            STA (ZBAS),Y (store in RAM address)
  51  CA                DEX          (decrement X index twice
   2  CA                DEX           for next lower hex byte)
   3  D0 F7            BNE WYDEC    (if not 0, do next byte)

 (now reset base address for next line-read and total byte count)

0C55  A2 FC              LDX #$FC  (set X index for 4 increments)
   7  A5 06      ADDOP  LDA ZBAS+6 (load # of bytes moved)
   9  18                CLC       (clear carry for add)
   A  75 04             ADC ZBAS+4,X (add to base address Lo)
0C5C  95 04             STA ZBAS+4,X (store new addr. Lo)
   E  90 02             BCC HISAME  (if carry clear, same Hi)
  60  F6 05             INC ZBAS+5,X  (increment addr. Hi)
   2  E8         HISAME INX           (up two for byte count)
   3  E8                INX
   4  D0 F1             BNE ADDOP  (if not 0, add to count)
   6  F0 CB             BEQ LNREAD (if 0, go read next line)

 (the verify sequence is reached when LNREAD read all spaces)

0C68  20 D0 0C  VERIF JSR SETAD  (reset base addr.)
   B  20 DD 0C        JSR LNCNT  (set # lines to display)
   E  20 AC 0D        JSR ZERCNT (zero byte count locations)
  71  F0 8D           BEQ DISPAG (really a forced branch)
  73  60        EXIT  RTS        (back to monitor)

 (main program ended, now come 4 of its subroutines)

0C74  A2 01     BYTCNT LDX #1  (cf. DDJ #22 for much comment!)
   6  2C 0F 80        BIT IRQBRK  (an 08 BIT-mask in SYM ROM)
   9  D0 08           BNE HALFOP
```

```
   B  C9 20            CMP #$20
   D  F0 0E            BEQ THREE
   F  29 9F            AND #$9F
  81  D0 0B            BNE TWO
   3  29 15    HALFOP  AND #$15
   5  C9 01            CMP #1
   7  F0 05            BEQ TWO
   9  29 05            AND #5
   B  F0 02            BEQ ONE
   D  E8         THREE INX
   E  E8           TWO INX
   F  60           ONE RTS

0C90  A2 10     LNFORM LDX #$10  (set X to display 16 bytes)
   2  AD 4F A6 NEXTOP LDA P1H  (is format control parameter)
   5  F0 05            BEQ SAMEOP (if 0, not lister format)
   7  B1 00            LDA (ZBAS),Y  (load opcode byte)
   9  20 74 0C         JSR BYTCNT (get its bytecount, in X)
   C  B1 00    SAMEOP  LDA (ZBAS),Y   (load byte)
   E  20 FA 82         JSR OUTBYT    (print it)
  A1  CA               DEX           (decrement count)
  A2  D0 03            BNE UNISPC    (if not 0, only one space)
  A4  20 42 83         JSR SPACE     (print one space)
  A7  20 42 83 UNISPC  JSR SPACE     ( "     "    "  )
  AA  C8               INY           (increment Y to get next byte)
  AB  C0 10            CPY #$10      (if Y=16+, will set carry)
  AD  8A               TXA           (check if X = 0)
  AE  D0 EC            BNE SAMEOP    (if not, same instruction)
  B0  90 E0            BCC NEXTOP    (if Y not yet 16, next one)
  B2  60               RTS           (line display completed)

0CB3  B5 07      CVHEX LDA ZBAS+7,X  (load topmost ASCII char.)
   5  20 75 82         JSR ASCNIB   (convert to hex, lo nibble)
   8  B0 15            BCS FINIS    (error, not a hex character!)
   A  95 07            STA ZBAS+7,X  (store conversion)
   C  CA               DEX           (to pick up next char.)
   D  8A               TXA           (move to Acc. to test if odd #)

0CBE  4A                LSR A  (bit 0 to carry, set if odd)
   F  B0 F2             BCS CVHEX  (if odd, convert hi half)
  C1  B5 08             LDA ZBAS+8,X  (reload hi conversion)
   3  0A                ASL A    (and do 4 left-shifts to
   4  0A                ASL A      move into hi nibble)
   5  0A                ASL A
   6  0A                ASL A
   7  15 09             ORA ZBAS+9,X  (insert lo nibble)
   9  95 09             STA ZBAS+9,X  (store whole hex byte)
   B  8A                TXA     (test if X now 0)
   C  D0 E5             BNE CVHEX  (if not, do next byte)
   E  18                CLC   (clear carry, no-error exit)
   F  60                RTS

0CD0  A2 FE      SETAD  LDX #$FE  (set X for 2 ops)
   2  38                SEC   (set carry for subtract)
   3  B5 02      SUBIT  LDA ZBAS+2,X  (load address, LO, HI)
   5  F5 04             SBC ZBAS+4,X (sub count, Lo, Hi)
   7  95 02             STA ZBAS+2,X (store new base address)
   9  E8                INX    (increment X to do HI)
   A  D0 F7             BNE SUBIT  (if not 0, go do it)
   C  60                RTS
```

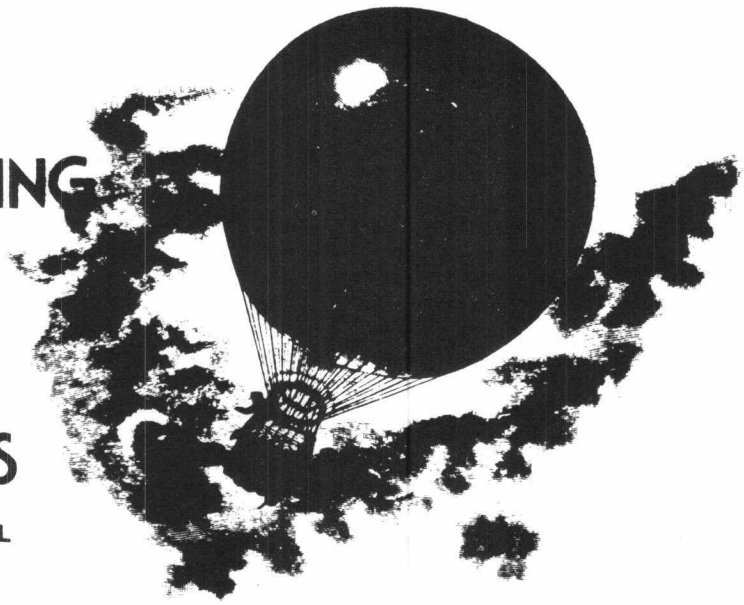
```
ØCDD  A5 Ø2       LNCNT LDA ZBAS+2  (load byte count Lo)
   F  A2 Ø4             LDX #4   (set X for 4 rotate ops)
  E1  18                CLC (clear carry to insert a Ø)
   2  66 Ø3       LNCALC ROR ZBAS+3  (rotate count Hi right)
   4  6A                ROR A      (  "      "    Lo right)
   5  CA                DEX   (four ops done?)
   6  DØ FA             BNE LNCALC  (if not Ø, continue)
   8  AA                TAX   (move line count from A into X)
   9  66 Ø3             ROR ZBAS+3  (rotate the carry in)
   B  FØ Ø1             BEQ STOLIN (if Ø, no fractional line)
   D  E8                INX   (if not, add one whole line)
   E  8E 4E A6  STOLIN  STX P1H  (store line count for DISPAG)
  F1  6Ø                RTS

 (the next program page starts with the unrecognized-command
  interpreter, INPRET.  Subroutines meant for interaction of
  the SYM and the 920-B follow, not all being used in the
  working program, then two new essential subroutines.)

ØDØØ  AD 4E A6  INPRET LDA P1L  (load line-count parameter)
   3  FØ 11             BEQ EROUT (zero is an error, so exit)
   5  C9 19             CMP #$19  (upper display limit is $18)
   7  BØ ØD             BCS EROUT  (if more, exit)
   9  A2 Ø1             LDX #1  (set for base address move)
   B  BD 4C A6   PTOZP  LDA  P2L,X  (pick up addr. parameter)
   E  95 ØØ             STA ZBAS,X  (store in zero page)
  1Ø  CA                DEX
  11  1Ø F8             BPL PTOZP  (get LO)
  13  6C 4A A6          JMP (P3L)  (jump indirect to program)
  16  6Ø          EROUT RTS     (if error, back to moniter)

ØD17  2Ø 5D ØD  CURSLD JSR CURASK  (asks cursor position)
   A  A2 Ø3     TRICHR LDX #3   (set for 3 characters)
   C  2Ø 58 8A  LODCHR JSR INTCHR  (get a character)
   F  95 Ø3            STA ZBAS+3,X (store in zero-page)
  21  CA                DEX     (decrement for next char.)
   2  DØ F8             BNE LODCHR  (if not Ø, get char.)
   4  6Ø                RTS

ØD25  2Ø 61 ØD  CURSET JSR SETCUR (signal to set cursor)
   8  2Ø 4F ØD         JSR YXSEN  (send coordinates)
   B  6Ø               RTS

 (series of multiple-entry character-send routines)

ØD2F  A9 Ø7       BEEP LDA #7   (terminal beep sound)
  31  DØ 36            BNE ONECHR (forced branch, output)
  33  A9 Ø8     CULEFT LDA #8  (move cursor left)
  35  DØ 32            BNE ONECHR
  37  A9 ØA     CUDOWN LDA #$ØA  (move cursor down)
  39  DØ 2E            BNE ONECHR
  3B  A9 ØB     CURSUP LDA #$ØB  (move cursor up)
  3D  DØ 2A            BNE ONECHR
  3F  A9 ØC     CRIGHT LDA #$ØC  (move cursor right)
  41  DØ 26            BNE ONECHR
   3  A9 ØD     CARETN LDA #$ØD  (carriage return)
   5  DØ 22            BNE ONECHR
   7  A9 1F     NEWLIN LDA #$1F  (= a CR,LF)
   9  DØ 1E            BNE ONECHR
   B  A9 1E       HOME LDA #$1E  (send cursor home)
```

```
   D  DØ 1A            BNE ONECHR
   F  A5 Ø4      YXSEN LDA ZBAS+4 (cursor Y in zero-page)
  51  A6 Ø3            LDX ZBAS+3 (cursor X  "     "  )
   3  DØ 1Ø            BNE TWOCHR (send both coordinates)
   5  A2 4B     TOGPAG LDX #$4B  (toggle to 2cd screen page)
   7  DØ ØA            BNE ESCOUT (send ESC, then char.)
   9  A2 36     SENLIN LDX #$36  (send cursor line)
   B  DØ Ø6            BNE ESCOUT
   D  A2 3F     CURASK LDX #$3F (terminal asked cursor position)
   F  DØ Ø2            BNE ESCOUT
  61  A2 3D     SETCUR LDX #$3D (set term. to new cursor status)
   3  A9 1B     ESCOUT LDA #$1B  (load ESC character)
   5  2Ø AØ 8A  TWOCHR JSR TOUT  (send the char. in Acc.)
   8  8A                TXA     (shift char. in X into Acc.)
   9  2Ø AØ 8A  ONECHR JSR TOUT   (send the character)
   C  6Ø                RTS   (sole exit for all routines)

   (now two special subroutines for main program use)

ØD6D  A2 FF       RDALL LDX #$FF  (set to read whole line)
   F  A9 2Ø             LDA #$2Ø  (including space chars.)
  71  DØ Ø4             BNE SETOP (forced branch)
   3  A2 FB     RNOCSP LDX #$FB  (exclude first 4 chars.)
   5  A9 21             LDA #$21  (  "     space chars.)

ØD77  85 Ø5       SETOP STA ZBAS+5 (store select comparand)
   9  86 Ø7             STX ZBAS+7 (save X index)
   B  2Ø 59 ØD          JSR SENLIN (command send cursor line)
   E  2Ø 43 ØD          JSR CARETN (CR, so echo overwrites)
  81  A6 Ø7             LDS ZBAS+7 (restore X index)

  83  2Ø 58 8A  MORCHR JSR INTCHR (accept a sent character)
   6  C5 Ø5            CMP ZBAS+5 (excluded by comparand?)
   8  9Ø Ø7            BCC TESEND (if yes, go test end-of-line)
   A  E8               INX        (increment X index)
   B  3Ø F6            BMI MORCHR (if negative, get next char.)
   D  95 Ø8            STA ZBAS+8,X (if not, store in z.p. buffer)
   F  1Ø F2            BPL MORCHR (forced branch, continue)
  91  C9 ØD     TESEND CMP #$ØD  (a CR? it's end-of-line signal)
   3  DØ EE            BNE MORCHR (if not, continue)
   5  2Ø 37 ØD         JSR CUDOWN (after the exit CR, cursor down,
   8  2Ø 33 ØD         JSR CULEFT  left, sets to end of next line)
   B  E8               INX       (X+1 is actual # of chars. stored)
   C  86 Ø7            STX ZBAS+7  (save character count)
   E  8A               TXA         (move from X into Acc.)
   F  4A               LSR A  (right-shift, divide by 2, =# of bytes)
  AØ  85 Ø6            STA ZBAS+6  (save count of hex bytes)
  A2  8A               TXA       (tests for char. count of Ø)
  A3  6Ø               RTS

 (note: above termination logic is OK for the RNOCSP entry, but not
  for the RDALL entry that has no exclusion logic)

ØDA4  A5 Ø9      STORAD LDA ZBAS+9 (load base address HI)
   6  85 Ø1             STA ZBAS+1 (store for indirect address)
   8  A5 ØB             LDA ZBAS+11 (load base address LO)
   A  85 ØØ             STA ZBAS   (and store for indirect)
   C  A9 ØØ      ZERCNT LDA #Ø    (to re-zero count locations)
   E  85 Ø3             STA ZBAS+3  (first the Hi)
  BØ  85 Ø2             STA ZBAS+2  (then the Lo)
  B2  6Ø                RTS
```

End

PROGRAMMING PASTIMES AND PLEASURES

BY CHARLES WETHERELL



Structured Programming: Disease or Cure?

If you have spent much time around professional programmers and their managers in the last five years, you have certainly heard about structured programming. However, you could be forgiven if you have not made up your mind whether structured programming is a blessing or a curse. As a matter of fact, you might well decide that structured programming is some kind of disease.

The reason for your confusion could be the conversations you had overheard. It is not uncommon to hear that over at Impossibly Big Machines, Inc., they have just taken up structured programming, said in tones suggesting that the company should live through the attack. Or you might hear a project leader from Completely Delerious Computing say thankfully that he finally got his system out in spite of all those fancy rules. Then again, you might go to a meeting trying to sell your application package to Golf & Wasters, Inc. and be told that it won t be considered unless every last IF statement is indented just so. All of this suggests that structured programming is something like staphylococcus—it causes boils, but it also can be turned into a valuable antibiotic.

The Source of the Confusion

A difficulty with the phrase "structured programming" is that it describes a variety of activities, by no means all comparable. First, structured programming is the name given to the program synthesis tehcnhique developed by Dijkstra in his book *A Discipline of Programming* (Prentice-Hall). This method (and similar ones developed by others) aims at building a provably correct program from a logical specification of the program's input and output relations. Two things are required for Dijkstra's structured program: a mathematical analysis of the programming language and considerable mathematical sophistication on the part of the programmer. When these requisites are available, Diskstra's synthesis can be very powerful, but it cannot be surprising that programmers without the necessary training are overwhelmed and bewildered by the heavy symbolism. Still, I can recommend a look at Dijkstra's book if only for the material on practical programming.

Second, structured programming often refers generically to a family of techniques for managing large programming projects. Typically these techniques reduce the input and output specifications of something like an inventory control system to a standard format, a format that reveals the interdependencies and common needs of the system. Structured Analysis and Design Technique, the Jackson Method, HIPO charts—these are a few of the methods available. As might be expected, these methods are not very useful to a programmer working alone. Rather, they are more appropriate for managers and planners in corporations and software houses.

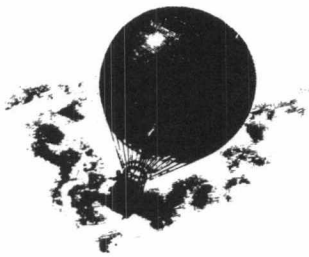
Down to Earth

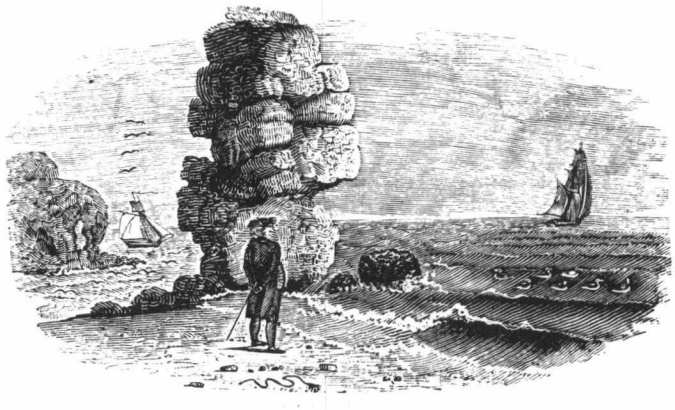
More appropriate to the programmer working alone is the third meaning of structured programming. Here, the term refers to the regulated and standardized analysis of a single program. Top-down programming is perhaps the best known technique of this type, but many methodical programmers have their own home-brew structuring techniques. The earmarks of a structured program are clean procedure interfaces, well described data structures, straightforward control flow, mnemonic variable names, ease of modification, and so on. The good working programmer may already produce programs like this, but structured programming is supposed to increase the probability.

Finally, structured programming often refers to what I call "structured coding." This is the genesis of the canard that structured programming is a blind book of rules governing every comma and keyword. Actually, structured coding is more like a newspaper style book. The purpose is to ensure a uniformity of presentation in all programs. Further, best solutions to common coding problems should be encapsulated in the book so that each programmer need not redevelop (at the programmer's mental expense) the solutions. A good style book does not inhibit a programmer's expressiveness, but it does make sure that his programs can be understood.

Now we can see why there is such a mixed reaction to structured programming. The manager thinks he has to be a mathematical genius to understand it. The line programmer is not very happy at having his creativity "managed." Free spirits don't like any rules at all. And mathematicians don't program much anyway and can't understand all the fuss.

Before I discuss how structured programming might apply to you let me say that the proponents of the techniques I have described might not all want to be lumped together. But it is clear that in the minds of many computer people, they *have* been lumped together. So when you hear structured programming discussed in the future, you will at least be able to decipher the conversation.





What Does Structured Programming Cure?

I assume that most of my readers are hobbyists and program for personal benefit. If that describes you, you may well wonder how structured programming, thought up by academics and industrialists to simplify the economical creation of correct, efficient, maintainable programs, has anything to offer you. Correct and efficient programs are always nice, but what is this about economy and maintenance? Sounds like an ad for an imported car. For the personal programmer, economy means writing a program with as little psychic pain as possible. And maintenance? An old philosopher once said that no man is the same person after six months. In six months, though, the new you is going to have to fix what the old you wrote; better make the repair job as easy as possible. You would be surprised how much you can forget about a program in six months.

And do you need to be programming in PL/I, COBOL, ADA, or some other high-falutin language to use structured programming? No, even in BASIC and assembly language, structure pays off. My third and fourth meanings—standardized analysis and structured coding—can be applied to any size program. We have all had experience beginning a new program in a fever of creativity, writing and testing and debugging and modifying at top speed. What comes out of these bouts? A program that works, but not one you would like to work on. Generally, the result is badly commented, has control flow that resembles a bowl of spaghetti, doesn't work on special cases, and gives readers hives. So back off and rewrite. Use your own style, but do the job tidily. Build on what you learned about how to do the program, but write the program in a structured way. For this is the disease structured programming cures. It cools brain fever.

Polymorphic Talks to the Heath H-14 Printer

BY MARVIN KONOPIK
American Embassy
APO San Francisco 96503

Here is a short update to Marvin Konopik's article in DDJ #44,*"All The Secrets You Ever Wanted About Poly 88 BASIC":*

> I originally tried to warm start the Editor at 2003, but it did not work. As a result of my article in *DDJ*, I was informed that 2003 was the location. I have retried it and it does work.
>
> My request for assistance in locating the end of BASIC brought many responses stating that 48AA is the place to go. This would simplify the process of combining programs drastically.

I purchased the Heath H-14 printer after careful thought and meditation. I certainly never assumed that my limited background in electronics was sufficient to contemplate building an interface to connect a printer to a computer. Desperately in need of professional advice, I called Heath to find out how hard it would be to interface the two. Needless to say, they said they were only concerned with Heath problems. This left only me and the computer to solve the problems.

Armed with determination and some hard-earned cash, I ordered a printer, waited out the delays until it arrived, and began putting it together. It ran tests with only a minimal amount of smoke. The smoke seems to occur only as I become a more experienced Heathkit builder. Actually Heath makes a very thorough, smoke-proof manual, but we professionals seem to lose our ability to read. There are three big questions in this project. First, what kind of signal is the computer putting out? Second, did you make any mistakes in building the printer inputs? Third, with all of those input wires, which ones will be needed to do what? Since I did not have an interface board, luck would be very important.

As I stated above, I am not even *close* to being an engineer, so the following is probably not the right way to do the job, but it worked for me. Give it your best shot, and it might work for you. I needed only 3 wires to make it go. You will need a TTL/RS232 converter from pin 1 (TXD), the computer out line, to pin 5 (SIN), on the printer board. When the printer is busy, it puts out a signal on board pin 7 (RTS). This is a RS232 output, and must be converted to TTL to talk to the computer. The RTS, connects to pin 4 (CTS) on the computer board. The third wire is the ground, to printer board pin 1. I made the TTL/RS232 interface featured in a June 79 *Kilobaud* issue, for the SIN line, and then that portion worked fine. But then the printer was not able to keep up, and I discovered I needed an RTS/CTS link from the Printer to the Computer. I used one of my already-procured spare chips, which I buy automatically when I get new equipment, and which is an absolute necessity when you live overseas. The printer uses two special Integrated Circuits. The 75189 (U103) a RS232/TTL converter and the 75188 (U104), which is a TTL/RS232 converter. 75189 is ideal for use on the RTS line. Had I known what the chips were for in the beginning, I could have used the 75188 (U104) as well for the serial output. The easiest way is to buy the widely advertised kit: however, a 7404 must be used to invert the RTS/CTS signal.

Also had I known that the U103 and U104 were RS232/TTL and TTL/RS232 converters, I would have certainly tried for a shorter, simpler direct TTL hookup, bypassing U103 and U104 completely. A 7400 would have provided enough buffer power, with room left over to invert the signal, if needed.

All that remains to getting it running is the setting of the baud rates. Imagine my surprise when I found that even though the printer will work up to 9600 baud, it still prints at the same line speed. A 9600 Baud printer at H-14 prices would have not only been a definite bargain, but it would be a real paper eater. The high baud rates mean only that it can load the printer with extra text much faster. I found the 4800 baud rate just seemed to work better, so I use that as recommended by the Heath People. But to make things more complicated, I used an unorthodox signal cable. I like things to go on and off with one switch, so I put power and signal in the same cable. Any engineer will tell you that this is a real no-no, but it works for me. If you do not have a plug to fit over the printer board, one half of a 40 pin dip socket works fine. I really felt guilty not using all of those wires in the giant cable Heath provides, and I know the wires must be important or they would not be there, but after hours of testing, I just could not find a place to put them. One wire will tell the computer that the printer is not turned on. If the printer is in the same room, this will be apparent without wiring. Because of frequent moves, my working corner changes size and shapes quite regularly, so I decided to try a worst-possible situation first. I purchased a 20 foot soft cable to give me later portability. A word of caution: if you use the molex plug, be sure to keep the signal and power wires separated and insulated. One side of a molex plug has a block that prevents reverse coupling. It's a little safer if you put the signal wires on this blocked side to prevent accidental contact if the plug is reversed, and tipped in.

I had only minimal software problems. The Poly 88 Editor worked super, but BASIC was putting garbage on the line during non-use. After days of frustration and study of the Poly documentation, I found that BASIC does indeed put out garbage, in fact they call it JUNK. The junk signal was a 7FH (DEL), which I changed to 00H in location 4A7E and the line was quiet.

Another discovery was the lost warm start provision when using the Print program. Warm start is bringing the computer up from reset, to restore not only the BASIC, but the program previously loaded in BASIC. Since BASIC and the Print program were separate, and each had their own starting locations, there was no provision for bringing both up warm. To use the printer program, the procedure was to start it, and it will in turn reach over and start BASIC. But the start is cold, so I merely changed that address, and it comes up warm every time.

My printer printed columns a little irregularly at first, but after a couple of pages the rough edges wear off, and the print quality, with the exception of the funny little 'g's, is quite good. The end result is really fantastic. It is hard to imagine life without a printer, and the H14 is certainly one way to go.

*

A NOTE ON 6502 INDIRECT ADDRESSING

BY W. D. MAURER
George Washington University
S.E.A.S.
Washington, D.C. 20052

There are two kinds of indirect addressing on the 6502, both of them with indexing. Except for JMP, however, indirect addressing *without* indexing is not provided. Since indirect addressing seems to be more useful without indexing than with it, many people have erroneously regarded this as a design flaw.

In fact, anything you can do with indirect addressing without indexing can be done, and done faster, with post-indexed (Y-register) indirect addressing on the 6502. We shall now give several illustrations of this.

Suppose we have two zero-page locations called IA and IA+1 for holding an indirect address. We can initialize these by setting them to AD and AD+1, as follows:

```
LDY  AD
STY  IA
LDY  AD+1
STY  IA+1
```

At this point we may do a JMP (IA), but not a STA (IA) (for example), since that instruction does not exist on the 6502. It may be simulated, of course, by loading the Y register with zero, so that the full sequence becomes

```
LDY  AD
STY  IA
LDY  AD+1
STY  IA+1
LDY  #0
STA  (IA),Y
```

But this is unnecessary. All we have to do is keep IA (the low-order address) set to zero at all times, and write

```
LDY  AD+1
STY  IA+1
LDY  AD
STA  (IA),Y
```

which is a *decrease* in the number of instructions (from 5 to 4), rather than an *increase* (from 5 to 6).

Why does this work? Suppose that the indirect address is 2345. Then AD contains 45 (hex) and AD+1 contains 23. Instead of putting 45 in IA and 23 in IA+1 (giving an indirect address of 2345, without indexing), we put *zero* in IA and 23 in IA+1, giving an indirect address of 2300. To this we apply indexing, with 45 (hex) in the Y register; 2300 plus 45 makes 2345.

A common use of this capability is in processing arrays of strings. Suppose we have an array T of n strings, with $n \leqslant 128$. This is implemented as an array of n addresses, with T($2k-1$) and T($2k$) containing, respectively, the low-order and high-order parts of the address of the first character of the k-th string in the string array T, for $1 \leqslant k \leqslant n$. To get the first character of the k-th string, we perform

```
LDA  K
ASL
TAY
LDA  T-1,Y
STA  IA+1
LDA  T-2,Y
TAY
LDA  (IA),Y
```

This assumes that T(1) is contained at T, T(2) at T+1, etc., so that T($2k-1$) is kept at T+$2k$-2 (= (T-2)+$2k$), and that IA contains the constant zero as before. This time the saving is not quite as great because we cannot do an LDY T-2,Y directly (we could not even do an LDY T-2,X unless T is in page zero). But TAY is certainly both shorter and faster than STA IA (which is what we would have done if indirect addressing without indexing had been available). (Note: ASL, in the above code, shifts the A register left by one, thus multiplying it by 2. On some assemblers one writes ASL A instead of simply ASL for this.)

The same sort of savings accrue if T is a parallel array rather than a serial array. That is, suppose we keep two arrays, T1 and T2, where all the low-order bytes are in T1 and all the high-order bytes are in T2. That is, what was in T($2k-1$) and T($2k$), respectively, is now in T1(k) and T2(k). There is now no need to multiply k by 2, and, perhaps more importantly, we can have $n \leqslant 256$ rather than $n \leqslant 128$ as in the preceding example. (The reason in both cases, of course, is that our indices into the arrays T, T1, and T2 must, in each case, fit into an eight-bit register, and must therefore be between 0 and 255, inclusive. Arrays with this property may be referred to as *short arrays*.) The code is now

```
LDY  K
LDA  T2-1,Y
STA  IA+1
LDA  T1-1,Y
TAY
LDA  (IA),Y
```

under the same assumptions as before.

In either case, incrementing the calculated indirect address is also speeded up. Incrementing the 16-bit quantity in IA and IA+1, if we needed that, would have been performed by

```
     INC  IA
     BNE  NXT
     INC  IA+1
NXT  (next instruction)
```

In the actual case, the incrementation is performed by

```
     INY
     BNE  NXT2
     INC  IA+1
NXT2 (next instruction)
```

which is again both shorter and faster.

It is true that, in this case, if we know that none of our strings has length greater than 256, we can do even better by using straightforward post-indexing rather than simulated non-indexing. The setup takes a little longer—we have to write something like

```
LDA  K
ASL
TAY
LDA  T-1,Y
```

```
STA  IA2+1
LDA  T-2,Y
STA  IA2
LDY  #0
LDA  (IA2),Y
```

—but now a simple INY suffices for incrementing the indirect address. Note that we carefully set up two further page-zero locations here, IA2 and IA2+1, because IA2, unlike IA, is *not* kept at the constant value of zero.

Another, even more common, application of these ideas is in indexing *long arrays* (this is, arrays of more than 256 bytes). Suppose that U is such an array and we wish to index U(J), where J has just been calculated. Here both U and J are 16-bit quantities. Assuming that #U and /U represent, respectively, the (constant) lower half and upper half of the address of U, we can add U and J and store the result at IA2 by the following code:

```
CLC
LDA  #U
ADC  J
STA  IA2
LDA  /U
ADC  J+1
STA  IA2+1
```

and then, if we had indirect addressing without indexing, we would be ready to apply it. As it is, all we have to write is

```
CLC
LDA  #U
ADC  J
TAY
LDA  /U
ADC  J+1
STA  IA+1
```

with IA set permanently to zero as before, and we are ready to use an instruction like LDA (IA),Y (again, this is both shorter and faster than what preceded it).

If we can set aside another two bytes in page zero for use only with *this particular array* U, we can do even better than that. Let us call them IA3 and IA3+1; the permanent contents of IA3 are the low-order byte of the address of U, rather than zero. This can be set up with a declaration like

```
IA3 ADR U
```

which actually sets up *both* IA3 and IA3+1, although only IA3 will be needed. We can now save two instructions each by writing

```
CLC
LDA  /U
ADC  J+1
STA  IA3+1
LDY  J
```

and now LDA (IA3),Y will load U(J) into the A register. Why does that work? Let us make the following abbreviations:

- p – high-order byte of address of U (i.e., /U)
- q – low-order byte of address of U (i.e., #U)
- r – high-order byte of J (contents of location J+1)
- s – low-order byte of J (contents of location J)

The 16-bit quantities we are adding are $256p+q$ and $256r+s$. The two bytes IA3 and IA3+1 contain, respectively, q and $p+r$, so that this 16-bit quantity is $256(p+r)+q$. To this we add s by post-indexing, so that the computed address is actually $(256(p+r)+q)+s = (256p+q)+(256r+s)$, or J plus the address of U.

The above may be extended immediately to arrays U of two-byte or four-byte quantities. A curious intermittent error may arise, however, if three-byte quantitites are used. We would, of course, multiply the 16-bit quantity in J and J+1 by 3 before using it, and then we would increment Y by one (using INY) to get the second byte set up, and again to get the third byte set up. This works fine, 99% of the time. However, if J is 85, so that 3J is 255, then incrementing Y by one is not enough, since there is carry into the high-order byte of 3J. We must test for nonzero after the increment, and then increment IA3+1 otherwise. (Similar problems arise for J = 170, 341, 426, etc.) It should not be hard to see that there will never be carry of this kind for an array of n-byte quantities where n is 2, 4, 8, or in general any power of 2.

In the above case it was assumed that U(0), rather than U(1), is contained at U; if this is not the case, then the address of U minus 1 must be substituted, in the above, for the address of U. We assume throughout, of course, that all 16-bit quantities are kept with the low-order byte first, as is customary on the 6502. The concepts of serial and parallel array are discussed further in [1]. Finally, users of the LISA assembler for the 6502 should substitute $0, $1, and $2 for 0, 1, and 2 respectively, throughout the coding examples above.

Reference: Maurer, W.D., *Programming*, Holden-Day, 1972, pp. 74-75.

About himself, Prof. Maurer has this to say:

I've worked for CSC, Lockheed, Project MAC, UC Berkeley (as an Assistant Professor) and The George Washington University, Washington, D.C. (as an Associate Professor and now a full Professor). I've written a book called Programming *as well as* The Programmer's Introduction to LISP *and* The Programmer's Introduction to SNOBOL. *In my spare time I write songs (My Pet Rock Is In Love, Dracula's Mood Ring, etc.) and have them performed by a group of weird people called The Hexagon Club.*

LISTING

```
]PRINT ""
!LOAD PPR5
BLOAD  PPR5, A$2000
OX!LIST
1                ORG $18
2                ADR U
3                ORG $0800
4                LDA #L2
5                STA AD
6                LDA /L2
7                STA AD+$1
8                LDA #$0
9                STA P
10               LDY AD
11               STY IA
12               LDY AD+$1
13               STY IA+$1
14               JMP (IA)
15       L2      LDA #P
16               STA AD
17               LDA /P
18               STA AD+$1
19               LDY AD
20               STY IA
21               LDY AD+$1
22               STY IA+$1
23               LDY #$0
24               LDA #$37
25               STA (IA),Y
26               JSR TS1
27               LDA #$0
28               STA IA
29               LDY AD+$1
30               STY IA+$1
31               LDY AD
32               LDA #$37
33               STA (IA),Y
34               JSR TS1
35               LDA #$11
36               STA K
37               LDA #P
38               STA T+$20
39               STA T1+$10
40               LDA /P
41               STA T+$21
42               STA T2+$10
43               LDA K
44               ASL
45               TAY
46               LDA T-$1,Y
47               STA IA+$1
48               LDA T-$2,Y
49               TAY
50               LDA (IA),Y
51               CMP #$0
52               BEQ L4
53               BRK
54       L4      LDA #$15
55               STA P
56               LDY K
57               LDA T2-$1,Y
58               STA IA+$1
59               LDA T1-$1,Y
60               TAY
61               LDA (IA),Y
```

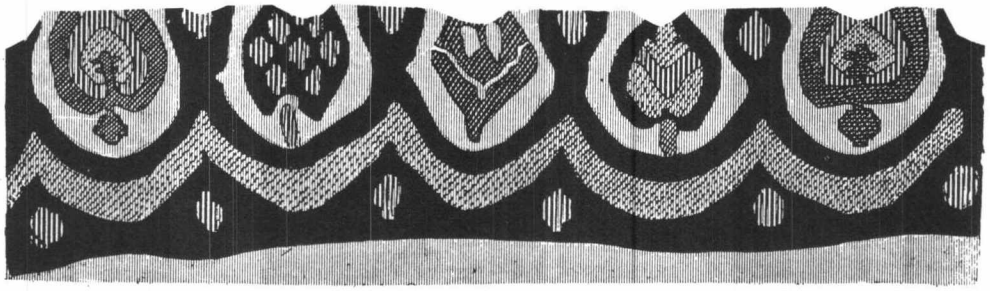
```
62              CMP #$15
63              BEQ L5
64              BRK
65      L5      LDA #$53
66              STA P+$1
67              LDX IA
68              LDA #P
69              STA IA
70              LDA /P
71              STA IA+$1
72              INC IA
73              BNE NXT
74              INC IA+$1
75      NXT     LDY #$0
76              LDA (IA),Y
77              CMP #$53
78              BEQ L6
79              BRK
80      L6      STX IA
81              LDY #P
82              INY
83              BNE NXT2
84              INC IA+$1
85      NXT2    LDA (IA),Y
86              CMP #$53
87              BNE BK
88              LDA #$7
89              STA P
90              LDA #$9
91              STA P2
92              LDA K
93              ASL
94              TAY
95              LDA T-$1,Y
96              STA IA2+$1
97              LDA T-$2,Y
98              STA IA2
99              LDY #$0
100             LDA (IA2),Y
101             CMP #$7
102             BNE BK
103             INY
104             LDA (IA2),Y
105             CMP #$9
106             BNE BK
107             LDA #$51
108             STA J
109             LDA #$3
110             STA J+$1
111             LDA #$14
112             STA U+$351
113             CLC
114             LDA #U
115             ADC J
116             STA IA2
117             LDA /U
118             ADC J+$1
119             STA IA2+$1
120             LDY #$0
121             LDA (IA2),Y
122             CMP #$14
123             BNE BK
124             CLC
125             LDA #U
126             ADC J
127             TAY
128             LDA /U
129             ADC J+$1
130             STA IA+$1
131             LDA (IA),Y
132             CMP #$14
133             BNE BK
134             CLC
135             LDA /U
136             ADC J+$1
137             STA IA3+$1
138             LDY J
139             LDA (IA3),Y
140             CMP #$14
141             BNE BK
142     EN      BRK
143     BK      BRK
144     P       ORG *+$1
145     P2      ORG *+$1
146     TS1     LDA #$37
147             CMP P
148             BNE BK
149             LDA #$0
150             STA P
151             RTS
152     IA3     EPZ $18
153     IA      EPZ $1A
154     IA2     EPZ $1C
155     AD      ADR L2
156     K       ORG *+$1
157     T       ORG *+$50
158     T1      ORG *+$50
159     T2      ORG *+$50
160     U       ORG *+$500
161     J       ORG *+$2
162             END
!ASM
**END OF PASS 1
**END OF PASS 2
0018            1               ORG $18
0018 320A       2               ADR U
0800            3               ORG $0800
0800 A91C       4               LDA #L2
0802 8D3F09     5               STA AD
0805 A908       6               LDA /L2
0807 8D4009     7               STA AD+$1
080A A900       8               LDA #$0
080C 8D3009     9               STA P
080F AC3F09     10              LDY AD
0812 841A       11              STY IA
0814 AC4009     12              LDY AD+$1
0817 841B       13              STY IA+$1
0819 6C1A00     14              JMP (IA)
081C A930       15      L2      LDA #P
081E 8D3F09     16              STA AD
0821 A909       17              LDA /P
0823 8D4009     18              STA AD+$1
0826 AC3F09     19              LDY AD
0829 841A       20              STY IA
082B AC4009     21              LDY AD+$1
082E 841B       22              STY IA+$1
0830 A000       23              LDY #$0
0832 A937       24              LDA #$37
0834 911A       25              STA (IA),Y
0836 203209     26              JSR TS1
0839 A900       27              LDA #$0
083B 851A       28              STA IA
083D AC4009     29              LDY AD+$1
0840 841B       30              STY IA+$1
0842 AC3F09     31              LDY AD
0845 A937       32              LDA #$37
0847 911A       33              STA (IA),Y
0849 203209     34              JSR TS1
084C A911       35              LDA #$11
084E 8D4109     36              STA K
0851 A930       37              LDA #P
0853 8D6209     38              STA T+$20
0856 8DA209     39              STA T1+$10
0859 A909       40              LDA /P
085B 8D6309     41              STA T+$21
085E 8DF209     42              STA T2+$10
0861 AD4109     43              LDA K
0864 0A         44              ASL
0865 A8         45              TAY
0866 B94109     46              LDA T-$1,Y
0869 851B       47              STA IA+$1
086B B94009     48              LDA T-$2,Y
086E A8         49              TAY
086F B11A       50              LDA (IA),Y
0871 C900       51              CMP #$0
0873 F001       52              BEQ L4
0875 00         53              BRK
0876 A915       54      L4      LDA #$15
0878 8D3009     55              STA P
087B AC4109     56              LDY K
087E B9E109     57              LDA T2-$1,Y
0881 851B       58              STA IA+$1
0883 B99109     59              LDA T1-$1,Y
0886 A8         60              TAY
0887 B11A       61              LDA (IA),Y
0889 C915       62              CMP #$15
088B F001       63              BEQ L5
088D 00         64              BRK
088E A953       65      L5      LDA #$53
0890 8D3109     66              STA P+$1
0893 A61A       67              LDX IA
```

```
0895 A930    68          LDA #P
0897 851A    69          STA IA
0899 A909    70          LDA /P
089B 851B    71          STA IA+$1
089D E61A    72          INC IA
089F D002    73          BNE NXT
08A1 E61B    74          INC IA+$1
08A3 A000    75  NXT     LDY #$0
08A5 B11A    76          LDA (IA),Y
08A7 C953    77          CMP #$53
08A9 F001    78          BEQ L6
08AB 00      79          BRK
08AC 861A    80  L6      STX IA
08AE A030    81          LDY #P
08B0 C8      82          INY
08B1 D002    83          BNE NXT2
08B3 E61B    84          INC IA+$1
08B5 B11A    85  NXT2    LDA (IA),Y
08B7 C953    86          CMP #$53
08B9 D074    87          BNE BK
08BB A907    88          LDA #$7
08BD 8D3009  89          STA P
08C0 A909    90          LDA #$9
08C2 8D3109  91          STA P2
08C5 AD4109  92          LDA K
08C8 0A      93          ASL
08C9 A8      94          TAY
08CA B94109  95          LDA T-$1,Y
08CD 851D    96          STA IA2+$1
08CF B94009  97          LDA T-$2,Y
08D2 851C    98          STA IA2
08D4 A000    99          LDY #$0
08D6 B11C   100          LDA (IA2),Y
08D8 C907   101          CMP #$7
08DA D053   102          BNE BK
08DC C8     103          INY
08DD B11C   104          LDA (IA2),Y
08DF C909   105          CMP #$9
08E1 D04C   106          BNE BK
08E3 A951   107          LDA #$51
08E5 8D320F 108          STA J
08E8 A903   109          LDA #$3
08EA 8D330F 110          STA J+$1
08ED A914   111          LDA #$14
08EF 8D830D 112          STA U+$351
08F2 18     113          CLC
08F3 A932   114          LDA #U
08F5 6D320F 115          ADC J
08F8 851C   116          STA IA2
08FA A90A   117          LDA /U
08FC 6D330F 118          ADC J+$1
08FF 851D   119          STA IA2+$1
0901 A000   120          LDY #$0
0903 B11C   121          LDA (IA2),Y
0905 C914   122          CMP #$14
0907 D026   123          BNE BK
0909 18     124          CLC
090A A932   125          LDA #U
090C 6D320F 126          ADC J
090F A8     127          TAY
0910 A90A   128          LDA /U
0912 6D330F 129          ADC J+$1
0915 851B   130          STA IA+$1
0917 B11A   131          LDA (IA),Y
0919 C914   132          CMP #$14
091B D012   133          BNE BK
091D 18     134          CLC
091E A90A   135          LDA /U
0920 6D330F 136          ADC J+$1
0923 8519   137          STA IA3+$1
0925 AC320F 138          LDY J
0928 B118   139          LDA (IA3),Y
092A C914   140          CMP #$14
092C D001   141          BNE BK
092E 00     142  EN      BRK
092F 00     143  BK      BRK
0931        144  P       ORG *+$1
0932        145  P2      ORG *+$1
0932 A937   146  TS1     LDA #$37
0934 CD3009 147          CMP P
0937 D0F6   148          BNE BK
0939 A900   149          LDA #$0
093B 8D3009 150          STA P
093E 60     151          RTS
093F        152  IA3     EPZ $18
093F        153  IA      EPZ $1A
093F        154  IA2     EPZ $1C
093F 1C08   155  AD      ADR L2
0942        156  K       ORG *+$1
0992        157  T       ORG *+$50
09E2        158  T1      ORG *+$50
0A32        159  T2      ORG *+$50
0F32        160  U       ORG *+$500
0F34        161  J       ORG *+$2
            162          END
***** END OF ASSEMBLY


          *************************
          *                       *
          * SYMBOL TABLE -- V 1 0 *
          *                       *
          *************************
LABEL.  LOC.   LABEL.  LOC.   LABEL.  LOC.
IA3     0018   IA      001A   IA2     001C
L2      081C   L4      0876   L5      088E
NXT     08A3   L6      08AC   NXT2    08B5
EN      092E   BK      092F   P       0930
P2      0931   TS1     0932   AD      093F
K       0941   T       0942   T1      0992
T2      09E2   U       0A32   J       0F32
!
```

```
*0800T
0800-   A9 1C       LDA   #$1C
 A=1C X=00 Y=51 P=31 S=E3
0802-   8D 3F 09    STA   $093F
 A=1C X=00 Y=51 P=31 S=E3
0805-   A9 08       LDA   #$08
 A=08 X=00 Y=51 P=31 S=E3
0807-   8D 40 09    STA   $0940
 A=08 X=00 Y=51 P=31 S=E3
080A-   A9 00       LDA   #$00
 A=00 X=00 Y=51 P=33 S=E3
080C-   8D 30 09    STA   $0930
 A=00 X=00 Y=51 P=33 S=E3
080F-   AC 3F 09    LDY   $093F
 A=00 X=00 Y=1C P=31 S=E3
0812-   84 1A       STY   $1A
 A=00 X=00 Y=1C P=31 S=E3
0814-   AC 40 09    LDY   $0940
 A=00 X=00 Y=08 P=31 S=E3
0817-   84 1B       STY   $1B
 A=00 X=00 Y=08 P=31 S=E3
0819-   6C 1A 00    JMP   ($001A)
 A=00 X=00 Y=08 P=31 S=E3
081C-   A9 30       LDA   #$30
 A=30 X=00 Y=08 P=31 S=E3
081E-   8D 3F 09    STA   $093F
 A=30 X=00 Y=08 P=31 S=E3
0821-   A9 09       LDA   #$09
 A=09 X=00 Y=08 P=31 S=E3
0823-   8D 40 09    STA   $0940
 A=09 X=00 Y=08 P=31 S=E3
0826-   AC 3F 09    LDY   $093F
 A=09 X=00 Y=30 P=31 S=E3
0829-   84 1A       STY   $1A
 A=09 X=00 Y=30 P=31 S=E3
082B-   AC 40 09    LDY   $0940
 A=09 X=00 Y=09 P=31 S=E3
082E-   84 1B       STY   $1B
 A=09 X=00 Y=09 P=31 S=E3
0830-   A0 00       LDY   #$00
 A=09 X=00 Y=00 P=33 S=E3
0832-   A9 37       LDA   #$37
 A=37 X=00 Y=00 P=31 S=E3
0834-   91 1A       STA   ($1A),Y
 A=37 X=00 Y=00 P=31 S=E3
0836-   20 32 09    JSR   $0932
 A=37 X=00 Y=00 P=31 S=E3
0932-   A9 37       LDA   #$37
 A=37 X=00 Y=00 P=31 S=E1
0934-   CD 30 09    CMP   $0930
 A=37 X=00 Y=00 P=33 S=E1
0937-   D0 F6       BNE   $092F
 A=37 X=00 Y=00 P=33 S=E1
0939-   A9 00       LDA   #$00
 A=00 X=00 Y=00 P=33 S=E1
```

```
093B-   8D 30 09    STA   $0930
 A=00 X=00 Y=00 P=33 S=E1
093E-   60           RTS
 A=00 X=00 Y=00 P=33 S=E1
0839-   A9 00        LDA   #$00
 A=00 X=00 Y=00 P=33 S=E3
083B-   85 1A        STA   $1A
 A=00 X=00 Y=00 P=33 S=E3
083D-   AC 40 09     LDY   $0940
 A=00 X=00 Y=09 P=31 S=E3
0840-   84 1B        STY   $1B
 A=00 X=00 Y=09 P=31 S=E3
0842-   AC 3F 09     LDY   $093F
 A=00 X=00 Y=30 P=31 S=E3
0845-   A9 37        LDA   #$37
 A=37 X=00 Y=30 P=31 S=E3
0847-   91 1A        STA   ($1A),Y
 A=37 X=00 Y=30 P=31 S=E3
0849-   20 32 09     JSR   $0932
 A=37 X=00 Y=30 P=31 S=E3
0932-   A9 37        LDA   #$37
 A=37 X=00 Y=30 P=31 S=E1
0934-   CD 30 09     CMP   $0930
 A=37 X=00 Y=30 P=33 S=E1
0937-   D0 F6        BNE   $092F
 A=37 X=00 Y=30 P=33 S=E1
0939-   A9 00        LDA   #$00
 A=00 X=00 Y=30 P=33 S=E1
093B-   8D 30 09     STA   $0930
 A=00 X=00 Y=30 P=33 S=E1
093E-   60           RTS
 A=00 X=00 Y=30 P=33 S=E1
084C-   A9 11        LDA   #$11
 A=11 X=00 Y=30 P=31 S=E3
084E-   8D 41 09     STA   $0941
 A=11 X=00 Y=30 P=31 S=E3
0851-   A9 30        LDA   #$30
 A=30 X=00 Y=30 P=31 S=E3
0853-   8D 62 09     STA   $0962
 A=30 X=00 Y=30 P=31 S=E3
0856-   8D A2 09     STA   $09A2
 A=30 X=00 Y=30 P=31 S=E3
0859-   A9 09        LDA   #$09
 A=09 X=00 Y=30 P=31 S=E3
085B-   8D 63 09     STA   $0963
 A=09 X=00 Y=30 P=31 S=E3
085E-   8D F2 09     STA   $09F2
 A=09 X=00 Y=30 P=31 S=E3
0861-   AD 41 09     LDA   $0941
 A=11 X=00 Y=30 P=31 S=E3
0864-   0A           ASL
 A=22 X=00 Y=30 P=30 S=E3
0865-   A8           TAY
 A=22 X=00 Y=22 P=30 S=E3
0866-   B9 41 09     LDA   $0941,Y
 A=09 X=00 Y=22 P=30 S=E3
0869-   85 1B        STA   $1B
 A=09 X=00 Y=22 P=30 S=E3
086B-   B9 40 09     LDA   $0940,Y
 A=30 X=00 Y=22 P=30 S=E3
086E-   A8           TAY
 A=30 X=00 Y=30 P=30 S=E3
086F-   B1 1A        LDA   ($1A),Y
 A=00 X=00 Y=30 P=32 S=E3
0871-   C9 00        CMP   #$00
 A=00 X=00 Y=30 P=33 S=E3
0873-   F0 01        BEQ   $0876
 A=00 X=00 Y=30 P=33 S=E3
0876-   A9 15        LDA   #$15
 A=15 X=00 Y=30 P=31 S=E3
0878-   8D 30 09     STA   $0930
 A=15 X=00 Y=30 P=31 S=E3
087B-   AC 41 09     LDY   $0941
 A=15 X=00 Y=11 P=31 S=E3
087E-   B9 E1 09     LDA   $09E1,Y
 A=09 X=00 Y=11 P=31 S=E3
0881-   85 1B        STA   $1B
 A=09 X=00 Y=11 P=31 S=E3
0883-   B9 91 09     LDA   $0991,Y
 A=30 X=00 Y=11 P=31 S=E3
0886-   A8           TAY
 A=30 X=00 Y=30 P=31 S=E3
0887-   B1 1A        LDA   ($1A),Y
 A=15 X=00 Y=30 P=31 S=E3
0889-   C9 15        CMP   #$15
 A=15 X=00 Y=30 P=33 S=E3
088B-   F0 01        BEQ   $088E
 A=15 X=00 Y=30 P=33 S=E3
088E-   A9 53        LDA   #$53
 A=53 X=00 Y=30 P=31 S=E3
0890-   8D 31 09     STA   $0931
 A=53 X=00 Y=30 P=31 S=E3
0893-   A6 1A        LDX   $1A
 A=53 X=00 Y=30 P=33 S=E3
0895-   A9 30        LDA   #$30
 A=30 X=00 Y=30 P=31 S=E3
0897-   85 1A        STA   $1A
 A=30 X=00 Y=30 P=31 S=E3
0899-   A9 09        LDA   #$09
 A=09 X=00 Y=30 P=31 S=E3
089B-   85 1B        STA   $1B
 A=09 X=00 Y=30 P=31 S=E3
089D-   E6 1A        INC   $1A
 A=09 X=00 Y=30 P=31 S=E3
089F-   D0 02        BNE   $08A3
 A=09 X=00 Y=30 P=31 S=E3
08A3-   A0 00        LDY   #$00
 A=09 X=00 Y=00 P=33 S=E3
08A5-   B1 1A        LDA   ($1A),Y
 A=53 X=00 Y=00 P=31 S=E3
08A7-   C9 53        CMP   #$53
 A=53 X=00 Y=00 P=33 S=E3
08A9-   F0 01        BEQ   $08AC
 A=53 X=00 Y=00 P=33 S=E3
08AC-   86 1A        STX   $1A
 A=53 X=00 Y=00 P=33 S=E3
08AE-   A0 30        LDY   #$30
 A=53 X=00 Y=30 P=31 S=E3
08B0-   C8           INY
 A=53 X=00 Y=31 P=31 S=E3
08B1-   D0 02        BNE   $08B5
 A=53 X=00 Y=31 P=31 S=E3
08B5-   B1 1A        LDA   ($1A),Y
 A=53 X=00 Y=31 P=31 S=E3
08B7-   C9 53        CMP   #$53
 A=53 X=00 Y=31 P=33 S=E3
08B9-   D0 74        BNE   $092F
 A=53 X=00 Y=31 P=33 S=E3
08BB-   A9 07        LDA   #$07
 A=07 X=00 Y=31 P=31 S=E3
08BD-   8D 30 09     STA   $0930
 A=07 X=00 Y=31 P=31 S=E3
08C0-   A9 09        LDA   #$09
 A=09 X=00 Y=31 P=31 S=E3
08C2-   8D 31 09     STA   $0931
 A=09 X=00 Y=31 P=31 S=E3
08C5-   AD 41 09     LDA   $0941
 A=11 X=00 Y=31 P=31 S=E3
08C8-   0A           ASL
 A=22 X=00 Y=31 P=30 S=E3
08C9-   A8           TAY
 A=22 X=00 Y=22 P=30 S=E3
08CA-   B9 41 09     LDA   $0941,Y
 A=09 X=00 Y=22 P=30 S=E3
08CD-   85 1D        STA   $1D
 A=09 X=00 Y=22 P=30 S=E3
08CF-   B9 40 09     LDA   $0940,Y
 A=30 X=00 Y=22 P=30 S=E3
08D2-   85 1C        STA   $1C
 A=30 X=00 Y=22 P=30 S=E3
08D4-   A0 00        LDY   #$00
 A=30 X=00 Y=00 P=32 S=E3
08D6-   B1 1C        LDA   ($1C),Y
 A=07 X=00 Y=00 P=30 S=E3
08D8-   C9 07        CMP   #$07
 A=07 X=00 Y=00 P=33 S=E3
08DA-   D0 53        BNE   $092F
 A=07 X=00 Y=00 P=33 S=E3
08DC-   C8           INY
 A=07 X=00 Y=01 P=31 S=E3
08DD-   B1 1C        LDA   ($1C),Y
 A=09 X=00 Y=01 P=31 S=E3
08DF-   C9 09        CMP   #$09
 A=09 X=00 Y=01 P=33 S=E3
08E1-   D0 4C        BNE   $092F
 A=09 X=00 Y=01 P=33 S=E3
08E3-   A9 51        LDA   #$51
 A=51 X=00 Y=01 P=31 S=E3
```

```
08E5-   8D 32 0F     STA   $0F32
 A=51 X=00 Y=01 P=31 S=E3
08E8-   A9 03        LDA   #$03
 A=03 X=00 Y=01 P=31 S=E3
08EA-   8D 33 0F     STA   $0F33
 A=03 X=00 Y=01 P=31 S=E3
08ED-   A9 14        LDA   #$14
 A=14 X=00 Y=01 P=31 S=E3
08EF-   8D 83 0D     STA   $0D83
 A=14 X=00 Y=01 P=31 S=E3
08F2-   18           CLC
 A=14 X=00 Y=01 P=30 S=E3
08F3-   A9 32        LDA   #$32
 A=32 X=00 Y=01 P=30 S=E3
08F5-   6D 32 0F     ADC   $0F32
 A=83 X=00 Y=01 P=F0 S=E3
08F8-   85 1C        STA   $1C
 A=83 X=00 Y=01 P=F0 S=E3
08FA-   A9 0A        LDA   #$0A
 A=0A X=00 Y=01 P=70 S=E3
08FC-   6D 33 0F     ADC   $0F33
 A=0D X=00 Y=01 P=30 S=E3
08FF-   85 1D        STA   $1D
 A=0D X=00 Y=01 P=30 S=E3
0901-   A0 00        LDY   #$00
 A=0D X=00 Y=00 P=32 S=E3
0903-   B1 1C        LDA   ($1C),Y
 A=14 X=00 Y=00 P=30 S=E3
0905-   C9 14        CMP   #$14
 A=14 X=00 Y=00 P=33 S=E3
0907-   D0 26        BNE   $092F
 A=14 X=00 Y=00 P=33 S=E3
0909-   18           CLC
 A=14 X=00 Y=00 P=32 S=E3
090A-   A9 32        LDA   #$32
 A=32 X=00 Y=00 P=30 S=E3
090C-   6D 32 0F     ADC   $0F32
 A=83 X=00 Y=00 P=F0 S=E3
090F-   A8           TAY
 A=83 X=00 Y=83 P=F0 S=E3
0910-   A9 0A        LDA   #$0A
 A=0A X=00 Y=83 P=70 S=E3
0912-   6D 33 0F     ADC   $0F33
 A=0D X=00 Y=83 P=30 S=E3
0915-   85 1B        STA   $1B
 A=0D X=00 Y=83 P=30 S=E3
0917-   B1 1A        LDA   ($1A),Y
 A=14 X=00 Y=83 P=30 S=E3
0919-   C9 14        CMP   #$14
 A=14 X=00 Y=83 P=33 S=E3
091B-   D0 12        BNE   $092F
 A=14 X=00 Y=83 P=33 S=E3
091D-   18           CLC
 A=14 X=00 Y=83 P=32 S=E3
091E-   A9 0A        LDA   #$0A
 A=0A X=00 Y=83 P=30 S=E3
0920-   6D 33 0F     ADC   $0F33
 A=0D X=00 Y=83 P=30 S=E3
0923-   85 19        STA   $19
 A=0D X=00 Y=83 P=30 S=E3
0925-   AC 32 0F     LDY   $0F32
 A=0D X=00 Y=51 P=30 S=E3
0928-   B1 18        LDA   ($18),Y
 A=14 X=00 Y=51 P=30 S=E3
092A-   C9 14        CMP   #$14
 A=14 X=00 Y=51 P=33 S=E3
092C-   D0 01        BNE   $092F
 A=14 X=00 Y=51 P=33 S=E3
092E-   00           BRK
092E-     A=14 X=00 Y=51 P=33 S=E3
*0800G
0930-     A=14 X=00 Y=51 P=33 S=DF
```



Northstar X-Ref Tool

BY JOHN HOLTRICHTER, JR.
P. O. Box 227
Redondo Beach, CA 90277

Dear *Dr. Dobb's*:

The Northstar BASIC Cross-Reference Tool program written by Titus Purdin and published in *DDJ* # 44 is not designed to operate in a double density system.

Since double density is now becoming very popular, I thought your readers might like to make the changes indicated at the end of this letter in order to implement the reading, in a double density system, of single density type 2 files on single density diskettes, and both single and double density files on double density diskettes.

First of all, the routine should be organized at 2D00H, and if it is wished to be able to leave the program after bringing it up but before naming a type 2 file, a CPI X and JZ EXIT should be added to the INFILE routine, where X may be 03H for a Control-C or 1BH if an Escape is desired in order to return immediately to DOS.

Since double density systems may use up to four drives under versions 5.0 and later, a CPI 34H and JZ GETCR should be added to the INUNT routine.

Next, the fact that, in the directory sectors of the diskettes, the type of file is listed with the 7th bit high in double density and low in single density formats, the CKTYP routine should be modified and an additional single byte storage location, DENS, must be defined. In this modified routine, the type of file is first checked to see if it is 02H (single density type 2 file) or 82H (double density type 2 file). In either case, only the highest bit is saved in the DENS location, where 00H = single density and 80H = double density.

The next problem to overcome is the fact that in double density format, the reading of a file using the DCOM routine at 2022H in DOS provides 512 bytes to a designated RAM address, while only 256 bytes is obtained in a single density access of this routine. Accordingly, the DATA storage location should be increased in size to 512 in order to accommodate a double density dump.

Now the routine which reads the DATA area in RAM for processing must be modified in accordance with the density of the type 2 file to be investigated so that no more and no less than the length of the data supplied with each DCOM access is examined. The modification is made to the RDBL1 subroutine in order to determine the density of the file, and then to load into an expanded BLKPT counter storage location (from 1 byte to 2 bytes long), 100H (256 bytes) for single density or 200H (512 bytes) for double density, in this initialization procedure. Then, the Check Block Routine, NXTCR is modified to load this 2 byte counter value from BLKPT into the H&L registers, decrement it, and then to read the next block from the diskette file into RAM, only if the counter has reached zero.

In the following partial listing, some presently existing code is shown to aid in locating where changes and additions are to be made.

```
;
;       ORG     2D00H
;
;
INFIL:  CALL    CIN
        STA     HOLDT
        CPI     3               ;CONTROL-C ?
        JZ      EXIT
        CPI     0DH
        JZ      INUNT
                ...
;               ...
;
; ACCEPTS DISK UNIT NUMBER.  CHECKS TO MAKE SURE INPUT IS ASCII 1,
; 2, 3 OR 4.
;
INUNT:  MVI     C,12
                ...
        CPI     32H
        JZ      GETCR
        CPI     33H             ;DISK DRIVE 3 ?
        JZ      GETCR
        CPI     34H             ;DISK DRIVE 4 ?
        JZ      GETCR
;               ...
;               ...
CKKTYP: MVI     B,0
        MVI     C,4
        DAD     B
        MVI     A,2             ;SINGLE DENSITY CHECK
        CMP     M
        JZ      CKTY1
        MVI     A,82H           ;DOUBLE DENSITY CHECK
        CMP     M
        JZ      CKTY1
        MVI     C,18
        LXI     H,HDR5
        CALL    PRINT
        CALL    CRLF
        JMP     ASKFL
CKTY1:  ANI     80H             ;STRIP ALL BUT BIT 7
        STA     DENS            ;STORE DENSITY CODE
;               ...
;
; THIS ROUTINE READS ONE 256 BYTE BLOCK OF DATA FROM A SINGLE
; DENSITY DISK FILE OR 512 BYTES FROM A DOUBLE DENSITY FILE
; INTO THE AREA CALLED 'DATA'.
;
RDBLK:  LXI     H,DRIVE
        MOV     B,M             ;GET DRIVE # BYTE
        LDA     DENS            ;LOAD DENSITY CODE
        ORA     B               ;TACK ON CODE TO DRIVE # BYTE
```

Jn Double Density Systems

```
                                                    C,A
                                                    A,1
                                                    B,1
                                                    D,DATA
                                                    DIR
                                                    H
                                                    DIR
                                                    H
                                                    RDDSK
                                                    RDBL1
                                                    C,14
                                                    H,HDR10
                                                    PRINT
                                                    CRLF
                                                    EXIT
                                                    DENS
                                        ;DENSITY CHECK
                                                    A
                                                    RDBL2
                                        ;DOUBLE DENSITY COUNTER
                                                    H,200H
                                                    RDBL3
                                        ;SINGLE DENSITY COUNTER
                                                    H,100H
                                        ;STORE BLOCK SIZE COUNTER
                                                    BLKPT
                                                    D,DATA

                                                    ...

                                        ; CHECK BLOCK ROUTINE
                                        ;
                                                    D
                                        ;LOAD BLOCK SIZE COUNTER VALUE
                                                    BLKPT
                                        ;DECREMENT IT
                                                    H
                                        ;STORE NEW VALUE
                                                    BLKPT
                                                    A,H
                                                    L
                                        ;READ DISKETTE FILE IF VALUE = 00H
                                                    RDBLK
                                                    H,D
                                                    L,E
                                                    A,0DH
                                                    M
                                                    NXTC1
                                                    LNTAB

                                                    ...                 ...
                                        ;                               ...
                                        ;                   DS      1
                                        DRIVE:      DS      1
                                        DENS:       DS      512
                                        DATA:       DS      1
                                        LINCT:      DS                  ...

                                        POINT:      DS      2
                                        BLKPT:      DS      2
                                        LNTAB:      DS      1
                                                            ...
```

ADDENDUM

Dear Editor:

Thank you for your kind letter of April 21 concerning the publication of my changes to Titus Purdin's X-Ref program.

The reason for my previous comments was that I run both double and single density North Star and CP/M systems in my IMSAI 8080 (Z-80 CPU) and the changes were needed in order to allow me to best utilize this fine tool. However, I also run both 80 X 24 and 64 X 16 formated monitors (along with four 5¼ inch and four 8 inch drives) and I find that the X-Ref program was designed for only the 80 X 24 format, which makes it difficult to read the last of the eight columns on the smaller formated screen. I can also see where this format would cause problems with those who have printers that cannot accommodate 80 characters on a line. Accordingly, I have set forth below a simple solution which may be incorporated as a post script to my previous letter, if you wish.

In the subroutine labeled 'LSTN1:' the MVI A,8 should be changed to MVI A,7 for a 64 character line, and MVI A,4 for a 40 character line. The former change will provide seven groups of line numbers per line, while the latter will provide four groups per line. Nothing will be lost, but the columns will become longer. The purest may also wish to shorten the dedicated length of PLINE: and the number of blanks (20H) deposited therein, etc., but the saving in program length will be minimal.

```
LSTN1:  XRA   A
        STA   TAB
        MVI   A,7       ;7=64 CHAR'S/LINE: ;4=40 CHAR'S/LINE
        STA   LINCT
              ...
```

I have also noticed that the program, as it stands, will not work in systems which check the A register to see which input device is being used. An example of this is the standard North*Star Horizon system. A solution to this problem is to add MVI A,0 for the standard input device (video terminal) and a 1, 2 or 3 etc., for other input devices, immediately before a CALL CIN. The locations of these additions are as follows:

```
INFIL:  MVI   A,0       ; 0=STANDARD VIDEO TERMINAL
        CALL  CIN
              ...

INUNT:        ...
        LXI   H,DRIVE
        MVI   A,0       ; 0=STANDARD
        CALL  CIN
              ...

GETCR:  ANI   0FH
        MOV   M,A
        MVI   A,0       ; 0=STANDARD
        CALL  CIN
              ...
```

The same general procedure should be followed (placing device number in the A register) before calling COUT, if the output is to go to a device other than the video terminal. For example, a 1 would send the output to a line printer.

HARDCOPY DEVICE DRIVER PROGRAMS FOR CP/M

BY STEVE WILLOUGHBY
Owner, W.E.D.
Box 87
Crane, Indiana 47522
812-854-7434

The following programs were written to add a flexible hardcopy output to a CP/M system. These programs may be coresident (i.e. punch tape and print on tty and also print on daisywheel printer). The hardware consists of a processor with 32K bytes of contiguous low read/write memory, (RAM, with 4 K bytes of RAM at D000, a serial interface wherein RS-232 pin 20 (Data terminal ready) disables transmit buffer empty (TBMT) status to allow high speed transfers with buffered printers by handshaking through DTR. The software shown uses the same interface for both devices, so these routines would not logically be coresident without changing one interface assignment.

Each program consists of a relocator which uses the CP/M jump vector at 0 to locate the active B10S jump table, retrieves the current jump link addresses, punches them into the driver program, relocates it to its place of residence, alters the CP/M B10S jump table entries to point to new driver, performs initialization of device and driver, and reboots CP/M.

My system has aVDM and parallel keyboard for the system console. When the driver program is run (as any other CAM file), the device becomes the system printer and console output is intercepted and printed, or not, depending on the position of the associated sense switch. If additional drivers are run, the last one is the system printer, all others being lost; however, the console output print/no print is stacked up through each successive driver. The Diablo driver also has provisions to replace the parallel keyboard by the serial keyboard of the diablo printer..

Caution!!! If a driver program is run twice it will lock everything up; if run immediately after itself, it will return to itself after the output; if run at any other time (intervening drive(s) run) it overwrites the valid CONOUT return and forms a loop through the intervening drivers.

For those who don't have sense switches: write a CONIN intercept routine to intercept normal keyboard input. If a control P (CDOS convention) is detected, complement a location. This location would then be read instead of sense switches. The control P could be passed along or suppressed (not recommended.)

No longer does your console output scroll off the screen to oblivion, yet neither are you a slave to pulpmills and slow printers when hard copy is not needed. You have hardcopy of listings, DDP disassemblies, ED text while in editor, and system journal when you can't stay and watch things grind.

DEVICE PECULIARITIES: The teletypewriter is of the kind that has no tabs, no top of forms, and needs a delay after carriage return, and overprints at the end of the line.

This routine counts lines and, at the end of the page, prints a cut mark (–) between two blank lines. A count of characters per line is maintained and an automatic CR/LF is generated to prevent overprinting. Tabs are expanded to spaces and occur at positions exactly divisible by 8. A form feed mark routine is done. All other control characters are ignored. (Thrown away.)

The daisywheel printer must have its tab stops preset by a character stream. It recognizes form feed, and does not require delay after CR. The printer receives commands in the print stream with escape sequences, of which is added two to allow resetting the carriage and top of form (Esc – R/r) and reset the tabls (Esc – T/t).

The programs are highly commented and should be self-explanatory.

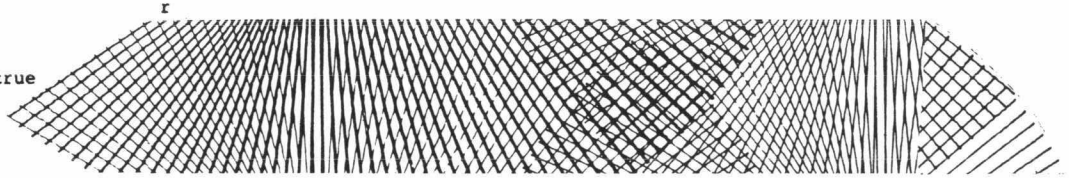
```
?

A>TYPE DIABLO.PRN

                ;       DIABLO 1650 I/O DRIVER FOR CP/M
                ;       Revision level   2.1    2-28-80
                ;
                ;       Steve Willoughby
                ;       Box 87
                ;       Crane,IN 47522
                ;
                ;
D400 =          DESTIN  EQU     0D400H          ;Driver destination address
0004 =          STATUS  EQU     04H             ;Status port
0080 =          MASK    EQU     80H             ;Printer ready bit
0040 =          INMASK  EQU     40H             ;Keyboard ready bit
0005 =          PORT    EQU     05H             ;Printer port
0000 =          SENSE   EQU     00H             ;Sense of data
0004 =          JSENSE  EQU     04H             ;Sense of status
0001 =          SENSWT  EQU     01H             ;Sense switch port
0002 =          SENMSK  EQU     02H             ;Printer on bit
0004 =          KEYMSK  EQU     04H             ;Keyboard substitute bit
0000 =          SENSEN  EQU     00H             ;Sense of switch register
0100            START   ORG     100H
0100 317F00             LXI     SP,7FH          ;Set up stack
0103 2A0100             LHLD    0001H           ;Get warm boot address
0106 110400             LXI     D,04H           ;Point to console status link
0109 19                 DAD     D
010A E5                 PUSH    H               ;Save constatus address
010B 7E                 MOV     A,M             ;Put it in program
010C 321B02             STA     CON1+1
010F 23                 INX     H
0110 7E                 MOV     A,M
0111 321C02             STA     CON1+2
0114 23                 INX     H               ;Step to conin link
0115 23                 INX     H
0116 7E                 MOV     A,M             ;Get conin link
0117 320702             STA     CON2+1
011A 23                 INX     H
011B 7E                 MOV     A,M
011C 320802             STA     CON2+2
```

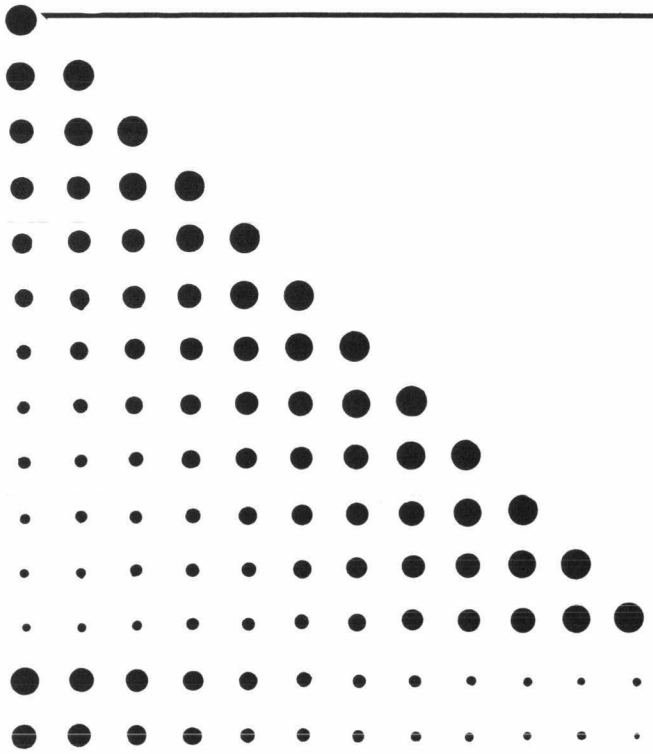
```
011F 23              INX   H              ;Step to conout link
0120 23              INX   H
0121 7E              MOV   A,M            ;Get conout link
0122 32FE01          STA   CON+1
0125 23              INX   H
0126 7E              MOV   A,M
0127 32FF01          STA   CON+2
012A 2100D4          LXI   H,DESTIN       ;Destination
012D 116501          LXI   D,LPDRV        ;Source
0130 01C300          LXI   B,ENDP-LPDRV   ;Byte count
0133 1A        LOOP  LDAX  D
0134 77              MOV   M,A
0135 13              INX   D
0136 23              INX   H
0137 0B              DCX   B
0138 78              MOV   A,B
0139 B1              ORA   C
013A C23301          JNZ   LOOP
013D E1              POP   H              ;Recover conout address
013E 11AFD4          LXI   D,CONSTS+BASE  ;Put pointer to consts
0141 73              MOV   M,E            ;at CONSTS vector
0142 23              INX   H
0143 72              MOV   M,D
0144 23              INX   H
0145 23              INX   H              ;Put pointer to conin
0146 119BD4          LXI   D,CONIN+BASE   ;at CONIN vector
0149 73              MOV   M,E
014A 23              INX   H
014B 72              MOV   M,D
014C 23              INX   H
014D 23              INX   H              ;Put pointer to conout
014E 118FD4          LXI   D,CONDRV+BASE  ;at CONOUT vector
0151 73              MOV   M,E            ;Put pointer to condrv
0152 23              INX   H              ;at conout vector
0153 72              MOV   M,D
0154 23              INX   H              ;Step to lister
0155 23              INX   H
0156 1100D4          LXI   D,LPDRV+BASE
0159 73              MOV   M,E            ;Put pointer to lpdrv
015A 23              INX   H              ;at lister vector
015B 72              MOV   M,D
015C CD6CD4          CALL  TABSET+BASE    ;Set up control port and set tabs
015F CD5FD4          CALL  RESET+BASE     ;Reset carriage to
                                          ;start of line/start of page
0162 C30000          JMP   0000H
               ;     START OF LINE PRINTER DRIVER
0165 79        LPDRV MOV   A,C
0166 E67F            ANI   7FH            ;Wipe off parity bits
0168 4F              MOV   C,A
0169 FE20            CPI   20H            ;Control?
016B D229D4          JNC   CHAR+BASE      ;Printable character
016E FE09            CPI   09H            ;Tab?
0170 CA2CD4          JZ    OUT8+BASE      ;Tab
0173 FE1B            CPI   1BH            ;Escape?
0175 CA23D4          JZ    ESC+BASE
0178 FE0A            CPI   0AH            ;Line feed?
017A CA2CD4          JZ    OUT8+BASE
017D FE0D            CPI   0DH            ;Carriage return?
017F CA2CD4          JZ    OUT8+BASE
0182 FE0C            CPI   0CH            ;Form feed?
0184 C0              RNZ                  ;Ignore all others
0185 C32CD4          JMP   OUT8+BASE
0188 3EFF      ESC   MVI   A,0FFH         ;Set escape flag
018A 32C2D4          STA   ESCFLG+BASE
018D C9              RET
018E FE7F      CHAR  CPI   7FH            ;Rubout?
0190 C8              RZ                   ;Don't print rubouts
0191 3AC2D4    OUT8  LDA   ESCFLG+BASE    ;Test escape flag for on
0194 B7              ORA   A
0195 CA4CD4          JZ    CHOUT+BASE     ;If not on, proceed!
0198 79              MOV   A,C            ;Test for ESC - R (reset)
0199 FE52            CPI   52H
019B CA5FD4          JZ    RESET+BASE
019E FE72            CPI   72H            ;Test for ESC - r (reset)
01A0 CA5FD4          JZ    RESET+BASE
01A3 FE54            CPI   54H            ;Test for ESC - T (tabset)
01A5 CA6CD4          JZ    TABSET+BASE
01A8 FE74            CPI   74H            ;Test for ESC - t (tabset)
01AA C5              PUSH  B              ;If not r or t, pass ESC-character
01AB 0E1B            MVI   C,1BH          ;sequence through
01AD CD4CD4          CALL  CHOUT+BASE
01B0 C1              POP   B
01B1 DB04      CHOUT IN    STATUS         ;Get status
01B3 EE04            XRI   JSENSE         ;Adjust for hi or lo true
01B5 E680            ANI   MASK           ;Isolate bit
01B7 CA4CD4          JZ    CHOUT+BASE
01BA 79        OUTCH MOV   A,C            ;Get char
01BB EE00            XRI   SENSE          ;High true logic
01BD D305            OUT   PORT
01BF AF              XRA   A              :Clear escape flag
01C0 32C2D4          STA   ESCFLG+BASE
01C3 C9              RET
01C4 C5        RESET PUSH  B              ;Save character
01C5 0E0D            MVI   C,0DH          ;Do a CR/FF
01C7 CD4CD4          CALL  CHOUT+BASE
01CA 0E0C            MVI   C,0CH
01CC CD4CD4          CALL  CHOUT+BASE
01CF C1              POP   B
01D0 C9              RET
01D1 C5        TABSET PUSH  B              ;Set up I/O control lines
01D2 D5              PUSH  D              ;and set tabs every 8 positions
01D3 AF              XRA   A              ;00 sets DSR,CTS,CXR
01D4 D304            OUT   STATUS
01D6 160F            MVI   D,0FH          ;15 tabs
01D8 1E08      TL0   MVI   E,08H          ;Every 8 spaces
01DA 0E14      TL1   MVI   C,20           ;Space over 1
01DC CD4CD4          CALL  CHOUT+BASE
01DF 1D              DCR   E
01E0 C275D4          JNZ   TL1+BASE       ;till 8 output, then
01E3 0E1B            MVI   C,1BH          ;output tabset sequence
01E5 CD4CD4          CALL  CHOUT+BASE
01E8 0E31            MVI   C,31H
01EA CD4CD4          CALL  CHOUT+BASE
01ED 15              DCR   D              ;Until 15 tabs are set
01EE C273D4          JNZ   TL0+BASE
01F1 D1              POP   D
01F2 C1              POP   B
01F3 C9              RET
01F4 DB01      CONDRV IN   SENSWT         ;Read sense switches
01F6 EE00            XRI   SENSEN         ;Adjust sense of sense switches
01F8 E602            ANI   SENMSK         ;Isolate print enable switch
01FA C400D4          CNZ   LPDRV+BASE
01FD C30000    CON   JMP   0              ;This location was filled
                                          ;in by the relocator to
                                          ;point to the main CON driver
0200 DB01      CONIN IN    SENSWT         ;Test sense switches
```

```
0202 EE00           XRI     SENSEN          ;Correct sense of switches
0204 E604           ANI     KEYMSK          ;Mask out keyboard substitute bit
0206 CA0000   CON2  JZ      0               ;Relocator fills in here
0209 CDAFD4   CON3  CALL    CONSTS+BASE     ;Test data ready
020C CAA4D4         JZ      CON3+BASE       ;Loop till ready
020F DB05           IN      PORT            ;Get data
0211 E67F           ANI     7FH             ;Mask out parity
0213 C9             RET
0214 DB01     CONSTS IN     SENSWT          ;Test substitution bit
0216 EE00           XRI     SENSEN
0218 E604           ANI     KEYMSK
021A CA0000   CON1  JZ      0               ;Relocator fills in here
021D DB04           IN      STATUS          ;Get status
021F EE04           XRI     JSENSE          ;Correct sense
0221 E640           ANI     INMASK          ;Get status bit
0223 C8             RZ                      ;Return if not ready--00
0224 3EFF           MVI     A,0FFH          ;Show ready--FF
0226 C9             RET
0227 00       ESCFLG DB     0               ;Escape flag
D29B =        BASE  EQU     DESTIN-LPDRV    ;Offset to relocation base
0228          ENDP  END     START

A>  R
?

A>TYPE TTYDEV.PRN


                ;    TELETYPE OUTPUT DRIVER FOR CP/M
                ;    Revision level 2.0       2-28-80
                ;
                ;    Steve Willoughby
                ;    Box 87
                ;    Crane,IN 47522
                ;
                ;
D400 =        DESTIN  EQU     0D400H          ;Driver destination address
0004 =        STATUS  EQU     04H             ;Status port
0080 =        MASK    EQU     80H             ;Printer ready bit
0005 =        PORT    EQU     05H             ;Printer port
0000 =        SENSE   EQU     00H             ;Sense of data
0004 =        JSENSE  EQU     04H             ;Sense of status
0001 =        SENSWT  EQU     01H             ;Sense switch port
0002 =        SENMSK  EQU     02H             ;Printer on bit
0000 =        SENSEN  EQU     00H             ;Sense of switch register
0042 =        PLENGTH EQU     66              ;Page length (in lines)
0048 =        LLENGTH EQU     72              ;Line length (in characters)
0100          START   ORG     100H
0100 317F00           LXI     SP,7FH          ;Set up stack
0103 2A0100           LHLD    0001H           ;Get warm boot address
0106 110A00           LXI     D,0AH           ;Point to console driver link
0109 19               DAD     D
010A E5               PUSH    H               ;Save conout address
010B 7E               MOV     A,M             ;Put it in program
010C 32F901           STA     CON+1
010F 23               INX     H
0110 7E               MOV     A,M
0111 32FA01           STA     CON+2
0114 2100D4           LXI     H,DESTIN        ;Destination
0117 114901           LXI     D,LPDRV         ;Source
011A 01B400           LXI     B,ENDP-LPDRV    ;Byte count
011D 1A       LOOP    LDAX    D
011E 77               MOV     M,A
011F 13               INX     D
0120 23               INX     H
0121 0B               DCX     B
0122 78               MOV     A,B
0123 B1               ORA     C
0124 C21D01           JNZ     LOOP
0127 E1               POP     H               ;Recover conout address
0128 11A6D4           LXI     D,CONDRV+BASE
012B 73               MOV     M,E             ;Put pointer to condrv
012C 23               INX     H               ;at conout vector
012D 72               MOV     M,D
012E 23               INX     H               ;Step to lister
012F 23               INX     H
0130 1100D4           LXI     D,LPDRV+BASE
0133 73               MOV     M,E             ;Put pointer to lpdrv
0134 23               INX     H               ;at lister vector
0135 72               MOV     M,D
0136 0E0D             MVI     C,0DH           ;Output a cr/lf
0138 CD87D4           CALL    CHOUT+BASE
013B CD9701           CALL    OUTLF           ;Feed new line
013E AF               XRA     A
013F 32B3D4           STA     CCNT+BASE
0142 3C               INR     A
0143 32FB01           STA     LCNT
0146 C30000           JMP     0000H
                ;     START OF LINE PRINTER DRIVER
0149 79       LPDRV   MOV     A,C
014A E67F             ANI     7FH             ;Wipe off parity bits
014C 4F               MOV     C,A
014D FE20             CPI     20H             ;Control?
014F D275D4           JNC     CHAR+BASE       ;Printable character
0152 FE09             CPI     09H             ;Tab?
0154 CA41D4           JZ      TAB+BASE        ;Tab
0157 FE0A             CPI     0AH             ;Line feed?
0159 CA5BD4           JZ      LF+BASE
015C FE0D             CPI     0DH             ;Carriage return?
015E CA66D4           JZ      CR+BASE
0161 FE0C             CPI     0CH             ;Form feed?
0163 C0               RNZ                     ;Ignore all others
0164 3AB2D4   FEED    LDA     LCNT+BASE       ;Get line count
0167 FE41             CPI     PLENGTH-1       ;Cut line yet?
0169 F229D4           JP      FEED1+BASE
016C CD4ED4           CALL    OUTLF+BASE      ;Output a LF
016F C31BD4           JMP     FEED+BASE       ; till line 65.
0172 0E2D     FEED1   MVI     C,2DH           ;Print a cut mark
0174 CD87D4           CALL    CHOUT+BASE
0177 0E0D             MVI     C,0DH           ;CR
0179 CD66D4           CALL    CR+BASE
017C CD4ED4           CALL    OUTLF+BASE      ;Advance to line PLENGTH + 1
017F CD4ED4           CALL    OUTLF+BASE
0182 3E01             MVI     A,1             ;Reset line count to 1
0184 32B2D4           STA     LCNT+BASE
0187 0E0C             MVI     C,0CH           ;Restore char for next dev
0189 C9               RET
018A 0E20     TAB     MVI     C,20H           ;Output spaces to tab
018C CD7BD4           CALL    OUT8+BASE
                                              ;Character count in A
018F E607             ANI     07H             ;Exact multiple of 8?
0191 C241D4           JNZ     TAB+BASE
0194 0E09             MVI     C,09H           ;Restore char
0196 C9               RET
0197 0E0A     OUTLF   MVI     C,0AH
0199 CD87D4           CALL    CHOUT+BASE      ;Output a LF
```

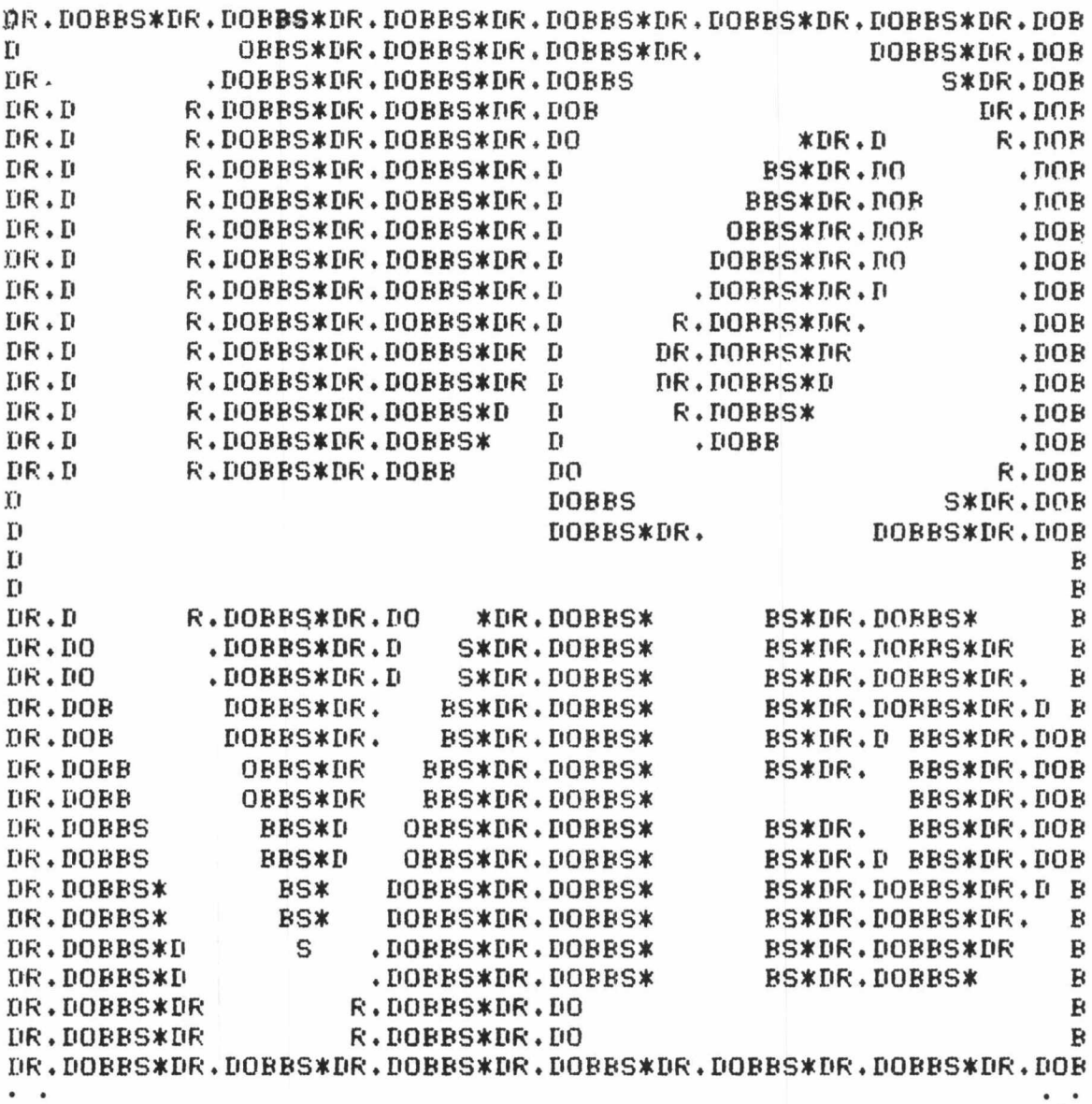
```
019C 3AB2D4          LDA     LCNT+BASE       ;Update line count
019F 3C              INR     A
01A0 32B2D4          STA     LCNT+BASE
01A3 C9              RET
01A4 CD4ED4    LF    CALL    OUTLF+BASE
01A7 FE40            CPI     PLENGTH-2       ;Blank line before cut mark?
01A9 CC1BD4          CZ      FEED+BASE       ;Yes--do auto form feed
01AC 0E0A            MVI     C,0AH           ;Restore char
01AE C9              RET
01AF CD87D4    CR    CALL    CHOUT+BASE      ;Output it and zero ccount
01B2 0E00            MVI     C,00H           ;Output 1 null
01B4 CD87D4          CALL    CHOUT+BASE
01B7 0E0D            MVI     C,0DH           ;Put cr back for main output
01B9 AF              XRA     A               ;Zero column count
01BA 32B3D4          STA     CCNT+BASE
01BD C9              RET
01BE FE7F      CHAR  CPI     7FH             ;Rubout?
01C0 C47BD4          CNZ     OUT8+BASE       ;Don't print rubouts
01C3 C9              RET
01C4 3AB3D4    OUT8  LDA     CCNT+BASE       ;Get char count
01C7 FE47            CPI     LLENGTH-1       ;Full line?
01C9 CAE201          JZ      NEWLIN          ;It full-do auto CR/LF
01CC 3C              INR     A               ;Up count
01CD 32B3D4          STA     CCNT+BASE       ;And save it
01D0 DB04      CHOUT IN      STATUS          ;Get status
01D2 EE04            XRI     JSENSE          ;Adjust for hi or lo true
01D4 E680            ANI     MASK            ;Isolate bit
01D6 CA87D4          JZ      CHOUT+BASE
01D9 79        OUTCH MOV     A,C             ;Get char
01DA EE00            XRI     SENSE           ;High true logic
01DC D305            OUT     PORT
01DE 3AB3D4          LDA     CCNT+BASE
01E1 C9              RET
01E2 C5        NEWLIN PUSH   B               ;Save character
01E3 0E0D            MVI     C,0DH           ;Do a CR/LF and let them
01E5 CD66D4          CALL    CR+BASE         ;decide if page eject is
01E8 CD5BD4          CALL    LF+BASE         ;to be done
01EB C1              POP     B
01EC C3C401          JMP     OUT8
01EF DB01      CONDRV IN      SENSWT          ;Read sense switches
01F1 EE00            XRI     SENSEN          ;Adjust sense of switches
01F3 E602            ANI     SENMSK          ;Which switch turns on printer
01F5 C400D4          CNZ     LPDRV+BASE
01F8 C30000    CON   JMP     0               ;This location was filled
                                             ;in by the relocator to
                                             ;point to the main CON driver
01FB 00        LCNT  DB      0               ;Line count
01FC 00        CCNT  DB      0               ;Column count
D2B7 =         BASE  EQU     DESTIN-LPDRV    ;Offset to relocation base
01FD           ENDP  END     START
```

```
A>type ttydev.prmm#
r
```





THE LOVE PROGRAM



BY G. GAUGLER
5706 Seascape Ct.
Citrus Heights, CA 95610

The enclosed listing is an assembly language (PDP-11) version of a program I originally saw in BASIC somewhere. The program was interesting, not terribly useful, and quite slow. However, this is exactly the kind of program one needs when trying to demonstrate a computer sytem to a non-computer person. I have found it rather successful for nocturnal social acquaintances, etc. Usually they would lose interest by the time the BASIC version would finish. This implementation will be as fast as the output device. An obvious advantage in such situations.

The fundamental principle of the program could be extended to other types of figures or drawings. The byte table contains the counts of characters and spaces (blanks) in an alternating order. A byte with the sign bit set (hence a negative quantity) indicates that the count is for characters at the far left of the figure; meaning that it starts a new row. The next byte would be the count of spaces. The following byte would be for the count of characters, and so on until another negative byte is found or a zero byte. The zero byte signals the end of the table.

The program prompts for a characters string which will be impressed into the figure. Character positions are preserved across blank fields (as can be seen in the enclosed example).

For any interested readers, I also have the same program coded in 6800 assembly language. A SASE will prompt me to send it to anyone who wants it.

```
LOVE    MACRO V03.02B16-MAR-80 01:11:12 PAGE 1


                        .TITLE  LOVE
                        .IDENT  /V001/
                ;
                ;
                ; WRITTEN BY GARY D. GAUGLER
                ;               5706 SEASCAPE CT.
                ;               CITRUS HEIGHTS, CA 95610
                ;
                        .MCALL  .PRINT,.GTLIN,.WRITW,.EXIT
                        .MCALL  .FETCH,.LOOKUP,.CLOSE
                ;
                ;
                        .MACRO  OUTPUT,ARG
                        MOVB    ARG,(R4)+
                        CALL    BUFOUT
                        .ENDM
                ;
                ;
        000015          CR=15
        000012          LF=12
                ;
                        .GLOBL  LOVE
                ;
                ;
000000          LOVE:
000000  005067          CLR     BLKNUM
        000460
000004  005002          CLR     R2
000006  012704          MOV     #BUFR,R4
        000466'
000012  012703          MOV     #512.,R3        ;BUFFER CHAR-COUNTER
        001000
000016  012701          MOV     #-1,R1          ;CHAR COUNTER
        177777
000022                  .GTLIN  #LINBUF,#MSG    ;PROMPT FOR INPUT
000042  012700          MOV     #LINBUF,R0      ;CHECK FOR TOO MANY CHARS
        001466'
000046  005201  L1:     INC     R1
000050  105720          TSTB    (R0)+           ;EOL?
000052  001375          BNE     L1              ;NOT YET
000054  020127          CMP     R1,#60.         ;> 60 CHARS?
        000074
000060  003347          BGT     LOVE            ;YEP.
                ;
                ; READY THE OUTPUT CHANNEL
                ;
000062                  .FETCH  #HSPACE,#LPNAME ;GET THE HANDLER
000074  103527          BCS     QUIT
000076                  .LOOKUP #EMTLST,#0,#LPNAME  ;ASSIGN THE CHANNEL
000116  103516          BCS     QUIT
                ;
000120  012700          MOV     #LINBUF,R0
        001466'
000124  012701          MOV     #TABLE,R1
        002204'
000130          LOVE1:
000130  004767          CALL    GETMSK          ;GET COUNT IN R2


LOVE    MACRO V03.02B16-MAR-80 01:11:12 PAGE 1-1


        000052
                ;
                ; PRINT BUFFER CHARS (R2) TIMES
                ;
000134          LOVE2:
000134                  OUTPUT  (R0)+           ;OUTPUT THE CHARACTER
000142  105710          TSTB    (R0)            ;END OF STRING?
000144  001002          BNE     L2              ;NOT YET
000146  012700          MOV     #LINBUF,R0      ;RESET TO START OF STRING
        001466'
000152  077210  L2:     SOB     R2,LOVE2
000154  004767          CALL    GETMSK          ;GET NUMBER OF SPACES
        000026
                ;
                ; OUTPUT NUMBER OF SPACES IN R2
                ;
000160          LOVE3:
000160                  OUTPUT  #40             ;OUTPUT A SPACE
000160  005200          INC     R0              ;INCR STRING POINTER
000170  105710          TSTB    (R0)            ;END OF STRING?
000172  001002          BNE     UPBUF1          ;NOT YET
000174  012700          MOV     #LINBUF,R0      ;RESET TO START OF STRING
        001466'
000176  077212  UPBUF1: SOB     R2,LOVE3        ;KEEP OUTPUTTING SPACES
000202  000751          BR      LOVE1           ;GET CHAR COUNT & CONTINUE
                ;
                ;
000204          GETMSK:
000206  112102          MOVB    (R1)+,R2        ;GET SPACE OR CHAR COUNT
000210  042702          BIC     #177400,R2      ;CLEAR UPPER BYTE
        177400
000214  105702          TSTB    R2              ;START OF NEW LINE?
000216  100402          BMI     NEWLIN          ;YES
000220  001417          BEQ     DONE            ;BR IF ALL DONE
000222  000207          RETURN                  ;RETURN WITH COUNT
                ;
                ;
000224          NEWLIN:
000224  142702          BICB    #200,R2         ;GET ONLY THE COUNT
        000200
000230  012700          MOV     #LINBUF,R0      ;RESET TO START OF STRING
        001466'
000234  005726          TST     (SP)+           ;SCRAP RETURN ADDRESS
000236                  OUTPUT  #CR             ;OUTPUT CR/LF
000246                  OUTPUT  #LF
000256  000726          BR      LOVE2           ;CONTINUE
                ;
                ; WRITE-OUT REMAINDER OF OUTPUT BUFFER
000260          DONE:
000260                  OUTPUT  #CR
000270                  OUTPUT  #LF
000300  012704          MOV     #512.,R4
        001000
000304  160304          SUB     R3,R4           ;SEE HOW MANY LEFT IN BUFFER
000306  006204          ASR     R4              ;CONVERT TO WORDS
000310                  .WRITW  #EMTLST,#0,#BUFR,R4,#BLKNUM
000346                  .CLOSE  #0


LOVE    MACRO V03.02B16-MAR-80 01:11:12 PAGE 1-2


                ;
000354          QUIT:   .EXIT
                ;
                ;
000356          BUFOUT:
000356  005303          DEC     R3              ;ONE LESS CHAR SPACE LEFT
000360  001026          BNE     BUFO1           ;STILL ROOM
000362  010046          MOV     R0,-(SP)        ;SAVE OUR R0
000364                  .WRITW  #EMTLST,#0,#BUFR,#256.,#BLKNUM
000424  012600          MOV     (SP)+,R0
000426  012704          MOV     #BUFR,R4        ;BEGINNING OF BUFFER NOW
        000466'
000432  012703          MOV     #512.,R3
        001000
000436  000207  BUFO1:  RETURN
                ;
                        .NLIST  BEX
                ;
000440  046600  LPNAME: .RAD50  /LP /           ;DEVICE NAME
000442  000000          .WORD   0
000444          EMTLST: .BLKW   10              ;EMT ARG AREA
000464          BLKNUM: .BLKW   1               ;BLOCK NUMBER
000466          BUFR:   .BLKW   256.            ;WRITE BUFFER
001466          LINBUF: .BLKB   82.
001610          MSG:
001610  033             .BYTE   33,52           ;CLEAR SCREEN (SOROC IQ120)
001612  012             .ASCII  <12><15>/THIS IS A TRIBUTE TO THE GREAT/
001652  012             .ASCII  <12><15>/AMERICAN ARTIST ROBERT INDIANA./
001713  012             .ASCII  <12><15>/HIS GREATEST WORK WILL BE REPRODUCED/
001761  012             .ASCII  <12><15>/WITH A MESSAGE OF YOUR CHOICE OF/
002023  012             .ASCII  <12><15>/UP TO 60 CHARACTERS. IF YOU CAN'T/
002066  012             .ASCII  <12><15>/THINK OF YOUR MESSAGE, SIMPLY TYPE/
002132  012             .ASCII  <12><15>/THE WORD 'LOVE'!/
```

```
    130 002154      012             .ASCII  <12><15>/YOUR MESSAGE PLEASE? /<200>
    131 002204      074     TABLE:  .BYTE   60.,201,12.,26.,9.,12.,203
    132 002213      010             .BYTE   8.,24.,17.,8.,204,6,23.
    133 002222      025             .BYTE   21.,6,204,6,22.,12.,5,6,5
    134 002233      204             .BYTE   204,6,21.,11.,8.,6,4,204,6,21.,10.,10.
    135 002247      005             .BYTE   5,4,204,6,21.,9.,11.,5,4,204,6,21.,8.
    136 002264      013             .BYTE   11.,6,4,204,6,21.,7,11.,7,4,204,6,21.
    137 002301      006             .BYTE   6,11.,8.,4,204,6,19.,1,1,5,11.,9.,4,204
    138 002317      006             .BYTE   6,19.,1,1,5,10.,10.,4,204,6,18.,2,1,6,8.
    139 002336      013             .BYTE   11.,4,204,6,17.,3,1,7,5,13.,4,204,6,15.,5
    140 002355      002             .BYTE   2,23.,5,201,29.,5,17.,8.,201,29.,9.,9.,12.
    141                             ; ROW 18
    142 002372      201             .BYTE   201,58.,1,201,58.,1,204,6,13.
    143 002403      003             .BYTE   3,10.,6,12.,5,1,205,6,11.,3,11.,6,14.,3,1,205,6
    144 002424      013             .BYTE   11.,3,11.,6,15.,2,1,206,6,9.,3,12.,6,16.,1
    145 002443      001             .BYTE   1,206,6,9.,3,12.,6,7,1,10.,207,6,7,3,13.
    146 002462      006             .BYTE   6,6,2,10.,207,6,7,3,13.,14.,10.,210,6,5
    147 002500      003             .BYTE   3,14.,6,6,2,10.,210,6,5,3,14.,6,7,1,10.,211
    148 002520      006             .BYTE   6,3,3,15.,6,16.,1,1,211,6,3,3,15.,6,15.,2
    149 002540      001             .BYTE   1,212,6,1,3,16.,6,14.,3,1,212,10.,16.,6,12.
    150 002557      005             .BYTE   5,1,213,8.,13.,27.,1,213,8.,13.,27.,1,274
    151 002574      000             .BYTE   0                       ;END OF TABLE
    152                     ;
    154                             .EVEN


LOVE    MACRO V03.02B16-MAR-80 01:11:12 PAGE 1-3

    155 002576              HSPACE: .BLKW   1000            ;SPACE FOR HANDLER
    156         000000'             .END    LOVE


LOVE    MACRO V03.02B16-MAR-80 01:11:12 PAGE 1-4
SYMBOL TABLE

BLKNUM  000464R         LF    = 000012          L2      000152R
BUFOUT  000356R         LINBUF  001466R         MSG     001610R
BUFO1   000436R         LOVE    000000RG        NEWLIN  000224R
BUFR    000466R         LOVE1   000130R         QUIT    000354R
CR    = 000015          LOVE2   000134R         TABLE   002204R
DONE    000260R         LOVE3   000160R         UPBUF1  000202R
EMTLST  000444R         LPNAME  000440R         ...V1 = 000003
GETMSK  000206R         L1      000046R         ...V2 = 000027
HSPACE  002576R

. ABS.  000000      000
        004576      001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  1852 WORDS  ( 8 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  67 PAGES
,DK1:LOVE=DK1:LOVE/L:TTM/C


LOVE    MACRO V03.02B16-MAR-80 01:11:12 PAGE S-1
CROSS REFERENCE TABLE (CREF V01-08 )

...V1   1-31    1-41    1-43    1-43    1-43    1-97    1-97
        1-98    1-98    1-107   1-107
...V2   1-43    1-43    1-43#   1-43#   1-97    1-97    1-97#
        1-97#   1-98    1-98#   1-107   1-107   1-107#  1-107#
BLKNUM  1-26#   1-97    1-107   1-118#
BUFO1   1-105   1-111#
BUFOUT  1-54    1-64    1-86    1-87    1-92    1-93    1-103#
BUFR    1-28    1-97    1-107   1-109   1-119#
CR      1-19#   1-86    1-92
DONE    1-78    1-91#
EMTLST  1-43    1-97    1-107   1-117#
GETMSK  1-49    1-59    1-73#
HSPACE  1-41    1-155#
L1      1-33#   1-35
L2      1-56    1-58#
LF      1-20#   1-87    1-93
LINBUF  1-31    1-32    1-46    1-57    1-68    1-84    1-120#
LOVE    1-22    1-25#   1-37    1-156
LOVE1   1-48#   1-70
LOVE2   1-53#   1-58    1-88
LOVE3   1-63#   1-69
LPNAME  1-41    1-43    1-115#
MSG     1-31    1-121#
NEWLIN  1-77    1-82#
QUIT    1-42    1-44    1-100#
TABLE   1-47    1-131#
UPBUF1  1-67    1-69#


LOVE    MACRO V03.02B16-MAR-80 01:11:12 PAGE M-1
CROSS REFERENCE TABLE (CREF V01-08 )

...CM0  1-31    1-31    1-31    1-41
...CM1  1-43    1-97    1-107
...CM2  1-43    1-43    1-97    1-97    1-97    1-97    1-107
        1-107   1-107   1-107
...CM3  1-98
...CM4  1-97    1-107
...CM5  1-41    1-43    1-97    1-107
.CLOSE  1-10#   1-98
.EXIT   1-9#    1-100
.FETCH  1-10#   1-41
.GTLIN  1-9#    1-31
.LOOKU  1-10#   1-43
.PRINT  1-9#
.WRITW  1-9#    1-97    1-107
OUTPUT  1-13#   1-54    1-64    1-86    1-87    1-92    1-93
```

ENDING

A PROBLEM WITH LOGIC PROBES

BY DAVID N. BERTOLLO
89 Westdale Avenue
Hillsdale, NJ 07842

ABSTRACT

While debugging a TTL system using the Micronta Model 22-300 "multi-logic compatable" digital logic probe, I discovered that this probe will only give a correct high level logic indication in fifty percent of the cases of a true TTL logic high. A TTL circuit is herein described which will operate properly when testing TTL logic levels.

The Problem

A few months ago, while debugging a new Z80 system, I decided a digital logic probe would be useful to have and that it would be much easier to purchase such a probe than to design and build one from scratch. Several digital logic probes are available from a number of manufacturers, ranging in price from $25-100.

On the surface, it appeared that the higher the probe's cost, the lower the value received. In the end, I decided to purchase the $25 variety: a Micronta 22-300. It seemed that this probe would suffice for all but a few fast pulse points in the system for which it would be used (the probe cannot indicate a pulse of under 300 nano-seconds duration). This probe also had the advantage of being advertised as working with several logic families.

After spending several hours trying to diagnose the ills of the new Z80 system with the Micronta probe, it seemed that more of the Z80's data lines were in an open state than seemed reasonable. Being suspicious of the probe, I studied the portion of the probe's user manual devoted to verifying the probe's correct operation. Using the test set-up shown, I determined that a logic low was being indicated for voltages of from 0 to 1.8 volts, and a logic high for inputs of 3.4 through 5 volts. Input voltages of 1.8 through 3.4 indicated an open condition. The manual indicated that these levels should be within 30 percent and 70 percent of Vcc and with an error range of ten percent. As a Vcc of five volts was being used, the proper voltages should be 0 through 1.5 for a logic low indication, 3.5 through 5 volts for a logic high indication, and 1.5 through 3.5 for an open indication. Thus, the probe appeared to be operating within the manufacturers specifications.

Wondering whether all those data lines were in their open state after all, I consulted a specification sheet for a 7400 TTL NAND gate. This told the story. A TTL logic low is defined as 0 through 0.8 volts and a logic high is defined as 2.4 through 5 volts. Further, a typical TTL gate will produce an output of between 3.3 and 5 volts. Therefore, fifty percent of all good TTL outputs will test properly with the Micronta probe; the other fifty percent, although perfectly good as far as TTL logic levels are concerned, will show up on this probe as opens.

In order to see if the Micronta's operations were typical of multi-family logic probes, I examined the specification sheet for the Heathkit TTL/CMOS digital logic probe (model 7410). When in TTL mode, this probe will indicate a logic zero for voltages of 0 through 0.8 volts and a logic high for voltages of 2.1 through 5 volts. When in CMOS mode, zero through thirty percent of Vcc gives a logic low; while seventy to one hundred percent of Vcc give a logic high indication.

Apparently, the Micronta probe will only operate properly with CMOS logic, even though it is advertised as having "multi-logic compatibility." In actuality, the only TTL compatibility this probe has is accidental.

At this point, three options were available: 1) Purchase a new probe, such as the one from Heathkit; 2) Modify the Micronta probe to operate properly with TTL logic levels; and, 3) Design and build a new probe for TTL logic.

As I did not desire to spend money on another logic probe, option 1 was quickly discarded. The Micronta probe did not come with a schematic. Having no love for tracing the probe's circuit, option 2 was swiftly discarded. This left only option 3.

Circuit Description

The circuit for the new probe is shown in the Figure. It uses two inexpensive TTL chips, 1 diode (for polarity protection), two capacitors, three LED's and 3 resistors: a total cost of under $3 for parts (less a case to house them). This probe will resolve TTL levels properly, and will also detect pulses as fast as 40 nanoseconds. Its two disadvantages are that it may only be used for testing TTL logic and it will not detect an open line condition (which will show up as a logic high).

The first hex inverter provides one TTL load to the circuit under test. Inverter stages two and three drive the high and low level LED logic state indicators. IC2 serves as a pulse stretcher. Any negative transition on the input line results in an output pulse of about 150 milliseconds, which lights the LED pulse indicator. For pulse trains slower than about seven pulses per second, the indicator will blink on and off. For faster pulse trains, the indicator will stay on continuously. For single pulses, it will blink once.

In order to make the indicators easier to differentiate, a red LED was used for the logic high indicator, a green LED for the logic low indicator and an amber (yellow) LED for the pulse indicator. The circuit was housed in the Micronta case, with the Micronta's innards now living in the junk box waiting for the day a CMOS logic probe is needed. As only a small number of components were required, the dead bug method of construction was used (i.e., the two IC's were glued to one side of the case, with direct solder connections made to their exposed pins).

Conclusion

Using the new TTL logic probe, those open state data lines were not open after all, the processor clock was no longer clocking and those 40 nano-second pulses looked fine. Within ten minutes of using the new probe, the Z80 processor was starting to respond to instructions; within four hours it was fully operational.

David Bertollo is a senior programmer for Information Sciences Division, Rockland Research Institute, Orangeburg, NY. He is involved in the development of both applications and systems programs for a network of mini computers feeding a maxi host computer. Before rebirth as a programmer, he was an electronics design engineer. –*SR*

(Continued on pg. 45)

an error-correcting record Linking method

BY SYLVAN RUBIN
Algorithmic Engineering
PO Box 3074
Santa Clara, CA 95051

To enable a program or operating system to verify and correct errors in the pointer links of linked record chains, a redundant set of links is necessary. This compact method uses a 'forward' link from each record to its successor record, and a 'cross' link, which is the XOR product of the previous record address and the successor record address. These two links are stored in each record. With a typical disk sectoring arrangement, they should probably be stored at opposite ends of the record, to minimize effects of recording medium flaws.

The major advantage of this method, compared to direct forward and backward links, which take the same space in each record, is that this algorithm can determine which link is wrong, and can correct it. Every record contains, in effect, two independent forward pointers (one based on the knowledge of the previous record address), and a back pointer (the XOR of the two links).

Because of the asymmetry of these links, verification and correction is best done during forward sequencing through the chain. Verification consists of:

> Determine the next record address by XORing the previous record address with the cross link in the current record. This should be identical with the forward link in the current record.

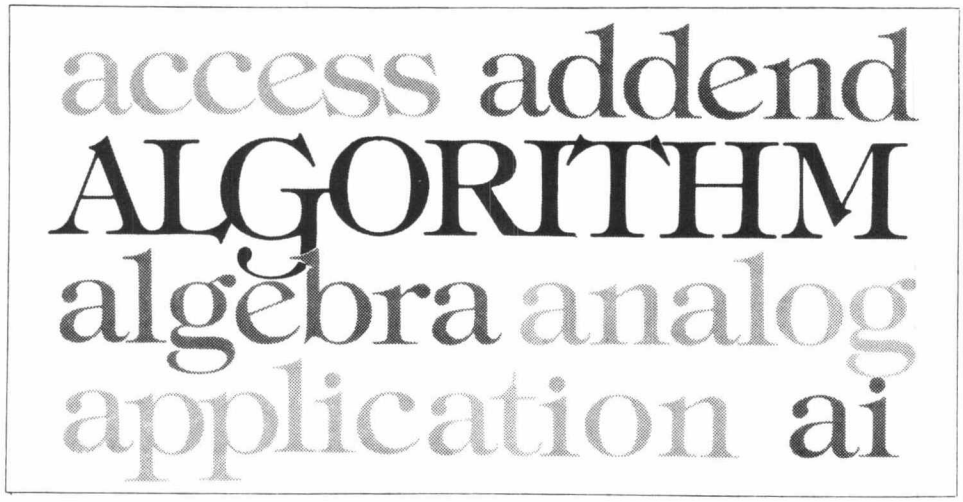
(Note that the XOR of any two of the above values should be equal to the third one. The procedure above yields an alternate successor address, if it is needed).

If the test fails, either the forward link or the cross link is wrong. The error can be corrected by examining both possible successor records, to determine which points back to the current record containing the error. The XOR result of the forward link and the cross link in the successor record should be the address of the current record. If one of the two tested successor records points back to the current record, the erroneous link is changed to point to that successor record. If the cross link is wrong, it is reconstructed by XORing the previous record address with the correct successor record address.

In the first record of a chain, the forward and cross links are equal, so their XOR result is 0. The logical address of the first element of a chain therefore must be non-zero, since the "previous address" is 0. In the last record, the forward link is 0, and the cross link is equal to the previous record address. Correction of links in the last record is a special case; if one successor address is 0, the non-zero successor record is examined to determine whether it points to the current record. If it does not, the current record is the end of a chain.

This link verification and correction algorithm can be done "on the fly" during any file read or write operations, so that randomly caused errors in the links can be routinely corrected without any error messages or significant overhead. No extra disk accesses are needed, except when an error is found

LINKS.N



BY DENNIS ALLISON

Many software systems need some sort of a symbol table. A symbol table is simply a mechanism which associates a name with some information. A name can be a string or a number. Normally one expects the space of possible names to be represented as much, much larger than the number of table entries. Further, there is usually a premium placed on finding entries efficiently.

Just what kind of mechanism is best for any particular application will depend upon the way the table is to be used and what information is to be stored in it. Let's presume we have an application where the entries are of fixed size and deletions will not be made. An assembler is one such application. The data can be represented by a structure of the form:

```
struct{
        char name [8] ;
        int val ;
} symtab [256] ;
```

One could, of course, fill this table from its first entry and determine whether a key is present by searching for it linearly. If *nentries* is the number of entries in the table, and *match* is a predicate function which returns non-zero if the keys match and zero if they do not, then one can search the table with the code:

```
lookup ( key ) char *key ; {

        int i;
        for(i =0; i <nentries;i ++)   {
           if( match(key,  symtab[i] .name)
                    return (i) ;
        /*  here only if fail */
        return ( -1 );
}
```

Here we have chosen to return the value −1 if the entry is not in the table. The average number of matches for an entry in the table is *nentries* /2 and the required number of matches to determine an entry is not in the table is *nentries*. We can improve upon this considerably.

For this particular situation a closed hashing scheme is effective. We provide a special function, *hash* which maps the key into an integer between 0 and 255. This function is chosen so that, on the average, for any large set of names all values will be generated the same number of times. As there are many more than 256 possible names, there is no guarantee that the hash value of two names which are themselves different is not the same.

The lookup algorithm then proceeds as follows: First, hash the key and find an index into the table. If the key matches the name stored at that location, we have found the entry. If the slot in the table is empty, the key does not appear in the table. Often, as we do below, the key is entered into the table at this point and the table index returned. If neither of these conditions hold, one must appeal to a collision resolution strategy. This key potentially hashes to the same value as some other table entry.

The traditional collision resolution strategy is to search the table linearly forward (or backward) modulo the table size. This is simple but tends to produce dense clusters of entries which degrades performance.

The scheme I like uses some properties of residue arithmetic and *depends upon the table size being a power of two*. An auxiliary counter is used which allows one to generate a sequence of values which steps through every element of the table, but the order of search is different for every initial value. This helps prevent clustering and improves the performance of the table. The particular algorithm used here was published by F. R. A. Hopgood and J. Davenport in *The Computer Journal* (15:4, November 1972). See their article for a discussion of the theory involved.

A program which implements their algorithm is:

```
lookup (key)  char *key; {

        int i, j;

        i = hash(key);  /* value is 0 to 255 */
        j = 1; /* used for collision resolution */

        for (;;)      /* infinite loop */

           /* test for empty */
           if (sym [i] .name[0] == ' ' ){
              scopy(key, sym[i]. name);
              return(i) ;
           }

           /* test for match */
           if ( match(key, sym [i] .name ) )
               return(i); /* found it */

           /* resolve a collision & test for full */
           i=(i+j) &  0xFF;
           if ( (j = (j + 1) & 0xFF) ==0) {
              error ("symbol table overflow");
              return (-1) ;
           }
        }
}
```

The function *scopy* copies the key into the designated table location. The logical-and operations are used to provide efficient arithmetic modulo the table size.

Performance of this particular table is very good when it is only partially full. When the table is less than 70% full, entries which are present can be found with an average of three probes. As the table fills, the performance will eventually degrade to a linear search, but full utilization of the storage space will be made.

There remains the problem of choosing the initial hash function. What one wants is to ensure that the information is spread pretty uniformly. One would like the string "ab" to generate a different hash than the string "ba." Lastly, one wants the operation to be efficient to implement.

One possibility is

```
hash (name) char *name; {

        int v, i;
        v = 0;
        for (i=0; i<8; i++){
                  v = v + v - *name++;
        }
        return(((v>>8) +v) & 0xFF);
}
```

This function gives fairly good results. On a base of 5000 words it gave frequency values ranging from 10 to 43 about two-thirds in the range of 15 to 24.

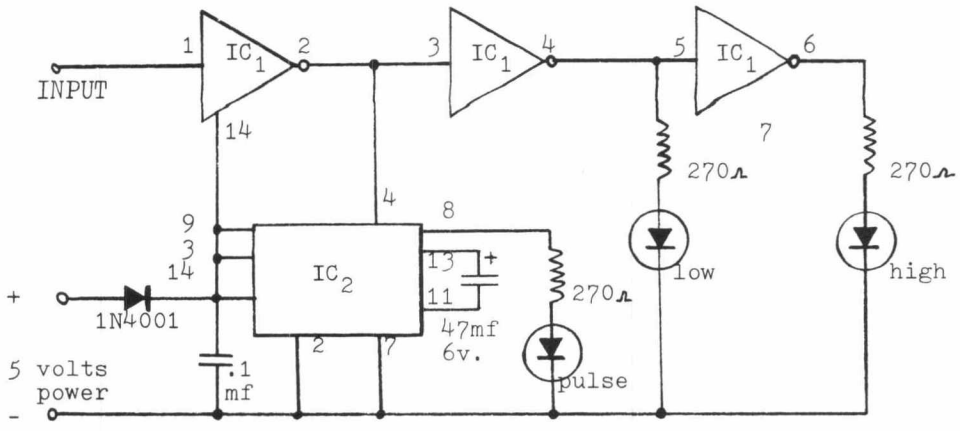
Now that we've got a mechanism like this, we can make a dandy little program which counts instances of words. We need

a procedure to fill the key array, call it *getword*. It reads a word from a file and places it in the buffer pointed to by it's argument. It returns non-zero as long as the buffer is valid, that is, as long as the end of the file has not been reached.

The main part of the program becomes

```
wordcount ( ) {
        char the word [ 8 ] ;
        while( getword ( the word ) )
          sym[lookup ( the word ) ] ;

        for (i = 0 ;i < 255 ;i++)
                if(sym [i] .name [0]
                  ! = ' ')
                    printf("%s %d0,
                      sym [i] .name ,
sym [i] .val ) ;
        }
```

(Continued from pg. 42)



IC_1 = 7404 Hex Inverter IC_2 = 74122 Monostable Mult.

FIGURE
Digital Logic Probe

Letters

THE DEBATE ON TOP-DOWN CONTINUES

Dear *DDJ*:

In *DDJ* #43 I stated that top-down programming is easy in COBOL but impossible in PASCAL. Mr. David Rowland contests this admittedly provocative statement (*DDJ* #46). He challenges me to define top-down programming in a way that excludes PASCAL and to defend that definition in a multi-lingual environment.

Top-down programming is an approach to programming that results in a particular structure in the finished product. The process of "stepwise refinement" cited by Mr. Rowland should be reflected in three ways: in the logic flow of the program, in the physical structure of the source listing and in the sequence of preparing and testing modules.

Necessarily, a program prepared by the top-down method is both modular and hierarchical. There is a main module where the program begins and ends. There are subordinate modules called by it. These may in turn call still lower level modules and so on to as many levels of hierarchy as the complexity of the task requires. There are also (in the real world) some little utility modules that are called all over the place and hence to not really fit anywhere in the hierarachy. In very large programs (compilers, operating systems) we may be dealing with hundreds of modules written by dozens of programmers.

I will use as an example a short program consisting of a main routine 000 which calls modules 100, 200, 300 and 400. Module 100 calls modules 110, 120 and 130. Module 200 calls modules 210 and 220 and so on. The program also uses utility routines 900, 910 and 920.

Proceeding in top-down fashion, after the general structure of the program has been roughed out we write module 000. We then write the modules subordinate to it, proceeding from left to right and from top to bottom in the hierarchical sequence of the program. After the main line module is written, there are two acceptable sequences for writing the remainder of the program. Usually it is best to sequence the program level by level. Our example program might be written thus:

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

Some programmers would find it convenient to put the 900 series modules early in the program, perhaps just after 000.

The second common sequence is the indented order. It is useful if the programmer likes to pursue a particular function to completion before proceeding further. It is also used where either the programmer or the operating system is going to do overlays to save memory space:

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

Note that the physical sequence of the modules is also the order in which they are written. Note also that the whole program is not written at one sitting. There are compilers for errors and test executions early in the life of the program. Modules called but not yet written are inserted as "dummy" routines.

Now let me offer a nonsensical sequencing of the same program, given our top-down approach to its construction:

```
900
910
920
        110
        120
        130
        210
        220
        310
        320
        330
        340
        410
    100
    200
    300
    400
000
```

This is the sequencing *required* by PASCAL, XPL and most other languages designed for use as teaching tools. In this class of languages a procedure must be defined before it is called. The main routine that should be written first must occur last in the source listing. Given a program of non-trivial dimensions this is an almost impossibly clumsy way to proceed. It also makes the finished program very hard to read since you must start near the end and jump around in the listing to follow the source text in any logical sequence.

Bottoms-up sequencing makes sense if you are writing an extensible language. It also allows implementation of a simple "one-pass" compiler. But for the serious programmer who is trying to produce a good product within a reasonable time span it is a miserable restriction. I have never seen a top-down program written in any commercial language that did not also order the program in a top-down sequence, with the main line routine first, followed by its subordinate routines in one of the two acceptable sequences described earlier. Languages such as COBOL, FORTRAN, PL/I and even most assemblers allow top-down sequencing of source code. PASCAL, unfortunately, does not.

I stated three ways in which the stepwise refinement and top-down programming approach should be reflected: in the physical sequence of program text, and the order of progam

development and of course in the logic structure of the program. PASCAL prohibits the first and thereby inhibits the second. I stand by my earlier assertion. Top-down programming as I understand the term is virtually impossible in a language that requires a bottoms-up sequencing of source modules. Mr. Rowland feels that he programs top-down "on (his) best days." I understand how difficult this must be in an inherently bottoms-up langauge like PASCAL, and I offer him my sympathy. In my experience, languages developed as classroom tools usually do not make good production languages. There are too many restrictions (such as the bottoms-up sequencing of PASCAL) which make sense only in the classroom. Indeed I question the utility of "upside down programming" even in an academic environment. Perhaps some knowledgeable reader can enlighten me on this issue.

Sincerely,
John R. Culleton, Jr.

2401 Haight Avenue
Sykesville, Maryland 21784

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 49 | October 1980 | Volume 5, Issue 9

A REFERENCE JOURNAL FOR USERS OF SMALL COMPUTERS

Even Further with Tiny BASIC 4
Robert Hudson

CP/M To UCSD PASCAL File Conversion 16
Tor Sjøwall

Extended Source-Text Area and Label Capability for SC/MP NIBASM 20
W.F. Long

MFT: A Multiple File Transfer Utility 24
Lawrence E. Hughes

A Cross Reference Program for Northstar BASIC 32
Larry Hudson

and more ...

EVEN FURTHER WITH TINY BASIC

BY ROBERT HUDSON
1315 August Drive
Austin, TX 78753

No – not full floating point math, which is unnecessary for a control language or string functions; but extended – yes, indeed. The original Tiny BASIC for the Motorola 6800 as implemented by Pittman is an extremely small (2K) and powerful BASIC ideally suited for a minimum system or control application. The original Tiny BASIC has the following commands:

LET var	GOTO expr	REM
IF . . . THEN	GOSUB expr	CLEAR
INPUT var	RETURN	LIST
PRINT	END	RUN

Variable Names A – Z.

Operators:

+ Addition
– Subtraction
* Multiplication
/ Division
( ) Parentheses

Functions:

RND
USR

Relational Operators:

= Equality
< Less than
> Greater than or equal to
<= Less than or equal to
>= Greater than or equal to
<> Not equal

As illustrated above, Tom Pittman has performed a fantastic feat of programming, considering the interpreter only requires 2K bytes. You may wonder what extensions are possible for such a colorful and rich minimum implementation of BASIC.

My extensions are:

SET
SET&
GET
GET&
CHR
NEW (CLEAR)
INPUT "TEXT"" var
FRE

Logicals:

AND
OR
NOT

The above extensions require only an additional 128 bytes of memory.

Tiny BASIC was originally a concept of Dennis Allison's and was actually the foundation which helped to create *Dr. Dobb's Journal*. The first few issues of *Dr. Dobb's* covered the theory of the tiny interpreters. I became intrigued with Tiny about a year ago, but had time to delve into it only recently. What really intrigued me were the two levels of interpretation. By this I mean the tiny BASIC interpreter is written in an Intermediate Language (IL), not machine code. The Intermediate Language is interpreted (read and decoded) by machine language routines. And since the IL is well defined with operation codes and addressing schemes, all the user has to do to change, alter, or add to a particular version is to redefine the IL program.

In Figure 1 is a program written in MITS 6800 BASIC (written by MICROSOFT). The program is an assembler for the IL code of Tiny BASIC. Because I did want the ability to redefine, expand and enhance Tiny, I had to write the assembler to facilitate this. This was necessitated mainly because the IL relies heaviliy on relative addressing. A great help in understanding the operation and mnemonics of the IL program was a book published by Tom Pittman, "Tiny BASIC Experimenter's Kit," which sells for about $15. If anyone wishes to pursue experimenting with Tiny and Pittman's IL program, then this book is a must. In the book Pittman has an assembler for Tiny IL code written in Tiny BASIC, but it was inadequate for my needs. Therefore, I developed my own assembler. In the book Pittman completely explains the operation codes and the addressing modes.

Also included in this article is a listing of the assembled enhanced Tiny BASIC IL code to implement the above additional functions. This can be found in Figure 2. In Figure 3 is a 6800 assembler listing of the machine language routines necessary to implement some of the additional functions. In Figure 4 is a hexadecimal memory dump of the installed enhanced IL interpreter.
I will briefly explain these:

IL BASIC ASSEMBLER PROGRAM (Figure 1)

A two-pass assembler, the first pass being necessary to resolve forward referenced labels. The source code file is part of the program being stored in DATA statements. A leading "*" asterisk indicates a comment line. A leading ">" greater than indicates a label. The source file is stored from line 90 to line 840. Following the source file are a group of DATA statements which define the operation codes of the Tiny IL. The main program loop is entered at line 1030 where an option menu is presented. "Assembler pass 1" must be selected first, followed by "Assembler pass 2". Pass 2 generates a paginated listing as can be seen in Figure 2. The assembler has complete error trapping except for the "GX" op code.

This op code normally has a maximum argument of 7. I decided not to check the argument in order to facilitate implementation of new op codes. Also the mnemonic for jump was changed from "J" to "JP", therefore making all the op codes the same length. The only other alteration necessary was with the circumflex or up-arrow. This is normally used to generate ASCII control codes by subtracting 64 from the preceding character—this is implemented. However, because I store the source file as DATA statements it was not possible to use the comma "," in a source line of code because my BASIC interprets the comma as a delimiter in the DATA statement. I therefore used the circumflex to also generate the necessary ASCII code for the comma (lower case L – 64 = 44). The same is also true of the colon ":". I had to use the lower case Z. This brings up an interesting point also, in the IL code the "@" symbol is used in conjunction with the circumflex to generate the ASCII null code. There was a slight problem with this because my BASIC interprets the "@" symbol as an input line delete code. Because of this the user will find in line 70 four POKE commands which are used to NO OP the area of the BASIC interpreter (KCBASIC) which honors the "@" as the input line delete code. Therefore, if, when using a similar but different BASIC **please**, delete this line or change the POKE addresses to agree with your BASIC.

ASSEMBLER OUTPUT (Figure 2)

The BASIC Assembler program produces a titled and paginated listing of the assembled IL code. The listing is complete, even including the error code numbers which are generated by incorrect operation. The listing provides a self documentation of the TINY IL code. After assembly the IL code is completely position independent and only one pointer need be changed to point to the new IL code. In other words, one may have several IL interpreters in a computer at one time and switch back and forth between them just by changing a pointer to the beginning of the IL code desired.

Changes and Extensions:

Prompt Character – changed from a colon ":" to the greater than symbol ">".

New – the "CLEAR" command was changed to "NEW" to agree with other common BASIC interpreters.

SET – stores a one byte word in memory.
Syntax: SET address expression = expression

SET& – stores a two byte word in memory.
Syntax: SET& address expression = expression

GET – returns the value of a one byte word from memory.
Syntax: GET (address expression)

GET& – returns the value of two bytes from memory.
Syntax: GET& (address expression)

CHR= – outputs the ASCII character indicated by the evaluation of an expression. If the expression is greater than 255 then output is mod 256.
Syntax: CHR= expression

INPUT – now allows an optional text string to be printed.
Syntax: INPUT "text" variable

FRE – returns the number of bytes remaining in memory in which a program can be stored.
Syntax: PRINT FRE

Logicals:

AND – returns the 16 bit results of a logical AND operation between the two arguments.
Syntax: AND (first expression, second expression)

OR – returns the 16 bit results of a logical OR operation between the two arguments.
Syntax: OR (first expression, second expression)

NOT – returns the 16 bit complement of the argument.
Syntax: NOT (expression)

MACHINE LANGUAGE ROUTINES REQUIRED (Figure 3)

Machine language routines had to be written in 6800 code to implement the logical operators and also the two byte commands (SET& and GET&). The other commands were implemented without additional machine language routines being necessary. However, the following is very important for custom installation and understanding.

SET uses the built-in POKE function located at 0118 HEX. This should be the same for all Tiny BASICs. GET uses the built in PEEK function located at 0114 HEX. This should also be the same for all Tiny BASICs.

CHR bypasses the regular output routine in Tiny and goes directly to your system output routine. The address which points to your output routine should be inserted on line 37 of the IL listing (Figure 2). Since my system is not MIKBUG, this address currently points to my output routine. All that is necessary is to change the source code to LN 265 which will produce IL code of 0A0109. Location 0109 being the tiny BASIC jump to the character output routines in MIKBUG.

FRE was implemented wtihout any problem. However, for other Tiny BASICs using different MPUs the storage locations for the limits of memory may be different. As can be seen in the IL listing (Figure 2) for the 6800 version these are defined as 34 HEX and 36 HEX in lines 224 and 226.

Logical functions are defined in the IL code (Figure 2) from line 196 to 213. They call the appropriate machine language routines as found in Figure 3.

INSTALLATION

Load your Tiny BASIC by Tom Pittman. Now using your system's appropriate memory change and examine commands, enter the code found in Figure 4 starting at 07A1 HEX. This will overlay the previous IL code with the new IL code. Change location 07F2 HEX from 07 92 to 01 09. This will permit the CHR function to use your output routines which location 0109 HEX should be pointing to.

Now change the breakpoint jump located at 010C HEX to contain 097A HEX. Add at 097A HEX the following: 7E 07 6A 7E 07 8D.

Now change the following pointers in the Tiny BASIC program:

location 01FE HEX change to 07 BC – points to IL code beginning
location 0201 HEX change to 09 08 – points to program storage.

This concludes the installation procedure. Now begin the Tiny BASIC interpreter at 0100 HEX and have fun.

REFERENCES

1. *TINY BASIC User's Manual.* Available from ITTY BITTY COMputers, P.O. Box 32189, San Jose, CA 95153.
2. *Doctor Dobb's Journal*, Volume I, No. 7
3. TINY BASIC Experimenter's Kit. Available from ITTY BITTY COMPUTERS, P.O. Box 23189, San Jose, CA 95153.

LISTING →

FIGURE 1

```
10 REMARK  WRITTEN BY ROBERT HUDSON
20 REMARK  COPYRIGHT 1979
30 CLEAR 400
40 LET PG=0:LET BM=5:LET TM=5:LET PL=63:LET LN=0
50 DIM O$(41),L$(100),L(100),H$(15)
60 LET TITLE$="ENHANCED AND MODIFIED TOM PITTMAN'S TINY BASIC"
70 POKE1041,1:POKE1042,1:POKE1043,1:POKE1044,1
80 REM SOURCE CODE FILE
90 DATA * MODIFIED TINY BASIC    6/16/1979
100 DATA * BY ROBERT HUDSON
110 DATA * ASSEMBLER WRITTEN IN MITS 680B BASIC
120 DATA >STRT PC ">Q↑",GL,SB,BE L0,BR STRT
130 DATA >L0 BN STMT,IL,BR STRT
140 DATA >XEC SB,RB,XQ
150 DATA >STMT BC SET "LET",BV *,BC * "="
160 DATA >LET JS EXPR,BE *,SV,NX
170 DATA * "SET&" ADDRESS EXPR "=" EXPR
180 DATA >SET BC CHR "SET",BC SET1 "&"
190 DATA LN 1975,BR CSET
200 DATA >SET1 LN 280,>CSET JS EXPR
210 DATA BC * "=",JS EXPR,BE *,US,SP,NX
220 DATA * "CHR=" EXPR
230 DATA >CHR BC GOTO "CHR="
240 DATA LN 1938,JS EXPR,BE *,DS,US,SP,NX
250 DATA >GOTO BC PRNT "GO"
260 DATA BC GOSB "TO",JS EXPR,BE *,SB,RB,GO
270 DATA >GOSB BC * "SUB",JS EXPR,BE *,GS,GO
280 DATA >PRNT BC SKIP "PR",BC P0 "INT"
290 DATA >P0 BE P3,BR P6,>P1 BC P4 ";"
300 DATA >P2 BE P3,NX,>P3 BC P7 '"'
310 DATA PQ,BR P1,>SKIP BR IF
320 DATA >P4 BC P5 "1↑",PT,BR P2,>P5 BC P6 "z↑"
330 DATA PC "S↑",>P6 BE *,NL,NX,>P7 JS EXPR
340 DATA PN,BR P1
350 DATA >IF BC INPT "IF",JS EXPR,JS RELO
360 DATA JS EXPR,BC I1 "THEN",>I1 CP,NX,JP STMT
370 DATA >INPT BC RETN "INPUT",* "INPUT" "TEXT" VAR
380 DATA >I15 BC I2 '"',PQ
390 DATA >I2 BV *,SB,BE I4,>I3 PC "? Q↑",GL,BE I4,BR I3
400 DATA >I4 BC I5 "1↑",>I5 JS EXPR,SV,RB,BC I6 "1↑"
410 DATA BR I2,>I6 BE *,NX,>RETN BC END "RETURN"
420 DATA BE *,RS,NX,>END BC LIST "END",BE *,WS
430 DATA >LIST BC RUN "LIST",BE L2,>L1 PC "@↑@↑@↑@↑@↑J↑@↑"
440 DATA LS,PC "S↑",NL,NX,>L2 JS EXPR,BE L3,BR L1
450 DATA >L3 BC * "1↑",BR L2,>RUN BC CLER "RUN",JP XEC
460 DATA >CLER BC REM "NEW",MT
470 DATA >REM BC DFLT "REM",NX,>DFLT BV *,BC * "=",JP LET
480 DATA >EXPR BC E0 "-",JS TERM,NE,BR E1
490 DATA >E0 BC E4 "+",>E4 JS TERM,>E1 BC E2 "+"
500 DATA JS TERM,AD,BR E1
510 DATA >E2 BC E3 "-",JS TERM,SU,BR E1,>E3 RT
520 DATA >TERM JS FACT,>T0 BC T1 "*",JS FACT,MP,BR T0
530 DATA >T1 BC T2 "/",JS FACT,DV,BR T0,>T2 RT
540 DATA >FACT BC F0 "RND",LN 32896,FV,LN 2345,MP
550 DATA LN 6789,AD,SV,LB 128,FV,DS,JS FUNC,BR F1
560 DATA >F0 BR F2,>F1 DS,SX 2,SX 4,SX 2,SX 3,SX 5,SX 3
570 DATA DV,MP,SU,DS,LB 6,LN 0,CP,NE,RT
580 DATA >F2 BC F20 "USR",BC * "(",JS EXPR
590 DATA JS ARG,JS ARG,BC * ")",US,RT
600 DATA * "AND(" EXPR1 " " EXPR2 ")"
610 DATA >F20 BC F22 "AND(",LN 1953,BR F23
620 DATA * "OR(" EXP1 " " EXPR2 ")",>F22 BC F26 "OR("
630 DATA LN 1960,>F23 JS EXPR,JS ARG
640 DATA >F24 BC * ")",US,RT
650 DATA * "NOT(" EXPR ")"
660 DATA >F26 BC GET "NOT(",LN 1967
670 DATA >F28 JS EXPR,DS,BR F24
680 DATA * "GET&(" ADDRESS EXPR ")"
690 DATA >GET BC FRE "GET",BC GET1 "&",LN 1970,BR CGET
700 DATA >GET1 LN 276,>CGET BC * "(",BR F28
710 DATA * RETURNS FREE MEMORY  "PRINT FRE"
720 DATA >FRE BC F3 "FRE",LB 34,FV,LB 36,FV,SU,RT
730 DATA >F3 BV F4,FV,RT
740 DATA >F4 BN F5,RT,>F5 BC * "("
750 DATA >F6 JS EXPR,BC * "(",RT
760 DATA >ARG BC A0 "1↑",JP EXPR,>A0 DS,RT
770 DATA >FUNC BC * "(",BR F6,RT
780 DATA >RELO BC R0 "=",LB 2,RT
790 DATA >R0 BC R4 "<",BC R1 "=",LB 3,RT
800 DATA >R1 BC R3 ">",LB 5,RT
810 DATA >R3 LB 1,RT,>R4 BC * ">",BC R5 "="
820 DATA LB 6,RT,>R5 BC R6 "<",LB 5,RT
830 DATA >R6 LB 4,RT
840 DATA END
850 REM READ SOURCE CODE
860 LET N=0
870 READ S$
880 LET N=N+1
890 IF S$<>"END" THEN  870
900 PRINT "TOTAL NUMBER OF SOURCE CODE LINES =";N
910 REM SETUP OP CODE TABLE
920 DATA SX10,JS212288,JP214336,BR196,BC1128
930 DATA BV1160,BN1192,BE1224,LB29,LN310
940 DATA NO18,DS111,SP112,SB116,RB117,FV118
950 DATA SV119,GS120,RS121,GO122,NE123,AD124
960 DATA SU125,MP126,DV127,CP128,NX129,LS131
970 DATA PN132,PQ133,PT134,NL135,PC136,GL139
980 DATA IL142,MT143,XQ144,WS145,US146,RT147,**1255
990 FORI=1TO41:READO$(I):NEXTI
1000 REM HEX TABLE
1010 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
1020 FOR I=0 TO 15:READ H$(I):NEXT I
1030 REM MAIN LOOP
1040 RESTORE
1050 PRINT "1) ASSEMBLER PASS 1"
1060 PRINT "2) ASSEMBLER PASS 2"
1070 PRINT "3) LIST SOURCE FILE"
1080 INPUT M:IF M<1 OR M>3 THEN 1080
1090 LET LN=PL+BM+TM
1100 ON M GOSUB 1580,1110,1780:PRINT:PRINT:GOTO 1030
1110 REM ASSEMBLER PASS 2
1120 LET C=0
1130 FOR I=1 TO N-1:LET P=1:LET T$=""
1140 GOSUB 2280
1150 READ S$:LET F$=LEFT$(S$,1)
1160 IF F$="*" THEN 1280
1170 IF F$=">" THEN GOSUB 1930
1180 IF F$="E" THEN 1280
1190 REM GET OP CODE
1200 PRINT C;TAB(6):LET X=C:GOSUB 1880:LET X$=MID$(S$,P,2)
1210 GOSUB 2030:IF J<11 THEN GOSUB 1380
1220 REM PRINT ADDR AND OP CODE
1230 PRINT TAB(12);
1240 LET X=V:IF X>255 THEN GOSUB 1880:GOTO 1270
1250 GOSUB 1870:LET X=S
1260 IF X=36 OR X=128 THEN GOSUB 2080:LET C=C+(LEN(T$)/2)
1270 PRINT T$;
1280 REM PRINT SOURCE
```

```
1290 PRINT TAB(26)I;TAB(31);:IF ASC(F$)>64 THEN PRINT " ";
1300 PRINT S$:NEXT I:READ S$:PRINT TAB(31)S$
1310 REM LIST ERROR AND LABEL
1320 PRINT:PRINT "TOTAL ERRORS=";E
1330 LET M=4
1340 IF LN>3 THEN PRINT:GOSUB 2280:GOTO 1340
1350 FOR I=1 TO A:LET X=L(I):PRINT L$(I)TAB(10)XTAB(20);:GOSUB 1880
1360 PRINT:GOSUB 2280:NEXT I
1370 RETURN
1380 REM SPECIAL PROCESSING
1390 GOSUB 1930
1400 REM GET ARGUMENT
1410 LET T=P:GOSUB 1990:LET X$=MID$(S$,T,P-T)
1420 REM PROCESS ARGUMENT
1430 IF X$="*" THEN RETURN
1440 IF V>63 THEN 1500
1450 IF LEFT$(X$,1)>"9" THEN GOSUB 2210:GOTO 1470
1460 LET B=VAL(X$)
1470 IF V=0 THEN LET V=B:RETURN
1480 IF V=9 THEN LET X=B:GOSUB 1900:LET T$=X$:RETURN
1490 LET X=B:GOTO 1910
1500 REM PROCESS JUMPS AND BRANCHES
1510 GOSUB 2240
1520 IF B<0 OR B>2047 THEN GOSUB 1570
1530 IF V>256 THEN LET V=V+B:RETURN
1540 IF B-C>31 OR B-C<-31 THEN GOSUB 1570
1550 IF V>96 AND B-C<0 THEN GOSUB 1570
1560 LET V=V+(B-C):RETURN
1570 PRINT "OFFSET";:GOTO 1970
1580 REM ASSEMBLER PASS 1
1590 LET A=0:LET C=0:LET E=0:LET Q=0
1600 FOR I=1 TO N-1:LET P=1
1610 READ S$:LET F$=LEFT$(S$,1)
1620 IF F$>"?" THEN 1730:REM MUST BE OP CODE
1630 IF F$<">" THEN 1770:REM MUST BE COMMENT
1640 LET P=P+1:IF P>LEN(S$) THEN GOSUB 1970
1650 GOSUB 1990
1660 IF P>(LEN(S$)-2) THEN GOSUB 1970
1670 LET X$=MID$(S$,2,P-2)
1680 FOR K=1 TO A
1690 IF L$(K)=X$ THEN PRINT "DUPLICATE LABEL";:GOTO 1970
1700 NEXT K
1710 LET A=A+1:LET L$(A)=X$:LET L(A)=C
1720 GOSUB 1940
1730 IF F$="E" THEN GOSUB 2150:GOTO 1770
1740 LET X$=MID$(S$,P,2):GOSUB 2030
1750 LET X=V:IF X=128 THEN GOSUB 1930
1760 IF X=36 OR X=128 THEN GOSUB 2080:LET C=C+(LEN(T$)/2)
1770 NEXT I:PRINT:PRINT "TOTAL ERRORS=";E:RETURN
1780 REM PRINT SOURCE FILE
1790 FOR I=1 TO N
1800 GOSUB 2280
1810 READ S$
1820 PRINT I;TAB(9);
1830 IF LEFT$(S$,1)<"A" THEN 1850
1840 PRINT " ";
1850 PRINT S$:NEXT I:RETURN
1860 REM SUBROUTINES
1870 PRINT H$(INT(X/16));H$(X AND 15);:RETURN
1880 LET T=X:LET X=INT(T/256):GOSUB 1870:LET X=T-256*X:GOSUB 1870
1890 LET X=T:RETURN
1900 LET X$=H$(INT(X/16))+H$(X AND 15):RETURN
1910 LET T=X:LET X=INT(T/256):GOSUB 1900:LET T$=X$
1920 LET X=T-256*X:GOSUB 1900:LET T$=T$+X$:LET X=T:RETURN
1930 GOSUB 1990
1940 REM SKIP OVER SPACE
1950 IF MID$(S$,P,1)=" " THEN LET P=P+1:GOTO 1950
1960 RETURN
1970 REM PRINT ERROR MESSAGE
1980 PRINT " ERROR IN LINE #";I:LET E=E+1:RETURN
1990 REM SKIP TO SPACE
2000 IF P>LEN(S$) THEN 2020
2010 IF MID$(S$,P,1)<>" " THEN LET P=P+1:GOTO 2000
2020 RETURN
2030 REM FIND OP CODE
2040 FOR J=1 TO 40:IF X$=LEFT$(O$(J),2) THEN 2060
2050 NEXT J:PRINT "OP CODE";:GOSUB 1970
2060 LET C=C+VAL(MID$(O$(J),3,1)):LET V=VAL(MID$(O$(J),4))
2070 LET S=V:RETURN
2080 REM GET TEXT STRING
2090 GOSUB 1930:LET D$=MID$(S$,P,1):LET T=P+1:LET T$=""
2100 LET P=P+1:LET X=ASC(MID$(S$,P,1)):IF X=94 THEN 2100
2110 IFMID$(S$,P+1,1)="↑"THENX=X-64:IFMID$(S$,P+2,1)=D$THENX=X+128
2120 IF MID$(S$,P+1,1)=D$ THEN LET X=X+128
2130 GOSUB 1900:LET T$=T$+X$:IF X>127 THEN RETURN
2140 GOTO 2100
2150 REM EQUATE
2160 GOSUB 1930:LET X=P:GOSUB 1990:LET Q=Q+1
2170 LET Q$(Q)=MID$(S$,X,P-X)
2180 GOSUB 1940:LET X=P:GOSUB 1990
2190 LET Q(Q)=VAL(MID$(S$,X,P-X))
2200 RETURN
2210 REM GET VALUE FROM EQUATE
2220 FOR K=1 TO Q:IF X$=Q$(K) THEN LET B=Q(K):RETURN
2230 NEXT K:GOTO 2270
2240 REM GET LOCATION OF LABEL
2250 FOR K=1 TO A:IF X$=L$(K) THEN LET B=L(K):RETURN
2260 NEXT K
2270 PRINT "NO VALUE";:LET B=0:GOTO 1970
2280 REM PAGINATION CONTROL
2290 LET LN=LN+1:IF (LN+BM+TM)<PL THEN RETURN
2300 FOR K=1 TO BM:PRINT:NEXT K:PRINT "-"
2310 FOR K=1 TO TM:PRINT:NEXT K:LET PG=PG+1:LET LN=0
2320 PRINT TITLE$;SPC(60-LEN(TITLE$));" PAGE";PG
2330 ON M GOTO 2340,2350,2390,2420
2340 RETURN
2350 PRINT
2360 PRINT "ERROR  HEX  TINY BASIC    LINE  SOURCE   AND COMMENTS"
2370 PRINT "CODE  ADDR  IL CODE        #"
2380 LET LN=3:RETURN
2390 PRINT
2400 PRINT "LINE #   SOURCE AND COMMENT"
2410 LET LN=2:RETURN
2420 PRINT
2430 PRINT "SYMBOL TABLE"
2440 PRINT "LABEL    DECIMAL    HEX"
2450 LET LN=3:RETURN
OK
```

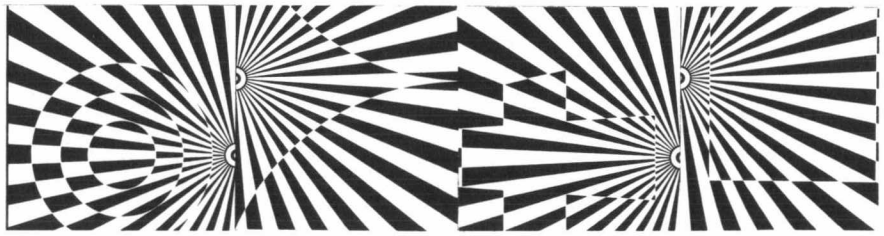


FIGURE 2

Enhanced and Modified TOM PITTMAN'S Tiny Basic

```
ERROR  HEX   TINY BASIC     LINE  SOURCE    AND COMMENTS
CODE   ADDR  IL CODE         #
                             1   * MODIFIED TINY BASIC    6/16/1979
                             2   * BY ROBERT HUDSON
                             3   * ASSEMBLER WRITTEN IN MITS 680B BASIC
 0     0000  243E91          4   >STRT PC ">Q↑"
 3     0003  27              5    GL
 4     0004  10              6    SB
 5     0005  E1              7    BE L0
 6     0006  59              8    BR STRT
 7     0007  C5              9   >L0 BN STMT
 8     0008  2A             10    IL
 9     0009  56             11    BR STRT
10     000A  10             12   >XEC SB
11     000B  11             13    RB
12     000C  2C             14    XQ
13     000D  8B4C45D4       15   >STMT BC SET "LET"
17     0011  A0             16    BV *
18     0012  80BD           17    BC * "="
20     0014  30E3           18   >LET JS EXPR
22     0016  E0             19    BE *
23     0017  13             20    SV
24     0018  1D             21    NX
                            22   * "SET&" ADDRESS EXPR "=" EXPR
25     0019  965345D4       23   >SET BC CHR "SET"
29     001D  85A6           24    BC SET1 "&"
31     001F  0A07B7         25    LN 1975
34     0022  63             26    BR CSET
35     0023  0A0118         27   >SET1 LN 280
38     0026  30E3           28   >CSET JS EXPR
40     0028  80BD           29    BC * "="
42     002A  30E3           30    JS EXPR
44     002C  E0             31    BE *
45     002D  2E             32    US
46     002E  0C             33    SP
47     002F  1D             34    NX
                            35   * "CHR=" EXPR
48     0030  8E434852BD     36   >CHR BC GOTO "CHR="
53     0035  0A0792         37    LN 1938
56     0038  30E3           38    JS EXPR
58     003A  E0             39    BE *
59     003B  0B             40    DS
60     003C  2E             41    US
61     003D  0C             42    SP
62     003E  1D             43    NX
63     003F  9447CF         44   >GOTO BC PRNT "GO"
66     0042  8854CF         45    BC GOSB "TO"
69     0045  30E3           46    JS EXPR
71     0047  E0             47    BE *
72     0048  10             48    SB
73     0049  11             49    RB
74     004A  16             50    GO
75     004B  805355C2       51   >GOSB BC * "SUB"
79     004F  30E3           52    JS EXPR
81     0051  E0             53    BE *
82     0052  14             54    GS
83     0053  16             55    GO
84     0054  9050D2         56   >PRNT BC SKIP "PR"
87     0057  83494ED4       57    BC P0 "INT"
91     005B  E5             58   >P0 BE P3
92     005C  71             59    BR P6
93     005D  88BB           60   >P1 BC P4 ";"
95     005F  E1             61   >P2 BE P3
96     0060  1D             62    NX
97     0061  8FA2           63   >P3 BC P7 '"'
99     0063  21             64    PQ
100    0064  58             65    BR P1
101    0065  6F             66   >SKIP BR IF
102    0066  83AC           67   >P4 BC P5 "1↑"
104    0068  22             68    PT
105    0069  55             69    BR P2
106    006A  83BA           70   >P5 BC P6 "z↑"
108    006C  2493           71    PC "S↑"
110    006E  E0             72   >P6 BE *
111    006F  23             73    NL
112    0070  1D             74    NX
113    0071  30E3           75   >P7 JS EXPR
115    0073  20             76    PN
116    0074  48             77    BR P1
117    0075  9149C6         78   >IF BC INPT "IF"
120    0078  30E3           79    JS EXPR
122    007A  319B           80    JS RELO
124    007C  30E3           81    JS EXPR
126    007E  84544845CE     82    BC I1 "THEN"
131    0083  1C             83   >I1 CP
132    0084  1D             84    NX
133    0085  380D           85    JP STMT
135    0087  9D494E5055D4   86   >INPT BC RETN "INPUT"
                            87   * "INPUT" "TEXT" VAR
141    008D  82A2           88   >I15 BC I2 '"'
143    008F  21             89    PQ
144    0090  A0             90   >I2 BV *
145    0091  10             91    SB
146    0092  E7             92    BE I4
147    0093  243F2091       93   >I3 PC "? Q↑"
151    0097  27             94    GL
152    0098  E1             95    BE I4
153    0099  59             96    BR I3
154    009A  81AC           97   >I4 BC I5 "1↑"
156    009C  30E3           98   >I5 JS EXPR
158    009E  13             99    SV
159    009F  11            100    RB
160    00A0  82AC          101    BC I6 "1↑"
162    00A2  4D            102    BR I2
163    00A3  E0            103   >I6 BE *
164    00A4  1D            104    NX
165    00A5  895245545552CE 105   >RETN BC END "RETURN"
172    00AC  E0            106    BE *
173    00AD  15            107    RS
174    00AE  1D            108    NX
175    00AF  85454EC4      109   >END BC LIST "END"
179    00B3  E0            110    BE *
180    00B4  2D            111    WS
181    00B5  984C4953D4    112   >LIST BC RUN "LIST"
186    00BA  EC            113    BE L2
```

```
187  00BB  24000000000A80  114 >L1 PC "@↑@↑@↑@↑J↑@↑"
194  00C2  1F              115  LS
195  00C3  2493            116  PC "S↑"
197  00C5  23              117  NL
198  00C6  1D              118  NX
199  00C7  30E3            119 >L2 JS EXPR
201  00C9  E1              120  BE L3
202  00CA  50              121  BR L1
203  00CB  80AC            122 >L3 BC * "1↑"
205  00CD  59              123  BR L2
206  00CE  855255CE        124 >RUN BC CLER "RUN"
210  00D2  380A            125  JP XEC
212  00D4  844E45D7        126 >CLER BC REM "NEW"
216  00D8  2B              127  MT
217  00D9  845245CD        128 >REM BC DFLT "REM"
221  00DD  1D              129  NX
222  00DE  A0              130 >DFLT BV *
223  00DF  80BD            131  BC * "="
225  00E1  3814            132  JP LET
227  00E3  85AD            133 >EXPR BC E0 "-"
229  00E5  30FA            134  JS TERM
231  00E7  17              135  NE
232  00E8  64              136  BR E1
233  00E9  81AB            137 >E0 BC E4 "+"
235  00EB  30FA            138 >E4 JS TERM
237  00ED  85AB            139 >E1 BC E2 "+"
239  00EF  30FA            140  JS TERM
241  00F1  18              141  AD
242  00F2  5A              142  BR E1
243  00F3  85AD            143 >E2 BC E3 "-"
245  00F5  30FA            144  JS TERM
247  00F7  19              145  SU
248  00F8  54              146  BR E1
249  00F9  2F              147 >E3 RT
250  00FA  3109            148 >TERM JS FACT
252  00FC  85AA            149 >T0 BC T1 "*"
254  00FE  3109            150  JS FACT
256  0100  1A              151  MP
257  0101  5A              152  BR T0
258  0102  85AF            153 >T1 BC T2 "/"
260  0104  3109            154  JS FACT
262  0106  1B              155  DV
263  0107  54              156  BR T0
264  0108  2F              157 >T2 RT
265  0109  97524EC4        158 >FACT BC F0 "RND"
269  010D  0A8080          159  LN 32896
272  0110  12              160  FV
273  0111  0A0929          161  LN 2345
276  0114  1A              162  MP
277  0115  0A1A85          163  LN 6789
280  0118  18              164  AD
281  0119  13              165  SV
282  011A  0980            166  LB 128
284  011C  12              167  FV
285  011D  0B              168  DS
286  011E  3197            169  JS FUNC
288  0120  61              170  BR F1
289  0121  73              171 >F0 BR F2
290  0122  0B              172 >F1 DS
291  0123  02              173  SX 2
292  0124  04              174  SX 4
293  0125  02              175  SX 2
294  0126  03              176  SX 3
```

```
295  0127  05              177  SX 5
296  0128  03              178  SX 3
297  0129  1B              179  DV
298  012A  1A              180  MP
299  012B  19              181  SU
300  012C  0B              182  DS
301  012D  0906            183  LB 6
303  012F  0A0000          184  LN 0
306  0132  1C              185  CP
307  0133  17              186  NE
308  0134  2F              187  RT
309  0135  8F5553D2        188 >F2 BC F20 "USR"
313  0139  80A8            189  BC * "("
315  013B  30E3            190  JS EXPR
317  013D  3191            191  JS ARG
319  013F  3191            192  JS ARG
321  0141  80A9            193  BC * ")"
323  0143  2E              194  US
324  0144  2F              195  RT
                           196 * "AND(" EXPR1 " " EXPR2 ")"
325  0145  88414E44A8      197 >F20 BC F22 "AND("
330  014A  0A07A1          198  LN 1953
333  014D  67              199  BR F23
                           200 * "OR(" EXP1 " " EXPR2 ")"
334  014E  8E4F52A8        201 >F22 BC F26 "OR("
338  0152  0A07A8          202  LN 1960
341  0155  30E3            203 >F23 JS EXPR
343  0157  3191            204  JS ARG
345  0159  80A9            205 >F24 BC * ")"
347  015B  2E              206  US
348  015C  2F              207  RT
                           208 * "NOT(" EXPR ")"
349  015D  8B4E4F54A8      209 >F26 BC GET "NOT("
354  0162  0A07AF          210  LN 1967
357  0165  30E3            211 >F28 JS EXPR
359  0167  0B              212  DS
360  0168  50              213  BR F24
                           214 * "GET&(" ADDRESS EXPR ")"
361  0169  8F4745D4        215 >GET BC FRE "GET"
365  016D  85A6            216  BC GET1 "&"
367  016F  0A07B2          217  LN 1970
370  0172  63              218  BR CGET
371  0173  0A0114          219 >GET1 LN 276
374  0176  80A8            220 >CGET BC * "("
376  0178  4C              221  BR F28
                           222 * RETURNS FREE MEMORY  "PRINT FRE"
377  0179  8B4652C5        223 >FRE BC F3 "FRE"
381  017D  0922            224  LB 34
383  017F  12              225  FV
384  0180  0924            226  LB 36
386  0182  12              227  FV
387  0183  19              228  SU
388  0184  2F              229  RT
389  0185  A2              230 >F3 BV F4
390  0186  12              231  FV
391  0187  2F              232  RT
392  0188  C1              233 >F4 BN F5
393  0189  2F              234  RT
394  018A  80A8            235 >F5 BC * "("
396  018C  30E3            236 >F6 JS EXPR
398  018E  80A8            237  BC * "("
400  0190  2F              238  RT
401  0191  83AC            239 >ARG BC A0 "1↑"
```

FIGURE 3

```
00001                    NAM   TINYEXT  TINY BASIC EXTENSIONS
00002                    OPT   S,P
00003 *
00004 *****
00005 *
00006 * THE FOLLOWING MACHINE LANGUAGE ROUTINES
00007 * ARE NECESSARY TO IMPLEMENT THE FUNCTIONS:
00008 * AND, OR, NOT, GET&, SET&.
00009 *
00010 *****
00011 *
00012          001E   TEMP   EQU   $001E
00013 07A1            ORG   $07A1
00014 *
00015 *****
00016 *
00017 * AND(first argument,second argument)
00018 * THE first argument IS STORED IN THE INDEX REGISTER
00019 * AND MAY BE AN EXPRESSION
00020 * THE second argument IS STORED IN THE ACCUMULATORS
00021 * AND MAY BE AN EXPRESSION.
00022 *
00023 07A1 DF 1E   AND    STX   TEMP    FIRST ARGUMENT
00024 07A3 94 1F         AND A TEMP+1  SECOND ARGUMENT
00025 07A5 D4 1E         AND B TEMP
00026 07A7 39            RTS
00027 *
00028 *****
00029 *
00030 * OR(first argument,second argument)
00031 * THE first argument IS STORED IN THE INDEX REGISTER
00032 * AND MAY BE AN EXPRESSION.
00033 * THE second argument IS STORED IN THE ACCUMULATORS
00034 * AND MAY BE AN EXPRESSION.
00035 *
00036 07A8 DF 1E   OR     STX   TEMP    FIRST ARGUMENT
00037 07AA 9A 1F         ORA A TEMP+1  SECOND ARGUMENT
00038 07AC DA 1E         ORA B TEMP
00039 07AE 39            RTS
00040 *
00041 *****
00042 *
00043 * NOT(argument)
00044 * THE argument IS CONTAINED IN THE ACCUMULATORS AND
00045 * MAY BE AN EXPRESSION.
00046 *
00047 07AF 43   NOT      COM A
00048 07B0 53            COM B
00049 07B1 39            RTS
00050 *
00051 *****
00052 *
00053 * GET&( decimal address expression )
00054 * FETCHES TWO BYTES FROM MEMORY
00055 *
00056 07B2 A6 01   GETTWO LDA A 1,X
00057 07B4 E6 00         LDA B 0,X
00058 07B6 39            RTS
00059 *
00060 *****
00061 *
00062 * SET& decimal address expression = data expression
```

SYMBOL TABLE →

```
 403  0193  38E3          240  JP EXPR
 405  0195  0B            241 >A0 DS
 406  0196  2F            242  RT
 407  0197  80A8          243 >FUNC BC * "("
 409  0199  52            244  BR F6
 410  019A  2F            245  RT
 411  019B  84BD          246 >RELO BC R0 "="
 413  019D  0902          247  LB 2
 415  019F  2F            248  RT
 416  01A0  8EBC          249 >R0 BC R4 "<"
 418  01A2  84BD          250  BC R1 "="
 420  01A4  0903          251  LB 3
 422  01A6  2F            252  RT
 423  01A7  84BE          253 >R1 BC R3 ">"
 425  01A9  0905          254  LB 5
 427  01AB  2F            255  RT
 428  01AC  0901          256 >R3 LB 1
 430  01AE  2F            257  RT
 431  01AF  80BE          258 >R4 BC * ">"
 433  01B1  84BD          259  BC R5 "="
 435  01B3  0906          260  LB 6
 437  01B5  2F            261  RT
 438  01B6  84BC          262 >R5 BC R6 "<"
 440  01B8  0905          263  LB 5
 442  01BA  2F            264  RT
 443  01BB  0904          265 >R6 LB 4
 445  01BD  2F            266  RT
                              END

TOTAL ERRORS= 0
```

LABEL	DECIMAL	HEX
STRT	0	0000
L0	7	0007
XEC	10	000A
STMT	13	000D
LET	20	0014
SET	25	0019
SET1	35	0023
CSET	38	0026
CHR	48	0030
GOTO	63	003F
GOSB	75	004B
PRNT	84	0054
P0	91	005B
P1	93	005D
P2	95	005F
P3	97	0061
SKIP	101	0065
P4	102	0066
P5	106	006A
P6	110	006E
P7	113	0071
IF	117	0075
I1	131	0083
INPT	135	0087
I15	141	008D
I2	144	0090
I3	147	0093
I4	154	009A
I5	156	009C
I6	163	00A3
RETN	165	00A5
END	175	00AF
LIST	181	00B5
L1	187	00BB
L2	199	00C7
L3	203	00CB
RUN	206	00CE
CLER	212	00D4
REM	217	00D9
DFLT	222	00DE
EXPR	227	00E3
E0	233	00E9
E4	235	00EB
E1	237	00ED
E2	243	00F3
E3	249	00F9
TERM	250	00FA
T0	252	00FC
T1	258	0102
T2	264	0108
FACT	265	0109
F0	289	0121
F1	290	0122
F2	309	0135
F20	325	0145
F22	334	014E
F23	341	0155
F24	345	0159
F26	349	015D
F28	357	0165
GET	361	0169
GET1	371	0173
CGET	374	0176
FRE	377	0179
F3	389	0185
F4	392	0188
F5	394	018A
F6	396	018C
ARG	401	0191
A0	405	0195
FUNC	407	0197
RELO	411	019B
R0	416	01A0
R1	423	01A7
R3	428	01AC
R4	431	01AF
R5	438	01B6
R6	443	01BB

```
1) ASSEMBLER PASS 1
2) ASSEMBLER PASS 2
3) LIST SOURCE FILE
?
OK
```

```
00063 * STORES TWO BYTES IN MEMORY AT LOCATION
00064 * POINTED TO BY THE INDEX REGISTER.
00065 *
00066 07B7 A7 01    SETTWO STA A 1,X
00067 07B9 E7 00           STA B 0,X
00068 07BB 39              RTS
00069                      END

TEMP    001E 00030
AND     07A1 01953
OR      07A8 01960
NOT     07AF 01967
GETTWO  07B2 01970
SETTWO  07B7 01975

TOTAL ERRORS 00000


ENTER PASS: 0R, 0C, 1P, 1S, 2L, 2T, E, X? 2T
S00B00004E594558542054499F
S11E07A1DF1E941FD41E39DF1E9A1FDA1E39435339A601E60039A701E700391A
S9030000FC


TOTAL ERRORS 00000
ENTER PASS: 0R, 0C, 1.., 1S, 2L, 2T, E, X? E

*MODIFIED EDITOR 1.1*
```

FIGURE 4

```
 D  ?07A0 ?097F
07A0   39 DF 1E 94 1F D4 1E 39 DF 1E 9A 1F DA 1E 39 43
07B0   53 39 A6 01 E6 00 39 A7 01 E7 00 39 24 3E 91 27
07C0   10 E1 59 C5 2A 56 10 11 2C 8B 4C 45 D4 A0 80 BD
07D0   30 E3 E0 13 1D 96 53 45 D4 85 A6 0A 07 B7 63 0A
07E0   01 18 30 E3 80 BD 30 E3 E0 2E 0C 1D 8E 43 48 52
07F0   BD 0A 07 92 30 E3 E0 0B 2E 0C 1D 94 47 CF 88 54
0800   CF 30 E3 E0 10 11 16 80 53 55 C2 30 E3 E0 14 16
0810   90 50 D2 83 49 4E D4 E5 71 88 BB E1 1D 8F A2 21
0820   58 6F 83 AC 22 55 83 BA 24 93 E0 23 1D 30 E3 20
0830   48 91 49 C6 30 E3 31 9B 30 E3 84 54 48 45 CE 1C
0840   1D 38 0D 9D 49 4E 50 55 D4 82 A2 21 A0 10 E7 24
0850   3F 20 91 27 E1 59 81 AC 30 E3 13 11 82 AC 4D E0
0860   1D 89 52 45 54 55 52 CE E0 15 1D 85 45 4E C4 E0
0870   2D 98 4C 49 53 D4 EC 24 00 00 00 00 0A 80 1F 24
0880   93 23 1D 30 E3 E1 50 80 AC 59 85 52 55 CE 38 0A
0890   84 4E 45 D7 2B 84 52 45 CD 1D A0 80 BD 38 14 85
08A0   AD 30 FA 17 64 81 AB 30 FA 85 AB 30 FA 18 5A 85
08B0   AD 30 FA 19 54 2F 31 09 85 AA 31 09 1A 5A 85 AF
08C0   31 09 1B 54 2F 97 52 4E C4 0A 80 80 12 0A 09 29
08D0   1A 0A 1A 85 18 13 09 80 12 0B 31 97 61 73 0B 02
08E0   04 02 03 05 03 1B 1A 19 0B 09 06 0A 00 00 1C 17
08F0   2F 8F 55 53 D2 80 A8 30 E3 31 91 31 91 80 A9 2E
0900   2F 88 41 4E 44 A8 0A 07 A1 67 8E 4F 52 A8 0A 07
0910   A8 30 E3 31 91 80 A9 2E 2F 8B 4E 4F 54 A8 0A 07
0920   AF 30 E3 0B 50 8F 47 45 D4 85 A6 0A 07 B2 63 0A
0930   01 14 80 A8 4C 8B 46 52 C5 09 22 12 09 24 12 19
0940   2F A2 12 2F C1 2F 80 A8 30 E3 80 A9 2F 83 AC 38
0950   E3 0B 2F 80 A8 52 2F 84 BD 09 02 2F 8E BC 84 BD
0960   09 03 2F 84 BE 09 05 2F 09 01 2F 80 BE 84 BD 09
0970   06 2F 84 BC 09 05 2F 09 04 2F 7E 07 6A 7E 07 8D
 ?
```

Microcomputers

...as they should be

BY OLAV NAESS
Welhavensgt. 65
Bergen, Norway

The microcomputer has great potental for interacting with the real world as well as for expanding into a large computer system by addition of various storage units, computing units, interface units, etc.

But the microcomputer systems of today are far from able to realize the potential for interfacing and expansion, or at least they implement such potential in an unnecessarily awkward way. It seems to me that the main reason for this is bad mechanical design. The mechanical problems evidently tend to escape attention. I will therefore try to throw some light upon these problems, and suggest solutions for them. The intention behind this article is to create widespread dissatisfaction with present designs, so that those with powers to bring about such reforms will be induced to get moving.

Now, let's take a look at the common mechanical design of a non-trivial microcomputer, i.e., one that can be expanded to a multi-module system. The modules, each normally one PC (printed circuit) card, are usually plugged into connectors which are situated in fixed positions on a rigid motherboard, and the whole assembly is mounted in a box. The box is a typical afterthought. It **does** protect the cards against dust and damage, but also hinders more constructive interaction between the computer and its environment. The few simple controls and connectors that can be placed on the edge of a PC card are difficult to use when the computer is on, even if the cabinet is opened.

Cables to the outer world must find their way out of some opening in a rather disorderly manner, and heaven knows what traumatic experience the modules are exposed to as some creature passing by in the external world gets entangled in one of the cables.

In my own mixed X-100 system, the keyboard communicates with the internal video terminal card through a flat cable that plugs into DIL (dual in-line; IC) sockets on each unit's PC card. There was obviously no standard for such a connection, so I had to identify the signal lines and resolder one of the DIL plugs. The crossing wires on this plug made it impossible to close its lid, which had to be left floating over the plug. If the keyboard is moved, the flat cable sweeps over the components of the video card. Disconnection of the keyboard necessitates opening the computer's cabinet, unplugging the video card, and then carefully wriggling that fragile DIL plug out of its socket. Those who enjoy dancing with an octopus can try to move the computer system while the external units are connected.

On the top of the video card sits an edge connector to whose solder lugs various other external connections are soldered, amongst them the video cable! This thick, rigid coaxial cable should certainly not be allowed to sweep across the interior of a computer and exert its powers upon some poor little solder lugs. Breaks and shorts have again and again caused a blank screen in my system. Conclusion: Video cables can only be connected with (coaxial) connectors properly mounted on a rigid external surface. Letting external connections go directly to the PC card is a dubious practice.

More thoroughly designed computers employ cable assemblies divided into an internal and external cable, the two meeting in a connector pair on a cabinet surface. This is safe and covenient, but an expensive solution which also imposes restrictions upon the types and number of connectors that can be used.

Is there any solution to our dilemma? Yes, and module systems with the desired properties are even commonly used in all sorts of laboratories. A PC card for such a system has a narrow plate mounted vertically on the card, running along the card edge that is opposite to the edge connector. When the cards are plugged into the system's frontless enclosure, these narrow plates, whose two ends may be screwed directly to the enclosure, constitute a front panel. All connectors, switches, indicators, etc., needed by a module may thus be mounted on the module's own piece of front panel, and their connections with the PC card can be short and fixed.

This packaging system does, however, still have some annoying shortcomings: the enclosure has a fixed size which is quite likely to be too small or too large. And modules are not accessible for testing and adjustments unless they are unplugged.

I will therefore suggest a further improvement: the front panel plate should go all the way around the PC card, as shown in Figure 1 and further illustrated in Figure 2. That is: the cards are mounted in frames which are attached to each other by separable hinges. When end-plates are hinged to the first and last module, the modules are completely enclosed: no space is wasted, and the enclosure can be expanded to any size. Such an assembly can be opened like a book during operation for testing or adjustments.

A hinged-frame microcomputer, which could have frames designed to hold S-100 cards, should have a flexible motherboard passing inside the hinged edges (which might be detachable from the side edges, too). Such a motherboard would restrict expandability somewhat, but the unused part could run along one or both endplates. Modules hinged together this way may be virtually anything from a framed PC card to large units like a power supply, a disk drive or even a video monitor. A frame for PC cards could have several small cards mounted side by side—useful for sub-modules with perhaps just one LSI IC. Heat-generating components could be mounted in perforated frames, and circuits requiring screening could have screening plates adjacent to their frames.

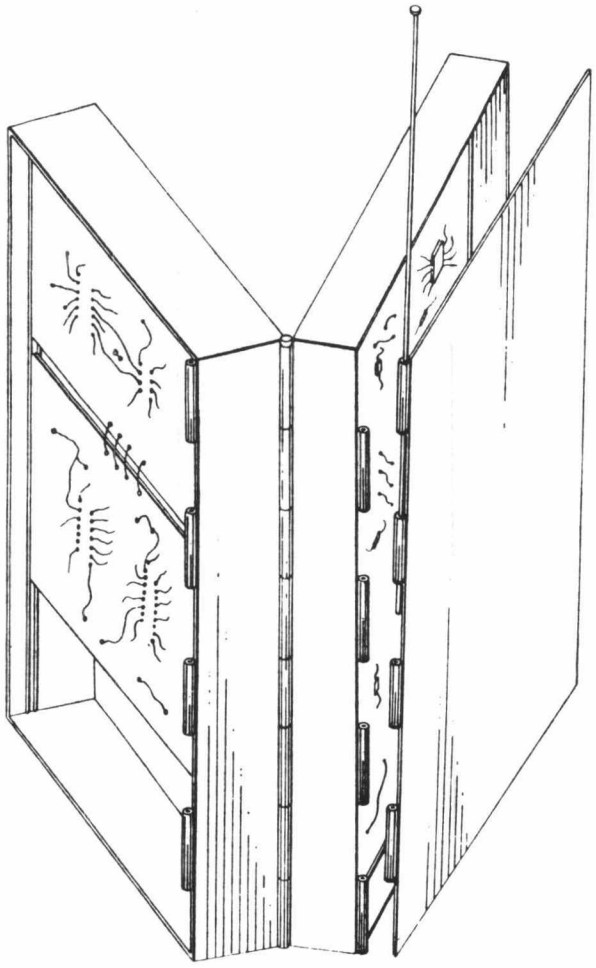


Figure 1. Hinged frames with PC cards, and one detached end plate.

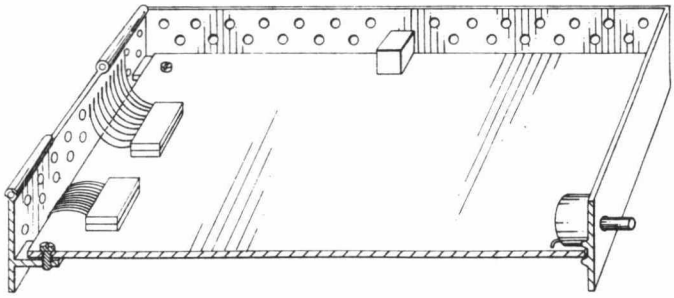


Figure 2. Section of frame-mounted PC card with controls and flat cable connections.

Frame-mounted cards are less vulnerable when removed from the system than unmounted cards are. If frame dimensions are standardized, cooperating systems like a computer and an audio system can be merged to a single unit.

Figure 3 shows what a large hinged-frame system may look like. The keyboard (in a wedge-shaped module) is here opened 180 degrees relative to the next module.

A small computer may have just one or a few thin modules just below the keyboard module, and resemble a TRS-80. If the user of such a small computer wanted to use control panels designed for special applications, he could mount them like a PC card in a frame anywhere in the computer.

Imagine sitting down in your easy-chair with your book-like microcomputer in your lap. You leaf through pages of control panels until you find one for a game you would like to play . . .

Enough about the enclosure; now let us look at the external cables and how signals are allocated in them.

Regarding external connections, there prevails an unfortunate tendency towards unnecessary splitting-up; by using separate cable assemblies for the various types of signals and power, and by splitting multiplexed (bus) signals into separate digital ports or analog channels at a too central site. The circuitry required for decoding/latching signals from a bus is cheap compared with a decent cable assembly. A universal I/O bus which can be extended out into the real world can greatly increase the controlling power of the computer, decrease costs, and decrease confusion caused by lack of standardization. The IEEE-488 bus is such a bus for peripheral units but is too specialized. Besides, its communication protocol is rather complex.

To illustrate my notion of universality, I would like to describe a universal I/O bus system I have just constructed. It communicates through a 25-conductor flat cable with standard DB25 connectors, but a stackable connector would be preferable. The conductors carry:

1: Power from the computer: 5V stabilized. 8V and ±18 V unstabilized, plus ground.
2: The 2 MHz clock from the X-100 bus.
3: Two analog lines: Output from a D/A converter and input to an A/D converter (placed between filtered power lines).
4: Count and gate inputs to two counter-timers (Intel 8253).
5: An interrupt input.
6: An 8-bit digital data/address bus with control lines.

(CLOCK, $\overline{\text{ADDRESS COMES}}$)

Even though hundreds of external units can be addressed, the I/O system occupies only a sequence of four ordinary I/O ports:

First port: Address of the external unit (output).
Second port: Data to/from the external digital data bus.
Third port: Least significant byte to the D/A converter or from the A/D converter.
Fourth port: Most significant byte to the D/A converter or from the A/D converter.

(Continued on page 14.)

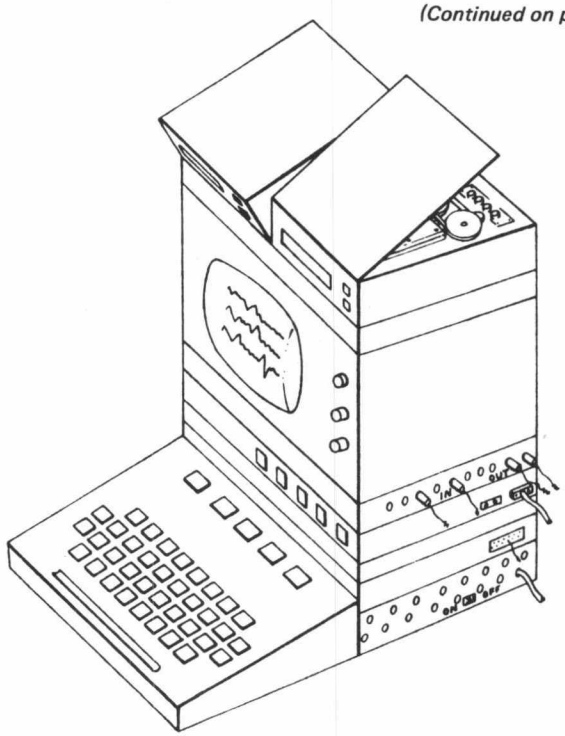


Figure 3. Small computer made up from hinged-frame modules has one cabinet rather than four.

(Continued from page 13.)

Ports required for the counter/timer and any interrupt controller are additional.

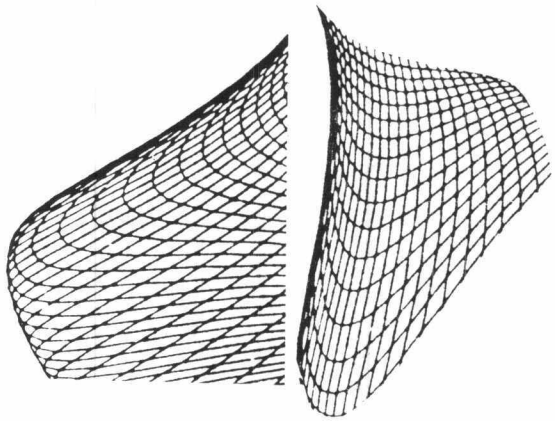
To operate the bus, an I/O transfer is initiated by outputting an external address through the first port. When this one-byte address goes out the external bus, the ADDRESS COMES line goes active (low), and CLOCK goes active (high). This activates the external address decoders and loads shift registers associated with these address decoders. Each parallel output of these shift registers will activate an input or output data transfer if it has a 0 (low). When an address decoder reports an address match, it loads a 0 into a certain shift register bit. All the other shift registers along the bus are filled with non-activating 1's. The transfer activated by a 0 may be in or out, digital or analog:

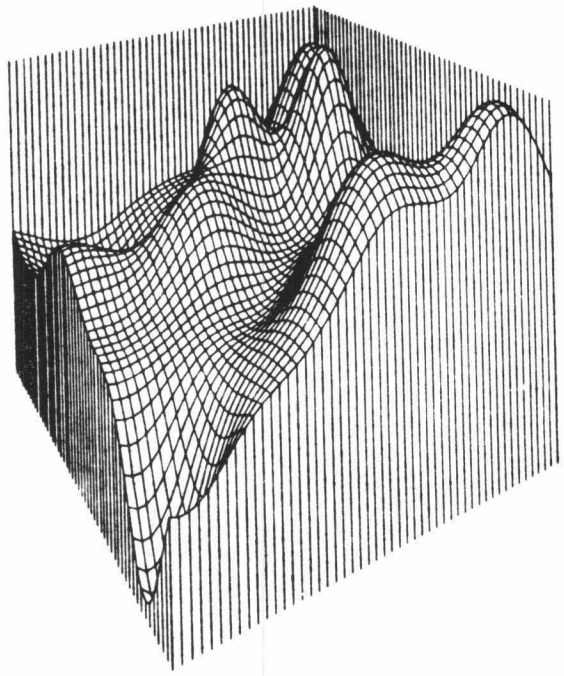
Digital in: An eight bit tri-state buffer puts its data on the external bus.
Analog in: An analog switch connects an analog input to the analog input line.
Digital out: An eight bit latch takes data from the external bus.
Analog out: A sample-and-hold circuit samples the analog output line (analog outputs need periodic refreshing).

Between each such external transfer the data byte is read or written through the second, third, or fourth port. Immediately after an I/O through the second or fourth port, a pulse emitted on the CLOCK line causes a shift in all the peripheral shift registers (with a 1-bit entering), so that the next transfer is activated by the 0-bit in the active shift register (besides, an A/D conversion is started).

The external address can indicate the entrypoint anywhere in a sequence of assorted-type transfers. Because time-consuming processes like A/D conversion and stabilization of long signal lines occur between the computer's I/O-operations, the computer is never forced to wait provided the intervals between I/O-operations don't become too short, like during DMA bursts.

Well, folks: if these things sound interesting, don't expect real products or properly tested designs to come from an amateur like me. I can only hope that manufacturers can be enticed to move along such lines, and that they will establish useful standards.





BY TOR SJØWALL
G-15
N-7035 Moholt
Norway

CP/M TO UCSD PASCAL file conversion

We were all very pleased with the article on PASCAL to CP/M file conversion in *Dr. Dobb's*, but there is also a need to go the other way; to use CP/M 2.0 files in the PASCAL II.0 environment.

In order to get at the CP/M files, you have to access the diskette on a direct track/sector basis because of the difference betweeen CP/M and PASCAL file formats. This is not possible under the PASCAL system as described in UCSD's manual but there is something they have not told us about the UNITREAD and UNITWRITE intrinsic procedures. The parameters to these procedures are, according to the manual:

procedure UNITREAD (unitnumber, array, length, blocknumber integer);

Unitnumber	integer name of an I/O device
Array	packed array for transfer
Length	number of bytes to transfer
Blocknumber	absolute PASCAL block number on a diskette
Integer	optional (1 indicates async transfer)

Disassembly shows that the integer parameter has more options

1. Asynchronous I/O, not implemented in our 8080 system
2. Direct track/sector diskette I/O
4. Text file indent code expansion on output activated
8. Automatic linefeed append after return

Values may be added to combine options.

4 & 8 only work on CONSOLE:, PRINTER: and REMOUT:.

Direct track/sector I/O do not use the ordinary blocknumber parameter. Instead a blocknumber is calculated as:

track: 0 . . 76 ; sector : 1 . . 26 ;
blocknumber : = (track * 26) + (sector – 1);

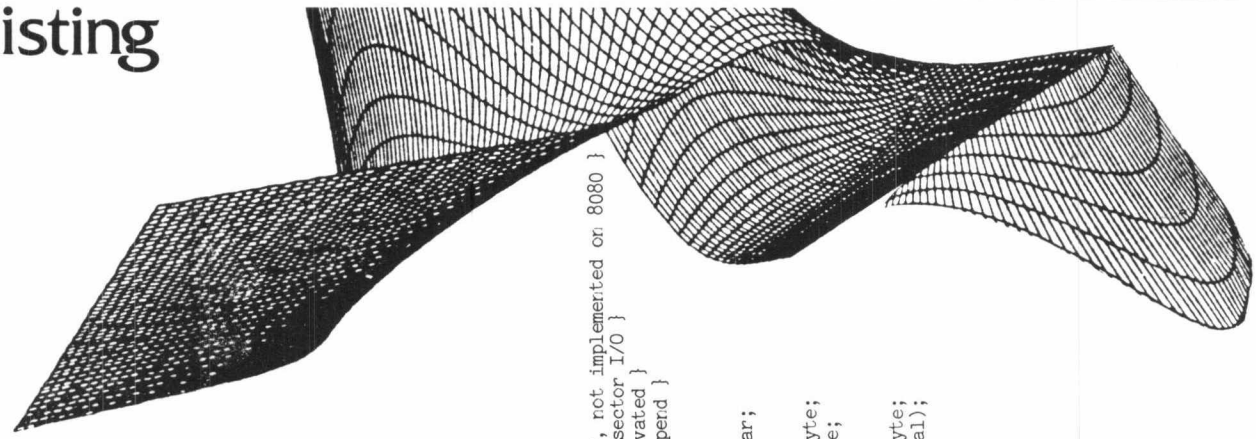
NOTE: Length parameter is ignored
Array parameter has to be at least 128 bytes (1 sector)
IORESULT is never set, always 0

I don't know if this option will work on all PASCAL systems, but it is likely since it is the only way to get at the bootloader on the diskette. The P2-code interpreter addresses in our implementation are: unitwrite 22A8H and unitread 22ADH, with direct option at 230CH.

The GETCPM program uses the direct sector option to convert CP/M textfiles to PASCAL .TEXT files. Datafile conversion is also supported. Random access CP/M files are not properly converted, but this is easily implemented in COPYPAGE by sending 1K of null or space when cpmpage=0.

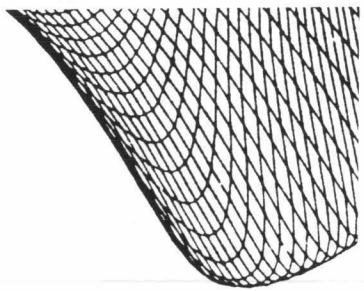
Tor Sjowall is a student at the Norwegian Institute of Technology who will soon be a professional engineer. Tor works mostly on microcomputer systems but his main field is digital switching in telephone networks.

listing



```
                UCSD  pascal compiler
{$L-}
program GETCPM ;

const void = 229;       { 0E5H }

{ ASCII control characters }
    cpmeof = 26;    { control Z }
      cr = 13;
      lf = 10;
     tab = 9;
    null = 0;

{ diskette drive unit numbers }
    drivea = 4;
    driveb = 5;

{ unitread / unitwrite attributes }
     async = 1;      { asynchronous I/O, not implemented on 8080 }
   sectdir = 2;      { direct diskette sector I/O }
    indent = 4;      { indent code activated }
   autolf = 8;       { auto linefeed append }

type nametype = packed array [0..10] of char;
     namestring = string [30];
     byte = 0..255;
     dirtype = packed array [0..2047] of byte;
     maptype = packed array [0..15] of byte;
     nontype = 'N'..'r';
     illetype = ' '..']';
     secttype = packed array [0..127] of byte;
     legaltypes = (textfile,datafile,illegal);

var iscpm : boolean;
    filetype : legaltypes;
    sourcename,destname : namestring;
    nono : set of nontype;
    illechar : set of illetype;
    filename : nametype;
    textdest : text;
    datadest : file of secttype;
    directbuf : dirtype;

function UPPERCASE (inchar : char) : char;      { convert to uppercase letters }

begin
  if inchar in ['a'..'z']
    then uppercase := chr (ord (inchar) - 32)
    else uppercase := inchar;
end;
```

```
procedure CHECKTYPE (var filename : namestring;
                     var filetype : legaltype);
                      { check for .TEXT or .DATA filetype and system diskette }
var i : integer;
    help : namestring;

begin
  for i := 1 to length (filename) do
    filename [i] := uppercase (filename [i]);
  if length (filename) <> 0 then
    if filename [1] <> '*' then
      filename := concat ('*',filename);
  help := filename;
  delete (help,1,length (help) - 5);
  filetype := illegal;
  if help = '.TEXT' then filetype := textfile;
  if help = '.DATA' then filetype := datafile
end;
```



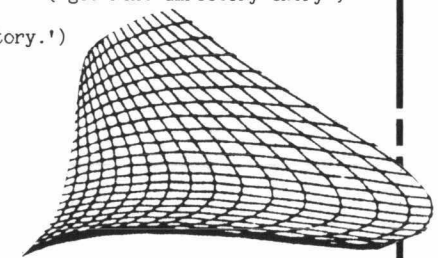
```
procedure CPMNAME (var wantname : nametype;
                   wantfile : namestring);
                                          { read and unpack CP/M filename }
var i,j : integer;

begin
  fillchar (wantname,11,' ');
  i := 1; j := 0;
  while ((j <= 10) and (i <= length (wantfile))) do
    begin
     if not (wantfile [i] in illechar)
        then wantname[j] := uppercase (wantfile [i])
        else if (wantfile [i] = '.') and (j <= 8) then j := 7 else j := 10;
      j := j+1; i := i+1
    end
end;
```

```
procedure WRITENAME (filename : nametype);
                                          { pack and print CP/M filename }
var i : integer;

begin
  i:=0;
  while (i<8) and (filename[i]<>' ') do
    begin
      write (filename[i]); i:=i+1
    end;
  write ('.');
  for i:=8 to 10 do if filename[i]<>' ' then write (filename[i])
end;
```

```
function BLOCKNUM (seqsect : integer) : integer;
                      { calculate blocknumber from sequential sector number }
begin
  if (seqsect mod 26) + 1 <= 13
    then blocknum := (26 * (seqsect div 26)) + ((seqsect mod 26) * 6) mod 26
    else blocknum := (26 * (seqsect div 26)) + ((seqsect mod 26) * 6) mod 26 + 1
end;
```

```
procedure DIRECTORY (var directbuf : dirtype;
                     var iscpm : boolean);
                        { check CP/M format and read directory into directbuf }

var i,seqsect : integer;

begin
  seqsect := 2*26;
  iscpm := true;
                              { read first sector and check if size is 128 bytes }
  for i := 128 to 150 do directbuf [i] := i;
  unitread (driveb,directbuf,128,blocknum (seqsect),sectdir);
  for i := 128 to 150 do iscpm := iscpm and (directbuf [i] = i);
                     { check for legal filename and extension 0 for first file }
  iscpm := iscpm
       and (directbuf [1] >= ord (' '))
       and (directbuf [1] < 128)
   and not (chr (directbuf [1] mod 128) in illechar)
       and (directbuf [12] = 0);
                          { read remaining 15 sectors in directory if CP/M format }
  if iscpm then for i := 1 to 15 do
    unitread (driveb,directbuf [i*128],128,blocknum (seqsect+i),sectdir)
end;
```

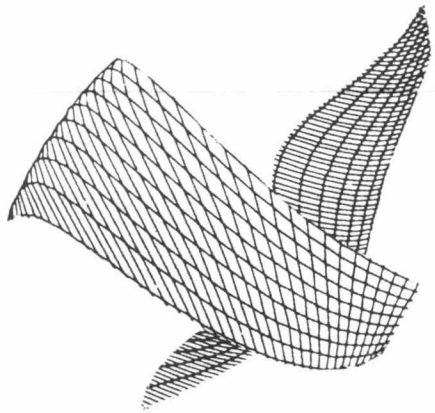
```
procedure PRINTDIRECT (directbuf : dirtype);
                                                        { print CP/M directory }
var i,dirbufpoint,filecount : integer;

begin
  writeln ('CP/M directory listing :   (<user> <name>.<type>)');
  dirbufpoint := 0;
  filecount := 0;
  repeat
    if (directbuf [dirbufpoint] <> void) and (directbuf[dirbufpoint+12]=0)
    then                                    { skip if void or not first extension }
      begin
        write (directbuf [dirbufpoint] :2,'  ');
        for i := 1 to 8 do
          write (chr (directbuf [i+dirbufpoint] mod 128));
        write (' ');
        for i := 9 to 11 do
          write (chr (directbuf [i+dirbufpoint] mod 128));
        filecount := filecount + 1;
        if (filecount mod 4) = 0 then writeln else write ('  |  ')
      end;
    dirbufpoint := dirbufpoint+32                  { get next directory entry }
  until dirbufpoint >= 2048;
  writeln; writeln (filecount,' files in directory.')
end;
```



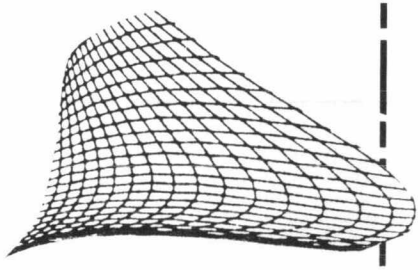
```
procedure SCANDIRECT (var diskmap : maptype;
                      var found : boolean;
                      var size : byte;
                          dirbuf : dirtype;
                          filename : nametype;
                          extension : byte);
                                { scan directory buffer for filename/extension
                                return array with disk map of file if found  }
var dirbufpoint,i : integer;
```

```
    namebuf : nametype;

begin
found := false;
fillchar (namebuf,11,' ');
dirbufpoint := 0;
                                                { scan directory for file }
repeat
  if dirbuf [dirbufpoint] <> void then                    { skip if void }
    begin
      for i := 0 to 10 do
        namebuf [i] := chr (dirbuf [dirbufpoint + i + 1] mod 128);
      found := (namebuf = filename) and (extension = dirbuf [dirbufpoint+12]);
    end;
  dirbufpoint := dirbufpoint + 32;
until found or (dirbufpoint >= 2048);
                                          { transfer disk map information }
if found then
  begin
    dirbufpoint := dirbufpoint - 32;
    size := dirbuf [dirbufpoint + 15];
    for i := 0 to 15 do
      diskmap [i] := dirbuf [dirbufpoint + i + 16]
  end
end;



procedure COPYFILE (filename : nametype;
                    filetype : legaltypes);
                                        { copy a CP/M file to a Pascal file }
var linepos : integer;
    size,extension : byte;
    found : boolean;
    diskmap : maptype;



procedure COPYSECT (sectbuf : secttype);
                                          { copy one sector of data file }

begin
  datadest^ := sectbuf;
  put (datadest)
end;



procedure COPYCHAR (inchar : byte);
                                    { copy one character if printable }
begin
  inchar := inchar mod 128;
  if inchar >= 32
    then
      begin
        linepos := linepos + 1;
        write (textdest,chr (inchar))
      end
    else
      begin
                                                { legal control characters }
        case inchar of
          tab     : begin
                      repeat
                        write (textdest,' ');
                        linepos := linepos + 1
                      until (linepos mod 8) = 0;
                      exit (copychar)
                    end;
          cr      : begin
                      writeln(textdest);
                      linepos := 0;
                      exit (copychar)
                    end;
          lf,null : exit (copychar);
          cpmeof  : begin
                      writeln;
                      writeln ('End of file');
                      exit (copyfile)
                    end
        end;  { case }
                                                  { illegal control characters }
        write (textdest,'?');
        linepos := linepos + 1
      end
end;


procedure COPYPAGE (valsects : integer;
                    cpmpage : byte);
                                                  { copy one CP/M file page, 1K }
var seqsect,j,temp : integer;
    sectbuf : secttype;

begin
  if cpmpage <> 0 then
  begin
    seqsect := (cpmpage * 8) + (2 * 26);
    temp := seqsect + valsects;
    while seqsect < temp do
    begin
      unitread (driveb,sectbuf,128,blocknum (seqsect),sectdir);
      seqsect := seqsect + 1;
      if filetype = textfile
        then for j :=0 to 127 do copychar (sectbuf [j])
        else copysect (sectbuf)
    end
  end
end;



procedure COPYEXTENSION (diskmap : maptype;
                         size : byte);
                                  { copy content of file extension at diskmap }
var i : integer;

begin
                                                        { copy all whole pages }
  for i := 0 to (size div 8) - 1 do copypage (8,diskmap[i]);
                                                      { copy last partial page }
  if (size mod 8) <> 0 then copypage (size mod 8,diskmap[size div 8])
end;


begin  { COPYFILE }
extension := 0;
linepos := 0;
```

```
repeat                                                  { find file extensions }
  scandirect (diskmap,found,size,directbuf,filename,extension);
  if found
  then
    begin
      if (extension = 0) and (size = 0)
        then writeln ('File is empty')
        else copyextension (diskmap,size);
      extension := extension + 1
    end
until not found;

if extension = 0
then                                                    { no file matched name }
  begin
    writename (filename);
    writeln (' not found')
  end
else                                                    { file transfer ok }
  begin
    writeln; writeln ('End of file')
  end
end; { copyfile }




{ MAIN PROGRAM }

begin
  illechar := [' ','<','>',',',',','.',';',':','=','?','*','[',']'];
  nono := ['N','n'];
```



```
                                                        { presentation }
page (output);
writeln; writeln;
writeln ('CP/M to PASCAL file transfer     19800509');
writeln;
writeln ('PASCAL diskette in drive A');
writeln ('CP/M   diskette in drive B');
writeln;
write ('Press <return> when ready '); readln; writeln;
                    { check if CP/M diskette and read directory into buffer }
directory (directbuf,iscpm);
if iscpm
then
  begin
    printdirect (directbuf);
    writeln;
    write ('Source CP/M file        : ');
    readln (sourcename);
    if length (sourcename) = 0 then exit (program);
    cpmname (filename,sourcename);
    filetype := illegal;
    repeat
      write ('Destination Pascal file : ');
      readln (destname);
      if length (destname) = 0 then exit (program);
      checktype (destname,filetype);
      if filetype = illegal
        then writeln ('Illegal file name. (.TEXT or .DATA)')
    until filetype <> illegal;
    if filetype = textfile
    then
      begin
        writeln; writeln ('TEXT file transfer');
        rewrite (textdest,destname);
        copyfile (filename,filetype);
        close (textdest,lock)
      end
    else
      begin
        writeln; writeln ('DATA file transfer');
        rewrite (datadest,destname);
        copyfile (filename,filetype);
        close (datadest,lock)
      end
  end
else writeln ('Not CP/M floppy in drive B')             { iscpm was false }
end.
```



Extended Source-Text Area and Label Capability for SC/MP NIBASM

BY W. F. LONG
Naval Medical Research Institute
Bethesda, Maryland 20014

Are you using Roger Marin's SC/MP assembler (SL0060) written in NIBL BASIC[1] ? Would you like to be able to expand the source-text RAM area beyond one 4K page for larger programs? Do you find that 26 labels aren't enough or that you need to begin two with the same alphabetic character? If you answered "yes," read on.

The following modifications are simple and brief. Together, they affect four existing lines and add one new line. Separately, the two extensions affect different lines and do not interact. They may be added singly if desired.

Expanded NIBASM reads source text across any allowable number of 4K pages consecutively from the starting address. (NIBL defines page 7 as the upper limit and you must remember to save space for generated object code.) Label capacity is doubled by permission of the entire lower-case alphabet as beginning characters.

Source-Text Area Expansion

Existing program line 120 begins the text search for each line of source:

```
120X=@L*256+@(L+1):N=L+4:IF(X=-1)OR(W=4)PR"":PR"END ASM":END
```

where variable L points to the current line number stored in NIBL format; N is the text cursor; and W is the pass counter. In the first statement, variable X is set to the current line number converted from base 16 to base 10. The first test in statement three looks for X equal to the NIBL program-page-end flag. A true result terminates the assembler.

Program line 120 is rewritten to cross 4K page boundaries with NIBASM:

```
120X=@L*256+@(L+1):N=L+4:IF@(L+2)=#FF L=(#0FFF OR L)+3:GOTO120
```

where the conditional portion of the line has been replaced. Now the test is for a new user-set flag in the byte following the two line-number-storage locations. If the flag tests true, variable L is advanced to the first line-number address of the following RAM page, and line 120 is re-interpreted with the new L value.

New line 130 tests for source-text end with the omitted portion of old line 120:

```
130IF(X=-1)OR(W=4)PR"":PR"END ASM":END
```

NIBL shares the new NIBASM user-flag position described above without interference NIBL uses it within program text as the line length indicator, storing a maximum possible value of 48H. At program-page end, it is the address of the NIBL function TOP.

Label Capacity Expansion

Original NIBASM scratchpad-RAM occupies the top 200 bytes of page 1, from 1F 38H to 1FFFH. The label-table takes 52 bytes (2 bytes per upper-case alphabetic character) from 1F3EH to 1F71H. Variable I is designated the label-table pointer in program line 20:

```
20PR"SC/MP ASM":PR"OBJCT CODE RAM=";:INPUTD:H=8100:I=7998:W=1
```

where 7998 is the first character-A address in base 10. All but the last two statements of line 30 comprise a label-table initialization loop:

```
30FORX=0TO51:@(I+X)=#FF:NEXTX:U=8050:Z=-1
```

After the first statement of line 200 normalizes the beginning character value of a suspected label, statement two tests character nonvalidity:

```
200X=(@N-65)*2:IF(X<0)OR(X>50)GOTO220
```

where N is the text cursor; X is set to the normalized character value; and a nonvalid character is outside the range of capitals A through Z.

To double label capacity, one must double the label-table. But, what about the first six scratchpad bytes 1F38H to 1F3DH used elsewhere by NIBASM? Six special ASCII characters separate the two alphabetic cases. If label-table pointer I is moved lower in memory by twice the number of lower-case and special ASCII characters combined, a version of the original label text/storage method can be devised that will skip over those original first-six scratchpad bytes. New program line 20 reflects the altered label-table pointer, I:

```
20PR"SC/MP ASM":PR"OBJCT CODE RAM=";:INPUTD:H=8100:I=7934:W=1
```

The modified second statement in line 200 still tests for character nonvalidity:

```
200X=(@N-65)*2:IF(X<0)OR(X>50)AND(X<64)OR(X>114)GOTO220
```

Lower-case alpha characters are no longer invalid, however, while the six special ASCII characters remain invalid.

Now when unmodified program line 210 indexes from pointer I by normalized valid-character X to store the label value, lower-case labels are stored in the original upper-case table. Upper-case labels now occupy the new scratchpad range 1EFEH though 1F31H, automatically skipping the originally allocated first six bytes (six free scratchpad bytes from 1F32H to 1F37H are also created.)

Label-table initialization adjustment is all that remains. An added loop-statement to program line 30 accesses the original table area by indexing from the new pointer-I address:

```
30FORX=0TO51:@(I+X)=#FF:NEXTX:
                        FORX=64TO115:@(I+X)=#FF:NEXTX:U=8050:Z=-1
```

The old loop is retained for initialization of the new table.

Using Expanded NIBASM

When writing source for expanded NIBASM, you may include the entire lower-case alphabet as beginning characters in 26 extra labels, while otherwise following Roger Marin's original rules.

Following the final source entry on a page, enter an immediate statement of the form:

```
■@TOP=#HH
```

where hh = FF when source to be assembled with the current page is contained on the next higher page or where hh = any lower value when assembly is not to be continued across a page boundary.

There are no other procedural differences between original and expanded NIBASM.

Incorporating the Modifications

Assuming packed entry, the source area expansion changes add 42 bytes to the original program. The label capacity expansion changes add 48 bytes. Either modification will fit if entered alone without further packing of Roger Marin's NIBASM source. NIBL may flag AREA ERROR during combined inclusion however.

Packing is one optimizing technique for interpreted programs. A line is read and executed more quickly when it contains no unnecessary characters or spaces. Packed text also allows more program within a given memory space.

Original NIBASM is well-packed as published. Still, I reduced its total size by 71 bytes as copied into my system. All eliminated bytes were spaces. Probably a reduction by half that amount would comfortably accommodate both modifications.

Addendum

I have just executed a test assembly with Expanded NIBASM on source text containing all SC/MP mnemonic/pointer permutations. This test assembly uncovered the only "bug" I've been able to find in the original program. The last statement on line 4525 tests the assembled opcode value in order to flag two-byte instructions for further processing. Unfortunately, the value tested for "greater-than" is the DELAY intruction opcode (mnemonic DLY), so the instruction is assembled incorrectly. The "bug" can be fixed by changing the test reference value from #8F to #8E.

Acknowledgment

Naval Medical Research and Development Command, Work Unit No. MF58.524.015.0018. The opinions and assertions contained herein are the private ones of the writer and are not to be construed as official or reflecting the view of the Navy Department or the Naval Service at large.

Footnote

[1] Both NIBL BASIC and NIBASM are available from NSC's user group: Compute/5226, National Semiconductor Corp., 2900 Semiconductor Dr., Santa Clara, CA 95051. SL0042A, NIBL-N—paper tape, SL0060, NIBASM—Listing.

Mr. Long accumulated 10 years of hardware experience in design, development modification, fabrication, and repair of analog and digital medical research instrumentation for the Naval Medical Research Institute before including microprocessors and software 2-½ years ago.

Opcode Design: 9 or 16 bits?

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

In an elementary essay on microprocessor instruction sets (*Kilobaud Microcomputing*, March and April, 1980), I tried to "read" the complex minds of the elite few who design them. Although I have met in person (too briefly for significant interaction) only two of them, I was not surprised to learn that they comprehend instantaneously not only a question but its *implications* (in my experience, a rare thing). But I know them by their work, unquestionably great art, as I know Aeschylus, or the architects of the great cathedrals of Bourges and Chartres, or certain poets who have used the more elusive constructs of English or German. As Juliet says, what's in a name? What matters is the profound structural diversity of human minds – and the possibility of symbolic intercommunication not restricted by time or space.

This prolegomenon has served only as a "signature," and I shall now lapse into rationality. The rational purpose of this note is to explore in depth a problem lightly touched on in my *Kilobaud* essay: how can one create, for an 8-bit CPU, more than the 256 possible one-byte opcodes? It is well-known that the Z-80 was the first design to use a "leading byte" (one of four different bit-patterns unused by the 8080) to endow the *next* byte with an utterly different meaning (from what it would have had standing-alone). The effect is to create 1024 *possible* two-byte codes, most of them not *defined* at the higher level of Z-80 assembler language. It is not surprising that the CPU circuitry nonetheless "runs" many of the undefined codes (cf. the letter by D. R. Lunsford, *DDJ* #44). The 6502 also runs most of its undefined (one-byte) opcodes, some as "nopcodes" (cf. my note in *DDJ* # 28, p. 29). I assume that it would complicate the microcircuitry if designers tried to "block" this (as *big* CPUs now do). Designers probably see undefined codes as a "reserve" that *could* be used for *new* instructions. In fact, the brief preview of the 6516 by David Kemp (*BYTE*, May 1979, p. 213) states that this upgrade will define the 105 8-bit opcodes not defined in the 6502 set. It also implies that the 6516 will go *no further*. That's a tenable position, but I wonder. The turmoil in the 6502 upgrade is indeed great (so much so that Chuck Peddle quit for a while!) but there are not enough bright people involved so that the number 256 *cannot* be an uncrossable limit. In reality, the limit must be *already* crossed, since the 6516 is not likely to discard the "decimal mode" status bit of the 6502. This bit, when set, *redefines* the 6502 arithmetic opcodes from binary to decimal operation.

We have now struck the *alternative* to the "leading byte" concept: the "ninth-bit" expansion (that the Signetics 2650, with its *two* status registers, uses very freely). This kind of expansion has its peculiar trade-offs. For example, if a programmer needs only *one* decimal-arithmetic instruction, this *must* be preceded by an SED (becoming in effect a *two-byte* opcode) and *should* be followed by a CLD to cancel the effect (effectively *three* opcode bytes!). However, the (one-byte) arithmetic opcode is normally inside the loop, and the two bit-control codes are outside. If the loop iterates the arithmetic code 20 times, the overall effect is to create a "1.1 byte" opcode with faster execution. Were it done by a "leading-byte," the program would have to load *forty* bytes during execution instead of only 22. On the other hand, since a "leading byte" is a one-time control, there is no need for an extra program byte to cancel its effect, and there may be less chance of error by careless programmers. The 6502 decimal bit does not, of course, eliminate a *leading* opcode byte, but the *following* "decimal-adjust" byte used to decimalize arithmetic instructions in most designs. However you look at it, *no* design creates a set of one-byte decimal arithmetic opcodes. *Most* use what is in effect a two-byte opcode to convert the binary arithmetic codes to decimal operation. This means they cannot have *dual-purpose* arithmetic programs, as the 6502 can, since the *same* program can be made to work either in binary or decimal depending on whether the decimal bit is set or reset before the call.

In my *Microcomputing* article (accepted by then-editor John Craig early in 1978) I express admiration for the 2650 and the belief that future designs will be influenced by its many innovations. In fact, the 6809 *did* adopt its flexible status-bit control, and its program-counter-relative addressing mode. However, this pioneer third-generation chip *rejected* the "ninth-bit" concept and imitated the Z-80 by creating multibyte opcodes. Since the 6809 is based on a lot of thought (cf. *BYTE*, Jan. 1979, p.14), this would seem to be a strong argument *against* the "ninth-bit" approach, that is not even *mentioned* in the *BYTE* presentation of the 6809. The brief discussion of "design decisions" for the 6809 does not say *who* made these decisions that to me sound like *restrictions* (I must emphasize that I am not acquainted with any of the restrictors or restrictees in this case, but I assume the universality of restriction.) One can speculate that Motorola *executives* imposed restrictions, and try to read *their* minds! It is my vague impression that these "practical" men fostered (tolerated?) the radical imaginings of Chuck Peddle that are the 6800, in the hope of overcoming the early lead of the 8080. They were justifiably annoyed when it did not, and perhaps surprised when Peddle's more advanced creation (the 6502) turned out to be an effective competitor. Since an obvious strength of the 6502 is its many addressing modes, and the success of the Z-80 could to some extent be attributed to enhanced addressing, a sure-fire response would be to endow the 6809 with vastly superior addressing power. This needs *lots* of opcodes! Since the Z-80 had succeeded with multibyte codes, while the 2650 had got nowhere with its "ninth-bits," practicality would demand "leading-

byte" solutions. The way of the pioneer is hard, and the 6809 shows signs of having left room for maneuvering beyond its "play-safe" philosophy, should any of its coming rivals include some spectacular innovation. It retains *both* one-byte *and* two-byte undefined opcodes. If you have lots of opcodes, and the *only* value of a "ninth-bit" is to economize on one-byte codes,. why bother with the complexities of status-bit redefinition?

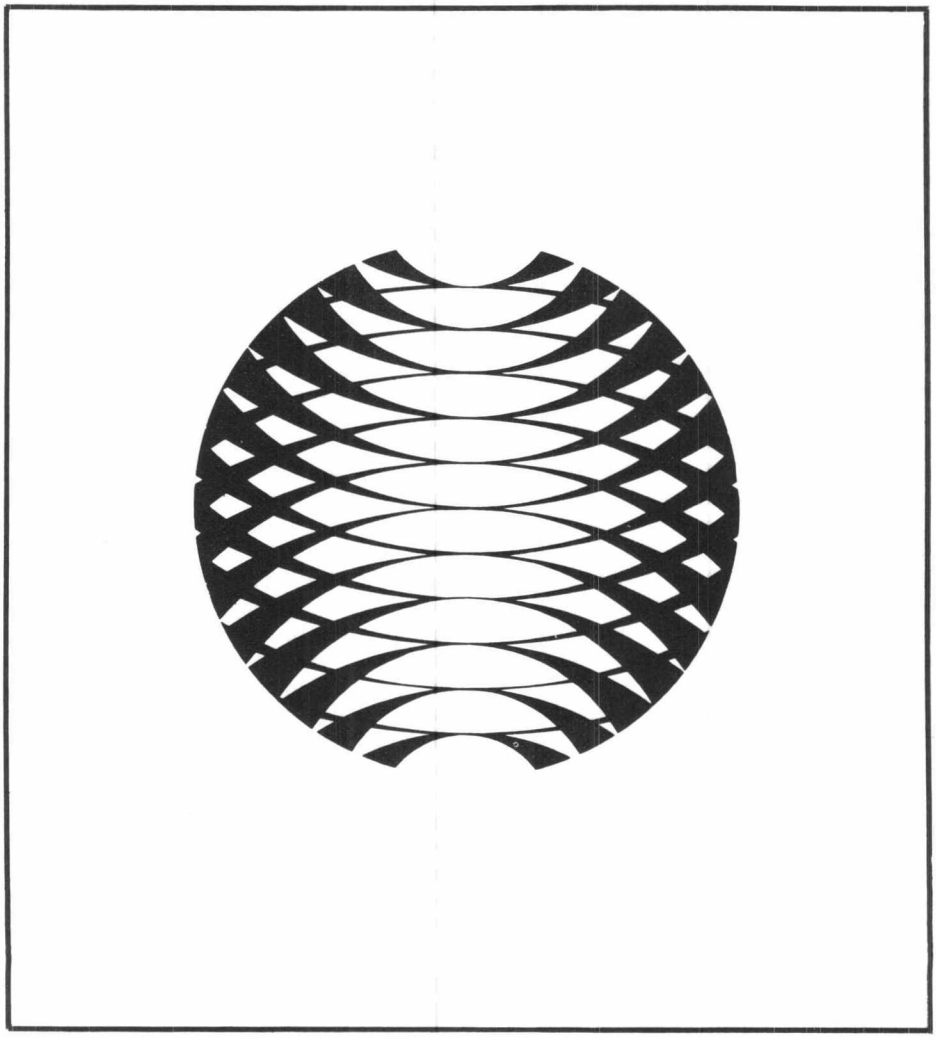
No design has yet used a status-bit to redefine *completely* the operation of an opcode bit-pattern (as a "leading-byte" does). However, the philosophic difference between the Z-80 and the 2650 extends to the implementation of their somewhat similar substitution of one set of on-chip registers for another. The Z-80 does this by a one-byte code that *exchanges* the information between its "on-line" set and its "off-line" one. The 2650 uses a status-bit to "shift" operation from one set to the other; this makes use of the existing two-byte codes for status-bit control, so it would seem that lower code-efficiency is being traded-off to save one new one-byte opcode. However, the possibility exists of *simultaneously* altering *other* control bits. It's hard to tell whether this would be useful often enough to compensate for the code-efficiency loss, or even how often the alternation of banks of registers will be used by programmers. Programmers are the *third* group of cooks in this cuisine, no less important than the designers and executives who can decide what the utensils shall be but not how they'll be used.

The register-interchange example shows that a *persistent state* can be established equally well by an opcode byte as by a status bit. However it's done, the programmer has to keep track of things. The 6809 has opcodes that cause exchange between any pair of its registers. As with the Z-80, the plethora of possible codes allowed lots of new operations!

The non-persistence of "leading-byte" modification may seem to be an advantage inimitable by status-bit control, but this is not so. A status-bit could be made volatile or non-volatile by using a *second* status-bit to control it. Consider the 6502 decimal-mode bit, and a hypothetical new control-bit. A "set-both-bits" code would be the equivalent of the present, persistent SED, reversed only by a "clear-both-bits" code. One-time operation could be had by a "set-decimal-clear-control" code; that would cause an *automatic* clear of the decimal bit by execution of any instruction involving that bit, making it equivalent to a "leading-byte" (except that it would not have to *precede immediately* the instruction code). Though not useful in the arithmetic context, the *principle* might work out well for status-bit shifting of operations like 16-bit vs. 8-bit mode (for which the 6516 seems to have created brand-new opcodes). Of course, the same thing could be done using two *kinds* of "leading-byte," one non-persistent (as now practiced) and the other persistent until reversed. There is some value in having a record of the status in a register, for debugging use, but the price entails having a *second* status register. I rather like this, since any otherwise unused bits could be used (2650-style) for some kind of imaginative application.

Although the two modes of opcode redefinition *could* be made equivalent, the status-bit mode has in fact only been used to redefine a *subset* of the one-byte opcodes that is expected to be used repeatedly and exclusively in a subprogram. Persistence is *desired*, and the gain is more in faster execution than in economy of one-byte opcodes. The "leading-byte" mode could not be optimized in the Z-80 (because it retained machine-code compatibility with the 8080), but it is carefully crafted in the 6809, that wisely sacrificed 6800 machine-code to gain efficiency. The "leading-byte" in effect "calls in" a *new* instruction set for a "one-time" use. Since this is less code- and time-efficient than the single-opcode-byte set, the instructions in the two-byte-opcode set are those expected to have a *lower usage frequency*. This will be a self-fulfilling expectation, since programmers will *avoid* using them wherever possible! The 6809 designers, well-aware that programmers make the ultimate decisions, studied program usage of the 6800 set. This kind of systems analysis is no doubt better than the inner-meditation that must have played a major role in the 6800 design. Yet, if the inner-mediator is a creative genius he may *impose* his will on programmers. I view systems analysts as the contemporary incarnations of the minor seers of the past, and likewise fated to be auxiliaries of the modern equivalent of kings. But destiny sporadically spawns a major prophet who can break into a truly new world. What dinosaur, however skillfully optimized, can withstand *that?*

The third generation of 8-bit micros will have to be more prodigious than the earlier ones. The 8080 had no competition. Now it's rife, and success will require a quantum leap in performance, not minor enhancement. I cannot even hazard a guess as to how close we are to the performance limits of this level of complexity!



MFT— a Multiple File Transfer Utility

BY LAWRENCE E. HUGHES
Mycroft Labs, P.O. Box 6045
Tallahassee, FL 32301

This article describes the design and use of a utility which makes it practical to use a single disk drive on the Digital Research CP/M operating system. The entire source is included in this article, and is also available from the author on an 8" single density, 128-byte sector diskette, along with a few other public domain programs, from the above address, for $15. I hereby release MFT into the public domain, for use on any CP/M (or other) system. No right is granted for resale, with or without modifications.

This program is the fifth generation of a mechanism which can transfer files from one diskette to another on a single drive CP/M system. CP/M was originally designed for use on a two-drive system, and a reasonable mechanism was provided for transferring files from one diskette to another (PEP). Unfortunately, no provisions were made for owners of one-drive systems. A temporary kludge is possible, which involves creating a "Pseudo drive" B. With this DIOS extension, anytime a new drive is accessed (via SELDSK), the user is prompted to mount the appropriate diskette in the one drive, and hit CR when it is mounted. This is simple enough to implement (if space is available in the DIOS), and it allows use of PIP to transfer files, but is somewhat clumsy to use, especially with PIP, which switches drives several times per file.
times per file.

Since it was fairly expensive for me to upgrade my single drive Imsai FDC2-1 to a dual drive FDC2-1, I made several attempts to solve the file transfer problem, with increasing degrees of success. Since I now have a dual drive Tarbell system, and will probably not be making any further improvements to MFT, I have decided to release it into the public domain. There are probably a number of good folk out there who are currently using a single drive system, or else are holding off on disk altogether until they can afford two drives—this program is for you! Even owners of dual drive systems may wish to study this code as an example of a fairly elaborate utility which interfaces to CP/M.

The following discussion is intended to serve as a guideline to reading the code, and was never meant to be particularly useful without the accompanying code.

Note the use of meaningful symbols for the system interface addresses, BDOS function codes, and FCB field offsets. Note further that all I/O is done via BDOS calls.

The first step is to clear the 'break flags' (IBFLG and OBFLG). These will be explained later. The next step is to determine how much RAM is available for use as a memory buffer. This is calculated by finding the starting address of the BDOS (there is a pointer to it at 0006H), then subtracting from that the starting address of the memory buffer (which starts at the end of MFT). Note that the buffer size is then rounded down to the nearest multiple of 128 bytes. The user is then notified of how large the memory buffer is (in terms of 128 byte sectors).

At this point, the user is asked to mount the input disk and type CR. This allows the user to transfer files from a disk which does not contain MFT. Once a CR is typed, if IBFLG is set, MFT jumps to MFT2A. IBFLG will be set only if we left off reading a file in the middle at the end of the last pass. On the first pass, the routine CFNT will be called to create the File Name Table (FNT) from the list of ambiguous file names on the command line. The pointers IFNTP (pointer to the next file to input) and OFNTP (pointer to the next file to output) are reset to the start of the FNT.

At MFT2A, the memory buffer pointer (MBUFP) is reset to the start of the memory buffer (MBUF). MSIZE is set to the total amount of available memory in MBUF. If IBFLG is set, the rest of the partially read file from last pass is set up for reading where we left off. Otherwise, the next available file from the FNT is set up for reading, and the file size (FSIZE) is set to 0.

At MFT6, the current file is read into memory starting at MBUFP. As each sector is read into memory, FSIZE is incremented, MSIZE is decremented, and MBUFP is advanced 128 bytes. If MSIZE reaches 0, MBUF is full, hence IBFLG is set, and reading ceases. If either MBUF fills, or the end of the current file is read, MFT8 is reached.

At MFT8, the current file is closed, and the user is told how many sectors were read into MBUF. Then the file size (FSIZE) is stored in the FNT (along

with the address at which this file started in MBUF). IFNTP is updated to point to the next file name to read. If IBFLG is not set, there must still be room in MBUF, so we loop back to MFT3 to read another file.

At MFT9, we either have run out of files, or MBUF is full, so we ask the user to mount the output disk and type CR. When the CR is typed, we make the new disk Read/Write (via subroutine RESET). If OBFLG is set, we are appending to the end of a partially written file, hence we reopen it where we left off. If OBFLG is not set, we make a new file with the name from the FNT which is pointed to by OFNTP. Note that we force the $R/W and $DIR attribute flags low for 2.0 users (can't write to an R/O file). We fetch the file size (into FSIZE) from the FNT (at OFNTP). As each sector is written, FSIZE is decremented an MBUFP is advanced 128 bytes, until FSIZE is 0. At this point, the file is closed. We then tell the user how many sectors were written, and loop to MFTA. If the last file was only partially written (IBFLG was set), then we set OBFLG. When we finish the last file that was read into MBUF, we jump to MFTF. At this point, if IBFLG is set, we loop to MFT1 and repeat this cycle until all files in the FNT have been transferred.

The subroutines are not described in this article, since what they do is fairly obvious. Pay particular attention to GFN and CFNT: all the others are trivial.

Note that a fair amount of error handling is included at various points in MFT to catch things like 'No such files', 'disk full', etc. Note also that as files are transferred, the user is informed as to what is going on. This is especially important in a program such as this, where the machine could be busy for many minutes.

As far as style goes, note that the main program comes first, followed by all subroutines, then the messages, and finally by the RAM storage. The use of semicolons to denote comments is consistent throughout, and makes it easy to determine where the different parts of the program lie.

For further information on the conventions of interfacing to CP/M, consult the excellent Digital Research manual set.

To use MFT, merely type its name, followed by one or more ambiguous file names (using ? and/or * wildcards). You will be prompted to mount the input disk and type CR. At this time the disk with MFT may be removed, if the desired files are on a different disk. Once the input disk is mounted, the directory will be searched and a list of all file names which satisfy at least one of the ambiguous file names will be compiled. As many files as will fit into memory will then be read. You will then be asked to mount the output disk and type CR. Note that MFT can write on this disk without your having to 'log it in'. All data read from the input disk is then written to the output disk. If more files remain to be read, this cycle repeats until all files are transferred. As an example, MFT *.* may be used to copy all files from one disk to another.

Lawrence Hughes is a Computer Research Specialist at Florida State University by day, and runs Mycroft Labs by night. He works with micro software in both pursuits. He is the organizer and current director of the Tallahassee Amateur Computer Society, and past vice president of the Atlanta Computer Society. He owns an Imsai with 64K, dual 8" floppies, Heath H-19, Decwriter, Hayes modem and Hiplot plotter. His computer's name is Seymour and has the distinction of being the first home computer to be listed in the Atlanta phone directory (under Computer, Seymour T.).

listing

```
0005 =          ENTRY   EQU     0005H           ;BDOS ENTRY POINT
005C =          TFCB    EQU     005CH           ;SYSTEM FILE CONTROL BLOCK
0080 =          DBUF    EQU     0080H           ;SYSTEM DISK BUFFER

000D =          CR      EQU     0DH
000A =          LF      EQU     0AH

0001 =          RCFC    EQU     01              ;READ CONSOLE
0002 =          WCFC    EQU     02              ;WRITE CONSOLE
000D =          RDFC    EQU     13              ;RESET DISK
000F =          OFFC    EQU     15              ;OPEN FILE
0010 =          CFFC    EQU     16              ;CLOSE FILE
0011 =          SFFC    EQU     17              ;SEARCH FIRST
0012 =          SNFC    EQU     18              ;SEARCH NEXT
0013 =          DFFC    EQU     19              ;DELETE FILE
0014 =          RRFC    EQU     20              ;READ RECORD
0015 =          WRFC    EQU     21              ;WRITE RECORD
0016 =          MFFC    EQU     22              ;MAKE FILE
001A =          SAFC    EQU     26              ;SET ADDRESS

0001 =          FN      EQU     01              ;FILE NAME OFFSET
0009 =          FT      EQU     09              ;FILE TYPE OFFSET
000C =          EX      EQU     12              ;EXTENT NUMBER OFFSET
0020 =          NR      EQU     32              ;NEXT RECORD OFFSET

0100                    ORG     100H
0100 31D10A     MFT:    LXI     SP,STACK+64
0103 219705             LXI     H,MSG1          ;PRINT 'MFT V3.1'
0106 CD8E04             CALL    WASC
0109 AF                 XRA     A               ;CLEAR BREAK FLAGS
010A 32940B             STA     IBFLG
010D 32950B             STA     OBFLG

                ;       CALCULATE BUFFER SIZE = FWA(BDOS) - FWA(MBUF)

0110 2A0600             LHLD    ENTRY+1         ;HL = (ENTRY+1) - 6
0113 11FAFF             LXI     D,-6
0116 19                 DAD     D
0117 7D                 MOV     A,L
0118 E680               ANI     80H             ;HL = HL MOD 128
011A 6F                 MOV     L,A
011B 11960B             LXI     D,MBUF          ;HL = HL - FWA(MBUF)
011E 7D                 MOV     A,L
011F 93                 SUB     E
0120 6F                 MOV     L,A
0121 7C                 MOV     A,H
0122 9A                 SBB     D
0123 67                 MOV     H,A

0124 29                 DAD     H               ;HL = HL / 128
0125 6C                 MOV     L,H
0126 3E00               MVI     A,0
0128 CE00               ACI     0
012A 67                 MOV     H,A
012B 2B                 DCX     H               ;SUBTRACT ONE
012C 22D10A             SHLD    SPACE           ;SAVE AS BUFFER SIZE
012F 21A205             LXI     H,MSG2          ;WRITE 'BUFFER SIZE = '
0132 CD8E04             CALL    WASC
0135 2AD10A             LHLD    SPACE           ;WRITE SIZE OF BUFFER
0138 CDA204             CALL    WDWC
013B 21B105             LXI     H,MSG3          ;WRITE ' SECTORS'
013E CD8E04             CALL    WASC

                ;       ASK USER TO MOUNT INPUT DISK

0141 21BC05     MFT1:   LXI     H,MSG4  ;WRITE 'MOUNT INPUT DISK, TYPE CR'
0144 CD8E04             CALL    WASC
0147 CD0105             CALL    RACC            ;READ RESPONSE
014A FE0D               CPI     CR              ;LOOP IF ANYTHING BUT CR
014C C24101             JNZ     MFT1
```

```
014F CD9804             CALL    WEOLC           ;ECHO CR,LF
0152 3A940B             LDA     IBFLG           ;JUMP IF IBFLG SET
0155 B7                 ORA     A
0156 C27D01             JNZ     MFT2A

                ;       COPY COMMAND LINE INTO CBUF

0159 218000             LXI     H,DBUF          ;FWA OF COMMAND LINE IMAGE
015C 11D50A             LXI     D,CBUF          ;FWA OF COMMAND BUFFER
015F 46                 MOV     B,M             ;FETCH COMMAND LINE IMAGE LENGTH
0160 23                 INX     H
0161 7E         MFT2:   MOV     A,M             ;FETCH NEXT BYTE FROM CLI
0162 23                 INX     H
0163 12                 STAX    D               ;STORE IN DBUF
0164 13                 INX     D
0165 05                 DCR     B               ;DECREMENT COUNT
0166 C26101             JNZ     MFT2            ;LOOP UNTIL ZERO
0169 AF                 XRA     A               ;STORE ZERO BYTE AT END
016A 12                 STAX    D
016B 21D50A             LXI     H,CBUF          ;RESET CBUFP
016E 22250B             SHLD    CBUFP
0171 CD1E04             CALL    CFNT            ;CREATE FILE NAME TABLE
0174 219006             LXI     H,FNT           ;RESET FNT POINTERS
0177 222B0B             SHLD    IFNTP
017A 222D0B             SHLD    OFNTP
017D 21960B     MFT2A:  LXI     H,MBUF          ;RESET MBUF POINTER
0180 222F0B             SHLD    MBUFP
0183 2AD10A             LHLD    SPACE           ;RESET MSIZE
0186 22D30A             SHLD    MSIZE
0189 3A940B             LDA     IBFLG           ;JUMP IF IBFLG NOT SET
018C B7                 ORA     A
018D CAC101             JZ      MFT3
0190 21310B             LXI     H,IFCB          ;COPY IFCB INTO TFCB
0193 115C00             LXI     D,TFCB
0196 0621               MVI     B,33
0198 CD9803             CALL    MOVE
019B AF                 XRA     A               ;CLEAR IBFLG
019C 32940B             STA     IBFLG
019F 3E01               MVI     A,1             ;SET OBFLG
01A1 32950B             STA     OBFLG
01A4 2A2B0B             LHLD    IFNTP           ;BACK IFNTP UP 4 BYTES
01A7 11FCFF             LXI     D,-4
01AA 19                 DAD     D
01AB 222B0B             SHLD    IFNTP
01AE 215D00             LXI     H,TFCB+FN       ;WRITE FILENAME
01B1 CD8E04             CALL    WASC
01B4 21F105             LXI     H,MSG6          ;WRITE ' - '
01B7 CD8E04             CALL    WASC
01BA 2A2B0B             LHLD    IFNTP           ;DE = IFNTP
01BD EB                 XCHG
01BE C3F501             JMP     MFT4            ;CONTINUE READING PREVIOUS FILE

                ;       PARSE OFF NEXT NAME FROM CBUF

01C1 2A2B0B     MFT3:   LHLD    IFNTP           ;FETCH INPUT FNT POINTER
01C4 7E                 MOV     A,M             ;JUMP IF END OF TABLE
01C5 FEFF               CPI     0FFH
01C7 CA7B02             JZ      MFT9
01CA 3601               MVI     M,1             ;SET 'FILE READ' FLAG
01CC 23                 INX     H
01CD 115D00             LXI     D,TFCB+FN       ;COPY FILENAME INTO TFCB
01D0 060B               MVI     B,11
01D2 CD9803             CALL    MOVE
01D5 222B0B             SHLD    IFNTP           ;SAVE INPUT FNT POINTER
01D8 215D00             LXI     H,TFCB+FN       ;WRITE FILENAME
01DB CD8E04             CALL    WASC
01DE 21F105             LXI     H,MSG6          ;WRITE ' - '
01E1 CD8E04             CALL    WASC
01E4 AF                 XRA     A               ;SETUP TFCB
01E5 325C00             STA     TFCB
01E8 326800             STA     TFCB+EX
01EB 327C00             STA     TFCB+NR
01EE CD1905             CALL    OPEN            ;OPEN FILE
01F1 2A2B0B             LHLD    IFNTP
01F4 EB                 XCHG
01F5 2A2F0B     MFT4:   LHLD    MBUFP
01F8 7C                 MOV     A,H             ;COPY INTO FNT ENTRY
01F9 12                 STAX    D
01FA 13                 INX     D
01FB 7D                 MOV     A,L
01FC 12                 STAX    D
01FD 13                 INX     D
01FE EB                 XCHG                    ;SAVE FNT POINTER
01FF 222B0B             SHLD    IFNTP
0202 210000             LXI     H,0             ;FILE SIZE (IN SECTORS) = 0
0205 22270B             SHLD    FSIZE

                ;       READ NEXT FILE FROM INPUT DISK

0208 2A2F0B     MFT6:   LHLD    MBUFP
020B EB                 XCHG
020C CD7305             CALL    SETDMA
020F CD2805             CALL    READ            ;READ NEXT SECTOR
0212 B7                 ORA     A               ;JUMP IF NORMAL TRANSFER
0213 CA2402             JZ      MFT7
0216 FE01               CPI     1               ;JUMP IF EOF
0218 CA5102             JZ      MFT8
021B 210206             LXI     H,MSG8          ;WRITE 'READ ERROR - '
021E CD8E04             CALL    WASC
0221 C35102             JMP     MFT8            ;CONTINUE AS IF EOF
0224 2A2F0B     MFT7:   LHLD    MBUFP           ;MBUFP = MBUFP + 128
0227 118000             LXI     D,128
022A 19                 DAD     D
022B 222F0B             SHLD    MBUFP
022E 2A270B             LHLD    FSIZE           ;FSIZE = FSIZE + 1
0231 23                 INX     H
0232 22270B             SHLD    FSIZE
0235 2AD30A             LHLD    MSIZE           ;DECREMENT MSIZE
0238 2B                 DCX     H
0239 22D30A             SHLD    MSIZE
023C 7C                 MOV     A,H             ;LOOP IF STILL POSITIVE
023D B5                 ORA     L
023E C20802             JNZ     MFT6
0241 215C00             LXI     H,TFCB          ;COPY TFCB INTO IFCB
0244 11310B             LXI     D,IFCB
0247 0621               MVI     B,33
0249 CD9803             CALL    MOVE
024C 3E01               MVI     A,1             ;SET IBFLG
024E 32940B             STA     IBFLG
0251 118000     MFT8:   LXI     D,DBUF          ;RESET DMA POINTER
0254 CD7305             CALL    SETDMA
0257 CD3705             CALL    CLOSE           ;CLOSE FILE
025A 2A270B             LHLD    FSIZE           ;WRITE FILE SIZE
025D CDA204             CALL    WDWC
0260 211006             LXI     H,MSG9          ;WRITE ' SECTORS READ'
0263 CD8E04             CALL    WASC

                ;       UPDATE FNT, LOOP

0266 2A270B             LHLD    FSIZE           ;DE = FILE SIZE
0269 EB                 XCHG
026A 2A2B0B             LHLD    IFNTP           ;STORE FILE SIZE IN FNT ENTRY
026D 72                 MOV     M,D
026E 23                 INX     H
026F 73                 MOV     M,E
0270 23                 INX     H
0271 222B0B             SHLD    IFNTP           ;SAVE FNT POINTER
0274 3A940B             LDA     IBFLG           ;LOOP IF IBFLG NOT SET
0277 B7                 ORA     A
0278 CAC101             JZ      MFT3

                ;       ASK USER TO MOUNT OUTPUT DISK

027B 212006     MFT9:   LXI     H,MSGA          ;WRITE 'MOUNT OUTPUT DISK, TYPE CR
027E CD8E04             CALL    WASC
0281 CD0105             CALL    RACC            ;READ RESPONSE
0284 FE0D               CPI     CR              ;LOOP IF ANYTHING BUT CR
0286 C27B02             JNZ     MFT9
0289 CD9804             CALL    WEOLC           ;ECHO CR,LF
028C CD0D05             CALL    RESET           ;RESET DISK SYSTEM (MAKE R/W)
028F 3A950B             LDA     OBFLG           ;JUMP IF OBFLG NOT SET
0292 B7                 ORA     A
0293 CABD02             JZ      MFTA
0296 21520B             LXI     H,OFCB          ;COPY OFCB INTO TFCB
0299 115C00             LXI     D,TFCB
029C 0621               MVI     B,33
029E CD9803             CALL    MOVE
02A1 CD1905             CALL    OPEN            ;OPEN PREVIOUS FILE
02A4 2A2D0B             LHLD    OFNTP           ;BACKUP OUTPUT FNT POINTER 4 BYTES
```

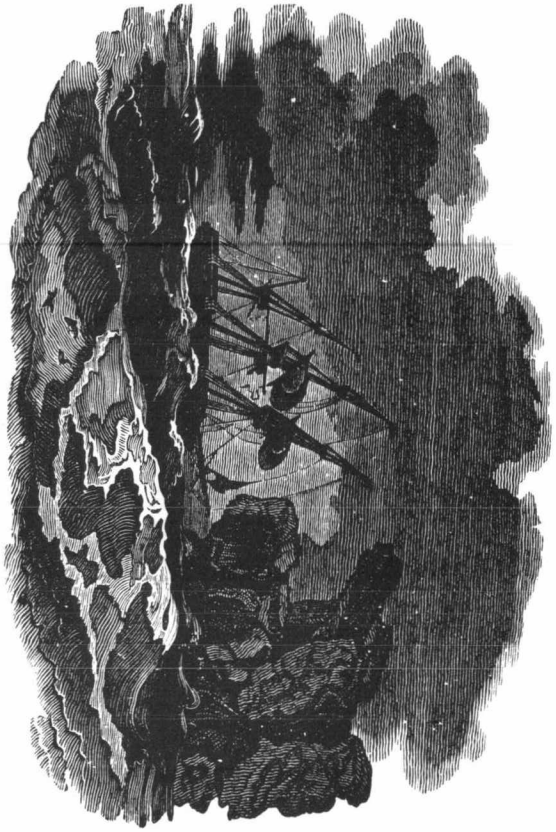
(Continued on page 28.)

(Continued from page 26.)

```
02A7 11FCFF            LXI     D,-4
02AA 19                DAD     D
02AB 222D0B            SHLD    OFNTP
02AE 215D00            LXI     H,TFCB+FN       ;WRITE FILE NAME
02B1 CD8E04            CALL    WASC
02B4 21F105            LXI     H,MSG6          ;WRITE ' - '
02B7 CD8E04            CALL    WASC
02BA C31003            JMP     MFTB            ;CONTINUE WRITING PREVIOUS FILE

                ;      WRITE NEXT FILE TO OUTPUT DISK

02BD 2A2D0B     MFTA:  LHLD    OFNTP
02C0 7E                MOV     A,M
02C1 B7                ORA     A
02C2 CA8E03            JZ      MFTF
02C5 FEFF              CPI     0FFH
02C7 CA8E03            JZ      MFTF
02CA 23                INX     H
02CB 115D00            LXI     D,TFCB+FN
02CE 060B              MVI     B,11
02D0 CD9803            CALL    MOVE
02D3 222D0B            SHLD    OFNTP
02D6 215D00            LXI     H,TFCB+FN
02D9 CD8E04            CALL    WASC
02DC 21F105            LXI     H,MSG6
02DF CD8E04            CALL    WASC
02E2 AF                XRA     A
02E3 325C00            STA     TFCB
02E6 326800            STA     TFCB+EX
02E9 327C00            STA     TFCB+NR
02EC 3A6500            LDA     TFCB+9          ;FORCE TO $R/W FOR 2.0
02EF E67F              ANI     7FH
02F1 326500            STA     TFCB+9
02F4 3A6600            LDA     TFCB+10         ;FORCE TO $DIR FOR 2.0
02F7 E67F              ANI     7FH
02F9 326600            STA     TFCB+10
02FC CD4605            CALL    DELT            ;TRY TO CREATE OUTPUT FILE
02FF CD5505            CALL    MAKE
0302 FEFF              CPI     255             ;JUMP IF OK
0304 C21003            JNZ     MFTB
0307 213B06            LXI     H,MSGB          ;WRITE 'UNABLE TO CREATE'
030A CD8E04            CALL    WASC
030D C39503            JMP     MFTG
0310 2A2D0B     MFTB:  LHLD    OFNTP
0313 56                MOV     D,M             ;FETCH FWA OF FILE FROM FNT
0314 23                INX     H
0315 5E                MOV     E,M
0316 23                INX     H
0317 EB                XCHG
0318 222F0B            SHLD    MBUFP           ;SAVE IT
031B EB                XCHG
031C 56                MOV     D,M             ;FETCH SIZE OF FILE FROM FNT
031D 23                INX     H
031E 5E                MOV     E,M
031F 23                INX     H
0320 EB                XCHG
0321 22270B            SHLD    FSIZE           ;SAVE IT
0324 22290B            SHLD    XSIZE           ;SAVE FOR PRINTOUT
0327 EB                XCHG
0328 222D0B            SHLD    OFNTP
032B 2A270B            LHLD    FSIZE           ;JUMP IF FSIZE=0
032E 7C                MOV     A,H
032F B5                ORA     L
0330 CA6003            JZ      MFTDA
0333 2A2F0B     MFTC:  LHLD    MBUFP           ;SET DMA ADDRESS TO MBUFP
0336 EB                XCHG
0337 CD7305            CALL    SETDMA
033A CD6405            CALL    WRITE           ;WRITE NEXT SECTOR
033D B7                ORA     A               ;JUMP IF OK
033E CA4A03            JZ      MFTD
0341 214E06            LXI     H,MSGC          ;WRITE 'ERROR WRITING FILE'
0344 CD8E04            CALL    WASC
0347 C39503            JMP     MFTG
034A 2A2F0B     MFTD:  LHLD    MBUFP           ;MBUFP = MBUFP + 128
034D 118000            LXI     D,128
0350 19                DAD     D
0351 222F0B            SHLD    MBUFP
0354 2A270B            LHLD    FSIZE           ;FSIZE = FSIZE - 1
0357 2B                DCX     H
0358 22270B            SHLD    FSIZE
035B 7C                MOV     A,H             ;LOOP UNTIL ZERO
035C B5                ORA     L
035D C23303            JNZ     MFTC
0360 215C00     MFTDA: LXI     H,TFCB          ;COPY TFCB INTO OFCB
0363 11520B            LXI     D,OFCB
0366 0621              MVI     B,33
0368 CD9803            CALL    MOVE
036B 118000            LXI     D,DBUF          ;RESET DMA POINTER
036E CD7305            CALL    SETDMA
0371 CD3705            CALL    CLOSE           ;TRY TO CLOSE FILE
0374 FEFF              CPI     255             ;JUMP IF OK
0376 C27F03            JNZ     MFTE
0379 216306            LXI     H,MSGD          ;WRITE 'UNABLE TO CLOSE'
037C CD8E04            CALL    WASC
037F 2A290B     MFTE:  LHLD    XSIZE           ;WRITE NUMBER OF SECTORS WRITTEN
0382 CDA204            CALL    WDWC
0385 217506            LXI     H,MSGE          ;WRITE ' SECTORS WRITTEN'
0388 CD8E04            CALL    WASC
038B C3BD02            JMP     MFTA
038E 3A940B     MFTF:  LDA     IBFLG           ;LOOP IF IBFLG SET
0391 B7                ORA     A
0392 C24101            JNZ     MFT1
0395 C30000     MFTG:  JMP     0               ;EXIT TO CP/M

                ;      SUBROUTINES

0398 7E         MOVE:  MOV     A,M
0399 23                INX     H
039A 12                STAX    D
039B 13                INX     D
039C 05                DCR     B
039D C29803            JNZ     MOVE
03A0 C9                RET

                ;;     GFN - GET FILE NAME
                ;
03A1 7E         GFN:   MOV     A,M
03A2 B7                ORA     A
03A3 C8                RZ
03A4 FE20              CPI     ' '
03A6 C2AD03            JNZ     GFN0
03A9 23                INX     H
03AA C3A103            JMP     GFN
03AD 11730B     GFN0:  LXI     D,XFCB
03B0 AF                XRA     A
03B1 12                STAX    D
03B2 13                INX     D
03B3 D5                PUSH    D
03B4 060B              MVI     B,11
03B6 3E20              MVI     A,' '
03B8 12         GFN6:  STAX    D
03B9 13                INX     D
03BA 05                DCR     B
03BB C2B803            JNZ     GFN6
03BE D1                POP     D
03BF 0609              MVI     B,9
03C1 7E         GFN1:  MOV     A,M
03C2 B7                ORA     A
03C3 CA1A04            JZ      GFN4
03C6 23                INX     H
03C7 FE20              CPI     ' '
03C9 CA1A04            JZ      GFN4
03CC FE2E              CPI     '.'
03CE CAF103            JZ      GFN2
03D1 FE2A              CPI     '*'
03D3 CADF03            JZ      GFN7
03D6 12                STAX    D
03D7 13                INX     D
03D8 05                DCR     B
03D9 CA1C04            JZ      GFN5
03DC C3C103            JMP     GFN1
03DF 05         GFN7:  DCR     B
03E0 CAEA03            JZ      GFN9
03E3 3E3F              MVI     A,'?'
03E5 12                STAX    D
```

```
03E6 13             INX     D
03E7 C3DF03         JMP     GFN7
03EA 7E       GFN9: MOV     A,M
03EB FE2E           CPI     '.'
03ED C21A04         JNZ     GFN4
03F0 23             INX     H
03F1 117C0B   GFN2: LXI     D,XFCB+FT
03F4 0604           MVI     B,4
03F6 7E       GFN3: MOV     A,M
03F7 B7             ORA     A
03F8 CA1A04         JZ      GFN4
03FB 23             INX     H
03FC FE20           CPI     ' '
03FE CA1A04         JZ      GFN4
0401 FE2A           CPI     '*'
0403 CA0F04         JZ      GFN8
0406 12             STAX    D
0407 13             INX     D
0408 05             DCR     B
0409 CA1C04         JZ      GFN5
040C C3F603         JMP     GFN3
040F 05       GFN8: DCR     B
0410 CA1A04         JZ      GFN4
0413 3E3F           MVI     A,'?'
0415 12             STAX    D
0416 13             INX     D
0417 C30F04         JMP     GFN8
041A AF       GFN4: XRA     A
041B C9             RET
041C 37       GFN5: STC
041D C9             RET

                ;;      CFNT - CREATE FILE NAME TABLE
                ;

041E 219006   CFNT:   LXI     H,FNT           ;RESET IFNTP
0421 222B0B           SHLD    IFNTP
0424 2A250B   CFNT1:  LHLD    CBUFP           ;GET CBUFP
0427 7E               MOV     A,M             ;EXIT IF END OF LIST
0428 B7               ORA     A
0429 C8               RZ
042A CDA103           CALL    GFN             ;GET NEXT AFN
042D 22250B           SHLD    CBUFP           ;SAVE COMMAND BUFFER PTR
0430 D23C04           JNC     CFNT2           ;JUMP IF FILENAME OK
0433 21D605           LXI     H,MSG5          ;WRITE 'SYNTAX ERROR IN FILENAME'
0436 CD8E04           CALL    WASC
0439 C32404           JMP     CFNT1           ;LOOP
043C AF       CFNT2:  XRA     A               ;CLEAR XFCB EXTENT FIELD
043D 327F0B           STA     XFCB+EX
0440 11730B           LXI     D,XFCB          ;SEARCH FOR FIRST OCCURANCE
0443 CD7F05           CALL    SRCHF
0446 FEFF             CPI     255             ;JUMP IF FOUND
0448 C25A04           JNZ     CFNT3
044B 21740B           LXI     H,XFCB+FN       ;WRITE FILENAME
044E CD8E04           CALL    WASC
0451 21F505           LXI     H,MSG7          ;WRITE ' NOT FOUND'
0454 CD8E04           CALL    WASC
0457 C32404           JMP     CFNT1           ;LOOP
045A E603     CFNT3:  ANI     3               ;INDEX INTO CBUF
045C 6F               MOV     L,A
045D 2600             MVI     H,0
045F 29               DAD     H
0460 29               DAD     H
0461 29               DAD     H
0462 29               DAD     H
0463 29               DAD     H
0464 118000           LXI     D,DBUF
0467 19               DAD     D
0468 EB               XCHG                    ;COPY FILENAME INTO FNT
0469 2A2B0B           LHLD    IFNTP
046C EB               XCHG
046D 060C             MVI     B,12
046F CD9803           CALL    MOVE
0472 218806           LXI     H,ZEROS         ;ZERO FILL REST OF ENTRY
0475 0604             MVI     B,4
0477 CD9803           CALL    MOVE
047A EB               XCHG
047B 222B0B           SHLD    IFNTP           ;SAVE INPUT FNT POINTER
047E 36FF             MVI     M,0FFH          ;INSURE FF BYTE AT END
0480 11730B           LXI     D,XFCB          ;SEARCH FOR NEXT OCCURANCE
0483 CD8B05           CALL    SRCHN
0486 FEFF             CPI     255             ;JUMP IF FOUND
0488 C25A04           JNZ     CFNT3
048B C32404           JMP     CFNT1           ;GO GET NEXT AFN

                ;;      WASC - WRITE ASCII STRING TO CONSOLE
                ;

048E 7E       WASC:   MOV     A,M
048F B7               ORA     A
0490 C8               RZ
0491 CDF204           CALL    WACC
0494 23               INX     H
0495 C38E04           JMP     WASC

                ;;      WEOLC - WRITE END OF LINE TO CONSOLE
                ;

0498 3E0D     WEOLC:  MVI     A,CR
049A CDF204           CALL    WACC
049D 3E0A             MVI     A,LF
049F C3F204           JMP     WACC

                ;;      WDWC - WRITE DECIMAL WORD TO CONSOLE
                ;

04A2 E5       WDWC:   PUSH    H
04A3 D5               PUSH    D
04A4 C5               PUSH    B
04A5 0600             MVI     B,0             ;CLEAR 'DIGIT WRITTEN' FLAG
04A7 111027           LXI     D,10000         ;WRITE 1ST DIGIT
04AA CDCB04           CALL    WNDD
04AD 11E803           LXI     D,1000          ;WRITE 2ND DIGIT
04B0 CDCB04           CALL    WNDD
04B3 116400           LXI     D,100           ;WRITE 3RD DIGIT
04B6 CDCB04           CALL    WNDD
04B9 110A00           LXI     D,10            ;WRITE 4TH DIGIT
04BC CDCB04           CALL    WNDD
04BF 110100           LXI     D,1             ;WRITE 5TH DIGIT
04C2 0601             MVI     B,1             ;FORCE LAST DIGIT TO PRINT
04C4 CDCB04           CALL    WNDD
04C7 C1               POP     B
04C8 D1               POP     D
04C9 E1               POP     H
04CA C9               RET

04CB 0E00     WNDD:   MVI     C,0             ;C=0
04CD 7D       WNDD1:  MOV     A,L             ;HL = HL - DE
04CE 93               SUB     E
04CF 6F               MOV     L,A
04D0 7C               MOV     A,H
04D1 9A               SBB     D
04D2 67               MOV     H,A
04D3 DADA04           JC      WNDD2           ;JUMP IF < 0
04D6 0C               INR     C               ;C = C + 1
04D7 C3CD04           JMP     WNDD1           ;LOOP
04DA 19       WNDD2:  DAD     D               ;HL = HL + DE
04DB 79               MOV     A,C             ;JUMP IF C NON-ZERO
04DC B1               ORA     C
04DD C2EA04           JNZ     WNDD4
04E0 78               MOV     A,B             ;JUMP IF DIGIT WRITTEN
04E1 B0               ORA     B
04E2 C2EA04           JNZ     WNDD4
04E5 3E20             MVI     A,' '           ;WRITE ONE SPACE
04E7 C3F204           JMP     WACC
04EA 0601     WNDD4:  MVI     B,1             ;SET 'DIGIT WRITTEN' FLAG
04EC 79               MOV     A,C             ;ENCODE C INTO DECIMAL ASCII
04ED C630             ADI     '0'
04EF C3F204           JMP     WACC            ;GO WRITE IT

                ;;      WACC - WRITE ASCII CHARACTER TO CONSOLE
                ;

04F2 E5       WACC:   PUSH    H
04F3 D5               PUSH    D
04F4 C5               PUSH    B
04F5 F5               PUSH    PSW
```

```
04F6 0E02               MVI     C,WCFC
04F8 5F                 MOV     E,A
04F9 CD0500             CALL    ENTRY
04FC F1                 POP     PSW
04FD C1                 POP     B
04FE D1                 POP     D
04FF E1                 POP     H
0500 C9                 RET

                ;;      RACC - READ ASCII CHARACTER FROM CONSOLE
0501 E5         RACC:   PUSH    H
0502 D5                 PUSH    D
0503 C5                 PUSH    B
0504 0E01               MVI     C,RCFC
0506 CD0500             CALL    ENTRY
0509 C1                 POP     B
050A D1                 POP     D
050B E1                 POP     H
050C C9                 RET

                ;;      RESET - RESET DISK SYSTEM
                ;
050D E5         RESET:  PUSH    H
050E D5                 PUSH    D
050F C5                 PUSH    B
0510 0E0D               MVI     C,RDFC
0512 CD0500             CALL    ENTRY
0515 C1                 POP     B
0516 D1                 POP     D
0517 E1                 POP     H
0518 C9                 RET

                ;;      OPEN - OPEN DISK FILE
                ;
0519 E5         OPEN:   PUSH    H
051A D5                 PUSH    D
051B C5                 PUSH    B
051C 115C00             LXI     D,TFCB
051F 0E0F               MVI     C,OFFC
0521 CD0500             CALL    ENTRY
0524 C1                 POP     B
0525 D1                 POP     D
0526 E1                 POP     H
0527 C9                 RET

                ;;      READ - READ RECORD FROM DISK FILE
                ;
0528 E5         READ:   PUSH    H
0529 D5                 PUSH    D
052A C5                 PUSH    B
052B 115C00             LXI     D,TFCB
052E 0E14               MVI     C,RRFC
0530 CD0500             CALL    ENTRY
0533 C1                 POP     B
0534 D1                 POP     D
0535 E1                 POP     H
0536 C9                 RET

                ;;      CLOSE - CLOSE DISK FILE
                ;
0537 E5         CLOSE:  PUSH    H
0538 D5                 PUSH    D
0539 C5                 PUSH    B
053A 115C00             LXI     D,TFCB
053D 0E10               MVI     C,CFFC
053F CD0500             CALL    ENTRY
0542 C1                 POP     B
0543 D1                 POP     D
0544 E1                 POP     H
0545 C9                 RET

                ;;      DELT - DELETE DISK FILE
                ;
0546 E5         DELT:   PUSH    H
0547 D5                 PUSH    D
0548 C5                 PUSH    B
0549 115C00             LXI     D,TFCB
054C 0E13               MVI     C,DFFC
054E CD0500             CALL    ENTRY
0551 C1                 POP     B
0552 D1                 POP     D
0553 E1                 POP     H
0554 C9                 RET

                ;;      MAKE - MAKE NEW DISK FILE
                ;
0555 E5         MAKE:   PUSH    H
0556 D5                 PUSH    D
0557 C5                 PUSH    B
0558 115C00             LXI     D,TFCB
055B 0E16               MVI     C,MFFC
055D CD0500             CALL    ENTRY
0560 C1                 POP     B
0561 D1                 POP     D
0562 E1                 POP     H
0563 C9                 RET

                ;;      WRITE - WRITE RECORD TO FILE
                ;
0564 E5         WRITE:  PUSH    H
0565 D5                 PUSH    D
0566 C5                 PUSH    B
0567 115C00             LXI     D,TFCB
056A 0E15               MVI     C,WRFC
056C CD0500             CALL    ENTRY
056F C1                 POP     B
0570 D1                 POP     D
0571 E1                 POP     H
0572 C9                 RET

                ;;      SETDMA - SET DMA ADDRESS
                ;
0573 E5         SETDMA: PUSH    H
0574 D5                 PUSH    D
0575 C5                 PUSH    B
0576 0E1A               MVI     C,SAFC
0578 CD0500             CALL    ENTRY
057B C1                 POP     B
057C D1                 POP     D
057D E1                 POP     H
057E C9                 RET

                ;;      SRCHF - SEARCH FOR FIRST OCCURANCE OF AFN
                ;
057F E5         SRCHF:  PUSH    H
0580 D5                 PUSH    D
0581 C5                 PUSH    B
0582 0E11               MVI     C,SFFC
0584 CD0500             CALL    ENTRY
0587 C1                 POP     B
0588 D1                 POP     D
0589 E1                 POP     H
058A C9                 RET

                ;;      SRCHN - SEARCH FOR NEXT OCCURANCE OF AFN
                ;
058B E5         SRCHN:  PUSH    H
058C D5                 PUSH    D
058D C5                 PUSH    B
058E 0E12               MVI     C,SNFC
0590 CD0500             CALL    ENTRY
0593 C1                 POP     B
0594 D1                 POP     D
0595 E1                 POP     H
0596 C9                 RET
```



```
0597 4D46542056MSG1:    DB      'MFT V3.1',CR,LF,0
05A2 4275666665MSG2:    DB      'Buffer size = ',0
05B1 2073656374MSG3:    DB      ' sectors',CR,LF,0
05BC 4D6F756E74MSG4:    DB      'Mount input disk, type CR',0
05D6 53796E7461MSG5:    DB      'Syntax error in filename',CR,LF,0
05F1 202D2000  MSG6:    DB      ' - ',0
05F5 206E6F7420MSG7:    DB      ' not found',CR,LF,0
0602 7265616420MSG8:    DB      'read error - ',0
0610 2073656374MSG9:    DB      ' sectors read',CR,LF,0
0620 4D6F756E74MSGA:    DB      'Mount output disk, type CR',0
063B 756E61626CMSGB:    DB      'unable to create',CR,LF,0
064E 6572726F72MSGC:    DB      'error writing file',CR,LF,0
0663 756E61626CMSGD:    DB      'unable to close',CR,LF,0
0675 2073656374MSGE:    DB      ' sectors written',CR,LF,0

0688 00000000  ZEROS:   DB      0,0,0,0

0690                    ORG     ($+15)/16*16

0690           FNT:     DS      16*64+1
```

0AD5 CBUF	0B25 CBUFP	0010 CFFC	041E CFNT	0424 CFNT1
043C CFNT2	045A CFNT3	0537 CLOSE	000D CR	0080 DBUF
0546 DELT	0013 DFFC	0005 ENTRY	000C EX	0001 FN
0690 FNT	0B27 FSIZE	0009 FT	03A1 GFN	03AD GFN0
03C1 GFN1	03F1 GFN2	03F6 GFN3	041A GFN4	041C GFN5
03B8 GFN6	03DF GFN7	040F GFN8	03EA GFN9	0B94 IBFLG
0B31 IFCB	0B2B IFNTP	000A LF	0555 MAKE	0B96 MBUF
0B2F MBUFP	0016 MFFC	0100 MFT	0141 MFT1	017D MFT2A
0161 MFT2	01C1 MFT3	01F5 MFT4	0208 MFT6	0224 MFT7
0251 MFT8	027B MFT9	02BD MFTA	0310 MFTB	0333 MFTC
0360 MFTDA	034A MFTD	037F MFTE	038E MFTF	0395 MFTG
0398 MOVE	0597 MSG1	05A2 MSG2	05B1 MSG3	05BC MSG4
05D6 MSG5	05F1 MSG6	05F5 MSG7	0602 MSG8	0610 MSG9
0620 MSGA	063B MSGB	064E MSGC	0663 MSGD	0675 MSGE
0AD3 MSIZE	0020 NR	0B95 OBFLG	0B52 OFCB	000F OFFC
0B2D OFNTP	0519 OPEN	0501 RACC	0001 RCFC	000D RDFC
0528 READ	050D RESET	0014 RRFC	001A SAFC	0573 SETDMA
0011 SFFC	0012 SNFC	0AD1 SPACE	057F SRCHF	058B SRCHN
0A91 STACK	005C TFCB	04F2 WACC	048E WASC	0002 WCFC
04A2 WDWC	0498 WEOLC	04CB WNDD	04CD WNDD1	04DA WNDD2
04EA WNDD4	0015 WRFC	0564 WRITE	0B73 XFCB	0B29 XSIZE
0688 ZEROS				

```
0A91           STACK:   DS      64
0AD1           SPACE:   DS      2               ;AVAILABLE SPACE
0AD3           MSIZE:   DS      2               ;MEMORY SIZE
0AD5           CBUF:    DS      80              ;COMMAND BUFFER
0B25           CBUFP:   DS      2               ;COMMAND BUFFER POINTER
0B27           FSIZE:   DS      2               ;FILE SIZE IN SECTORS
0B29           XSIZE:   DS      2               ;FILE SIZE FOR PRINTOUT
0B2B           IFNTP:   DS      2               ;INPUT FNT POINTER
0B2D           OFNTP:   DS      2               ;OUTPUT FNT POINTER
0B2F           MBUFP:   DS      2               ;MEMORY BUFFER POINTER
0B31           IFCB:    DS      33              ;INPUT FCB
0B52           OFCB:    DS      33              ;OUTPUT FCB
0B73           XFCB:    DS      33              ;TEMPORARY FCB
0B94           IBFLG:   DS      1               ;INPUT BREAK FLAG
0B95           OBFLG:   DS      1               ;OUTPUT BREAK FLAG

0B96 =         MBUF     EQU     $

0B96                    END     MFT
```

end

A Cross Reference Program for Northstar BASIC

BY LARRY HUDSON
2232 Alameda Ave.
Ventura, Calif. 93003

Here is a combined line number and variable cross referencing program for Northstar BASIC. Because this is a machine language program, it runs much faster than the others I have seen which were written in BASIC. This one will do a 500 line program in about 30 seconds.

I have tried to write it to be as general as possible. All I/O is through the normal Northstar DOS I/O, and naturally the disk routines are through the DOS. All other subroutines are within this program. The output routine uses the standard Northstar convention of specifying the output device by the code in the accumulator. As it is written now, it uses 0 for the console device and 1 for a hard-copy device. These numbers can easily be changed if needed.

It reads the program to be cross referenced directly from disk. It accepts the standard Northstar convention of following the program name with the drive number. It creates the cross references for both the line numbers and variables in the same pass, and follows this with a second pass to flag any bad line number references. After these lists are created, the program will ask which list you want printed, and when that is done it will ask if you want another one. In the line number cross reference list, any reference to a non-existent line is flagged with an asterisk.

When answering the questions in the program, it will only accept L or V for lines or variables, or Y or N for yes or no. They can be entered in either upper or lower case, and no carriage return is required. All other keys except Control-C are ignored. The control-C will cause an immediate abort back to the DOS. This is not the standard DOS control-C check; it will not interrupt the running program; it is effective only when it expects a keyboard input.

When entering the program name, either the underscore or delete will backstep to delete characters. Both will echo the underscore. This set of backstep characters can also easily be changed. The buffer to hold the program name is only 11 bytes long, but the input routine keeps counters to prevent overrunning either end of it.

This program loads in the area normally occupied by BASIC, with the disk buffer located at address 2A00 through 2AFF. If you reassemble it for a different location, be sure this buffer is located at a page boundary or the GETCHR routine won't work. There should be no other restrictions on locations. The cross reference lists are created as linked lists immediately following this program.

2B00 is the normal entry point of this program. 2B03 is a second entry point which jumps directly to the list printing routine. Naturally, to be able to use this second entry point, the lists must have already been created and must not have been changed. However, this does make it possible to go back to display or print more copies of the cross references without reading the BASIC program again.

The data in addresses 2B06 and 2B07 are for the page lengths of your console display and hard copy device. The printing routine will pause after printing the specified number of lines and wait for another keyboard input before continuing. This pause can be disabled for either of the output devices by storing a 0 as its page length data.

Each entry in the two main lists in this program, the VLIST and LLIST, consists of three items. First is the link, which is the memory address of the next entry. Second is the data, either the variable or the line number with its good/bad flag. Third is the first entry of another linked list of the line numbers that refer to this data. Each entry in this set of lists, the RLISTs, consists of two items, the link and the line number. The last entry in each of the lists is marked by the link set to zero. These relations are diagrammed in Figure 1. The program keeps all these lists in order as they are created.

Finally, a few words about this assembly listing. This assembler allows a few extra non-standard mnemonics. The ones used in this listing are:

CLA clear accumulator, assembles as XRA A.
SF set flags, assembles as ANA A.
SKP1 skip one byte, assembles as CPI without a data byte.

The DC pseudo op is for Define Characters. It assembles a quoted string as a string of ASCII characters and it sets bit 7 of the last character high to serve as an end-of-message marker. The message display routines in this program are written to use this format.

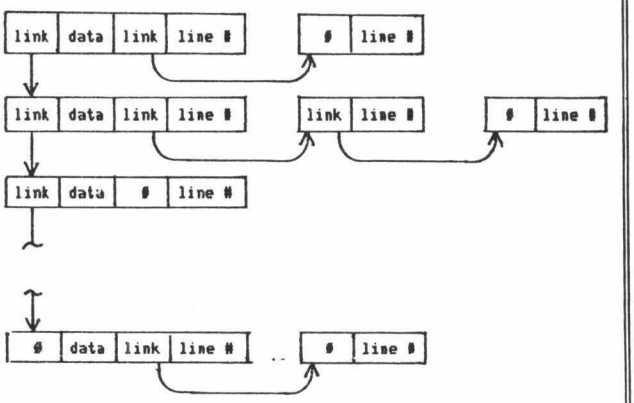


Figure 1

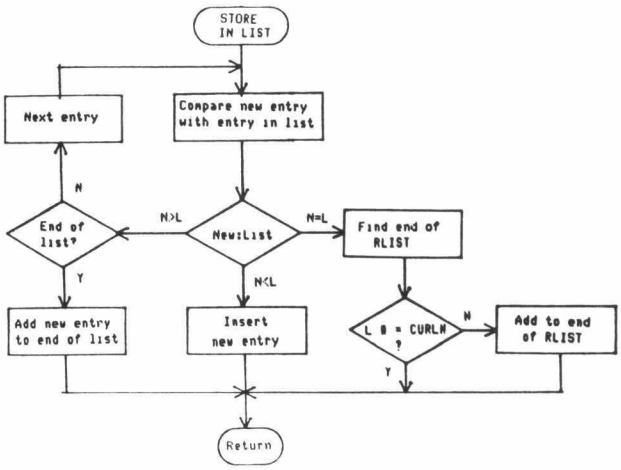


Figure 2b

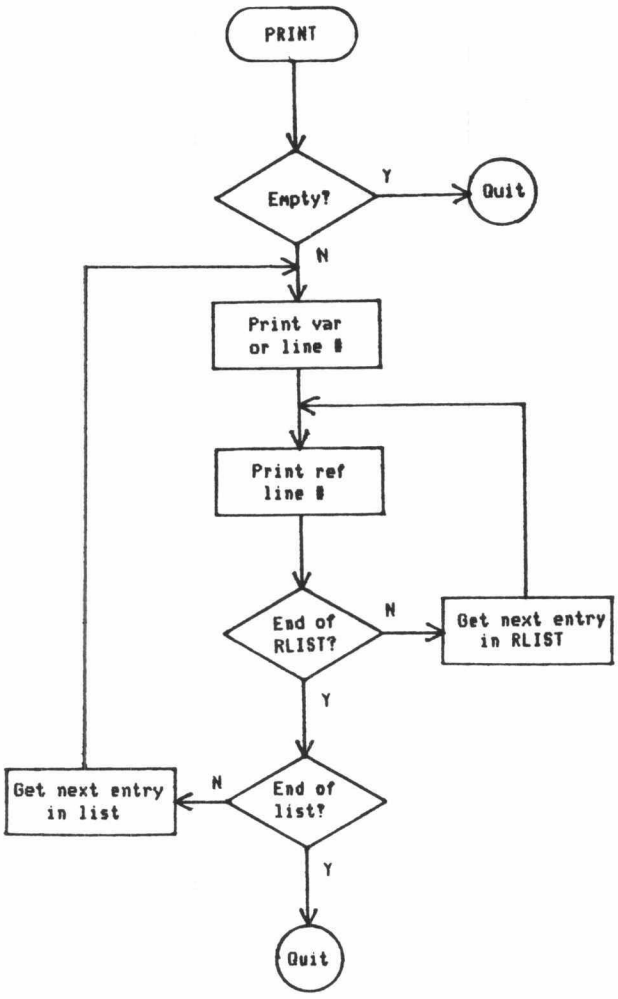


Figure 2a

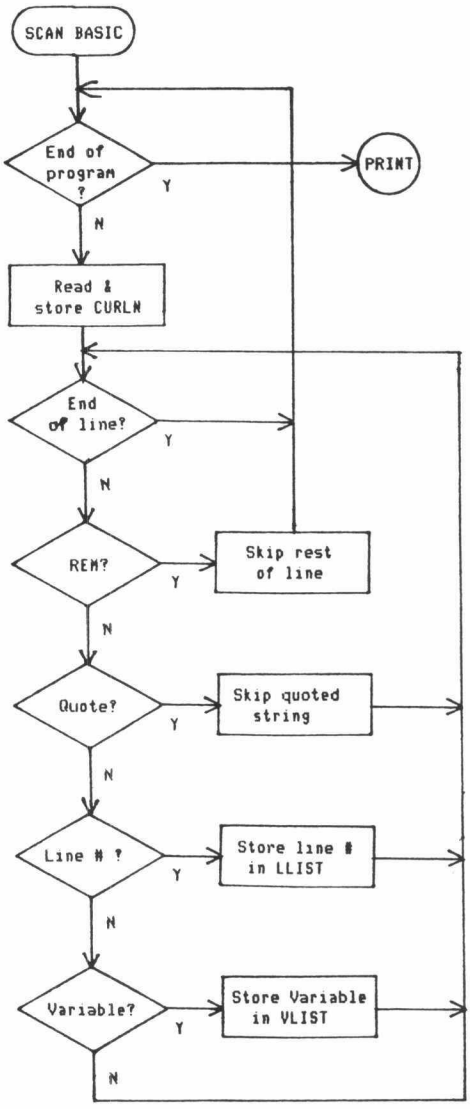


Figure 2c

```
0001                            ;Combined Variable and Line number
0002                            ;  cross reference program for Northstar BASIC.
0003                            ;  written by L Hudson.
0004
0005                                    ORG 2A00H
0006
0007                            ; Disk buffer, must not cross page boundries
0008    2A00 - (256)            BUF     DS 256
0009
0010                            ; Entry points
0011    2B00 - C3 24 2B                 JMP INIT
0012    2B03 - C3 BD 2D                 JMP PRINT
0013
0014                            ; Storage
0015
0016    2B06 - 0F               VLINES  DB 15           ;lines for video
0017    2B07 - 2C               PLINES  DB 44           ;lines for printer
0018    2B08 - (5)              VBUF    DS 5            ;buffer for variable
0019    2B0D - (11)             NBUF    DS 11           ;program name
0020    2B18 - (2)              CURLN   DS 2            ;current line #
0021    2B1A - (2)              AVAIL   DS 2            ;next available addr
0022    2B1C - (2)              BKLNK   DS 2            ;back-up link
0023    2B1E - (1)              VLFLG   DS 1            ;lines or variables
0024    2B1F - (1)              D#      DS 1            ;drive number
0025    2B20 - (2)              DA      DS 2            ;disk address
0026    2B22 - (2)              FDA     DS 2            ;1st disk address
0027                            ; Reuse some of this storage
0028            = 2B08          CTR     = VBUF          ;entries per line
0029            = 2B09          LCTR    = CTR+1         ;line ctr for page
0030            = 2B0A          OCODE   = CTR+2         ;output code
0031            = 2B0B          LINES   = CTR+3         ;lines per page
0032            = 2B0C          TCT     = CTR+4         ;tab counter
0033            = 2B1C          BPTR    = BKLNK         ;pointer to BASIC program
0034
0035                            ; Program starts here
0036
0037    2B24 - 21 00 00         INIT    LXI H 0         ;initialize lists
0038    2B27 - 22 EA 30                 SHLD VLIST
0039    2B2A - 22 EC 30                 SHLD LLIST
0040    2B2D - 21 EE 30                 LXI H LIST
0041    2B30 - 22 1A 2B                 SHLD AVAIL
0042
0043                            ; Input program name, read parameters
0044
0045    2B33 - 21 9B 30         NAMIN   LXI H NAM       ;ask for name
0046    2B36 - CD 48 2F                 CALL VMSG
0047    2B39 - 21 0D 2B                 LXI H NBUF      ;set buffer ptr
0048    2B3C - 01 0B 01                 LXI B 10BH      ;set counters
0049    2B3F - CD F1 2F         NC      CALL CHIN       ;get character
0050    2B42 - FE 5F                    CPI BSP         ;backspace char?
0051    2B44 - CA 63 2B                 JZ BKSTP
0052    2B47 - FE 7F                    CPI DLT         ;ASCII delete?
0053    2B49 - CA 63 2B                 JZ BKSTP        ;this is backspace too
0054    2B4C - CD A3 2F                 CALL VOUT       ;echo char
0055    2B4F - FE 0D                    CPI CR          ;done?
0056    2B51 - CA 76 2B                 JZ NMDN
0057    2B54 - FE 20                    CPI " "
0058    2B56 - DA 3F 2B                 JC NC           ;ignore control chars
0059    2B59 - 77                       MOV MA          ;store char
0060    2B5A - 23                       INX H           ;point to next
0061    2B5B - 0D                       DCR C           ;count char
0062    2B5C - CA 63 2B                 JZ BKSTP        ;too far?
0063    2B5F - 04                       INR B
0064    2B60 - C3 3F 2B                 JMP NC          ;continue
0065    2B63 - 2B               BKSTP   DCX H           ;back up
0066    2B64 - 05                       DCR B
0067    2B65 - C2 6D 2B                 JNZ BOK
0068    2B68 - 23                       INX H           ;too far back
0069    2B69 - 04                       INR B           ;  ignore it
0070    2B6A - C3 3F 2B                 JMP NC
0071    2B6D - 0C               BOK     INR C
0072    2B6E - 3E 5F                    MVI A BSP       ;echo backspace
0073    2B70 - CD A3 2F                 CALL VOUT
0074    2B73 - C3 3F 2B                 JMP NC          ;continue
0075    2B76 - 77               NMDN    MOV MA          ;store cr
0076    2B77 - 21 0D 2B                 LXI H NBUF      ;set registers for
0077    2B7A - 3E 01                    MVI A 1         ;  directory search
0078    2B7C - CD 1C 20                 CALL DLOOK
0079    2B7F - D2 8B 2B                 JNC $+12        ;found?
0080    2B82 - 21 A9 30                 LXI H NFILE     ;no,
0081    2B85 - CD 48 2F                 CALL VMSG       ;  print err msg
0082    2B88 - C3 33 2B                 JMP NAMIN       ;input another name
0083    2B8B - 32 1F 2B                 STA D#          ;save drive number
0084    2B8E - 5E                       MOV EM          ;read disk address
0085    2B8F - 23                       INX H           ;  into D/E
0086    2B90 - 56                       MOV DM
0087    2B91 - 23                       INX H
0088    2B92 - 23                       INX H
0089    2B93 - 23                       INX H           ;point to type
0090    2B94 - 7E                       MOV AM          ;read file type
0091    2B95 - FE 02                    CPI 2           ;BASIC program?
0092    2B97 - CA A3 2B                 JZ $+12
0093    2B9A - 21 B1 30                 LXI H TPER      ;no,
0094    2B9D - CD 48 2F                 CALL VMSG       ;  print error msg
0095    2BA0 - C3 33 2B                 JMP NAMIN       ;go back for another
0096    2BA3 - 21 0D 2B                 LXI H NBUF      ;find end of name
0097    2BA6 - 7E               FEND    MOV AM          ;  to convert to format
0098    2BA7 - FE 0D                    CPI CR          ;  required by the
0099    2BA9 - CA BA 2B                 JZ ENDNM        ;  PMSG routine
0100    2BAC - FE 20                    CPI " "
0101    2BAE - CA BA 2B                 JZ ENDNM
0102    2BB1 - FE 2C                    CPI ","
0103    2BB3 - CA BA 2B                 JZ ENDNM
0104    2BB6 - 23                       INX H
0105    2BB7 - C3 A6 2B                 JMP FEND
0106    2BBA - 2B               ENDNM   DCX H           ;found the end,
0107    2BBB - 7E                       MOV AM          ;  put in stop marker
0108    2BBC - F6 80                    ORI 80H
0109    2BBE - 77                       MOV MA
0110    2BBF - EB                       XCHG
0111    2BC0 - 22 22 2B                 SHLD FDA        ;save disk addr
0112    2BC3 - 22 20 2B                 SHLD DA
0113    2BC6 - CD 14 30                 CALL SCT        ;xfr 1st sector to buffer
0114    2BC9 - D2 D5 2B                 JNC $+12
0115    2BCC - 21 BC 30                 LXI H ERM       ;print error msg
0116    2BCF - CD 48 2F                 CALL VMSG       ;  if bad here
0117    2BD2 - C3 33 2B                 JMP NAMIN
0118
```

```
0119                        ; Read and process data from BASIC program
0120
0121   2BD5 - 21 00 2A              LXI H BUF
0122   2BD8 - CD FB 2F      NXLINE  CALL GETCHR     ;read line length
0123   2BDB - 3D                    DCR A           ;done?
0124   2BDC - CA 74 2D              JZ CHKLN        ;then 2nd pass
0125   2BDF - CD FB 2F              CALL GETCHR     ;  else read line #
0126   2BE2 - 5F                    MOV EA          ;  into D/E
0127   2BE3 - CD FB 2F              CALL GETCHR
0128   2BE6 - 57                    MOV DA
0129   2BE7 - EB                    XCHG            ;save line #
0130   2BE8 - 22 18 2B              SHLD CURLN      ;  as current line
0131   2BEB - EB                    XCHG
0132   2BEC - 11 08 2B              LXI D VBUF      ;preset ptr to var buffer
0133   2BEF - CD FB 2F      NXCHR   CALL GETCHR     ;get character
0134   2BF2 - FE 0D                 CPI CR          ;end of line?
0135   2BF4 - CA D8 2B              JZ NXLINE
0136   2BF7 - FE 8F                 CPI REM         ;remark?
0137   2BF9 - C2 07 2C              JNZ CKQOT       ;no
0138   2BFC - CD FB 2F              CALL GETCHR     ;yes, skip rest of line
0139   2BFF - FE 0D                 CPI CR
0140   2C01 - C2 FC 2B              JNZ $-5
0141   2C04 - C3 D8 2B              JMP NXLINE
0142   2C07 - FE 22         CKQOT   CPI '"'         ;quote?
0143   2C09 - C2 17 2C              JNZ CKLN#       ;no
0144   2C0C - CD FB 2F              CALL GETCHR     ;yes, skip quoted string
0145   2C0F - FE 22                 CPI '"'
0146   2C11 - C2 0C 2C              JNZ $-5
0147   2C14 - C3 EF 2B              JMP NXCHR
0148   2C17 - FE 9A         CKLN#   CPI L#          ;line number reference?
0149   2C19 - C2 2C 2C              JNZ CKVAR       ;no
0150   2C1C - CD FB 2F              CALL GETCHR     ;yes, read line number
0151   2C1F - 4F                    MOV CA          ;  into B/C
0152   2C20 - CD FB 2F              CALL GETCHR
0153   2C23 - 47                    MOV BA
0154   2C24 - E5                    PUSH H          ;save pointer
0155   2C25 - CD 89 2C              CALL STORL      ;store line # into LLIST
0156   2C28 - E1                    POP H           ;restore pointer
0157   2C29 - C3 EC 2B              JMP NXCHR-3     ;  and continue
0158   2C2C - 06 00         CKVAR   MVI B 0         ;flag for variable
0159   2C2E - FE 90                 CPI FN          ;function?
0160   2C30 - C2 3C 2C              JNZ NOFN        ;no
0161   2C33 - 06 E0                 MVI B PAR       ;yes, set flag for FN
0162   2C35 - 12                    STAX D          ;store FN code in buffer
0163   2C36 - 13                    INX D
0164   2C37 - AF                    CLA             ; w/stop code
0165   2C38 - 12                    STAX D
0166   2C39 - CD FB 2F              CALL GETCHR     ;get next character
0167   2C3C - FE 41         NOFN    CPI "A"         ;variable?
0168   2C3E - DA EF 2B              JC NXCHR        ;no
0169   2C41 - FE 5B                 CPI "Z"+1
0170   2C43 - D2 EF 2B              JNC NXCHR       ;no
0171   2C46 - 12                    STAX D          ;store var in buffer
0172   2C47 - 13                    INX D
0173   2C48 - AF                    CLA             ;  w/stop code
0174   2C49 - 12                    STAX D
0175   2C4A - CD FB 2F              CALL GETCHR     ;get next char
0176   2C4D - FE 30                 CPI "0"         ;number?
0177   2C4F - DA 5E 2C              JC NONMR        ;no
0178   2C52 - FE 3A                 CPI "9"+1
0179   2C54 - D2 5E 2C              JNC NONMR
0180   2C57 - 12                    STAX D          ;store number in buffer
0181   2C58 - 13                    INX D
0182   2C59 - AF                    CLA             ;  w/stop code
0183   2C5A - 12                    STAX D
0184   2C5B - CD FB 2F              CALL GETCHR     ;get next char
0185   2C5E - FE E0         NONMR   CPI PAR         ;array variable?
0186   2C60 - C2 70 2C              JNZ NOPAR       ;no
0187   2C63 - B8                    CMP B           ;if function
0188   2C64 - CA 7C 2C              JZ STR          ;  ignore parenthasis
0189   2C67 - 3E 28                 MVI A "("       ;else array,
0190   2C69 - 12                    STAX D          ;  store () in buffer
0191   2C6A - 13                    INX D
0192   2C6B - 3E 29                 MVI A ")"
0193   2C6D - C3 75 2C              JMP SAV
0194   2C70 - FE 24         NOPAR   CPI "$"         ;string variable?
0195   2C72 - C2 7C 2C              JNZ STR         ;no
0196   2C75 - 12            SAV     STAX D          ;store $ or ) in buffer
0197   2C76 - 13                    INX D
0198   2C77 - AF                    CLA             ;  w/stop code
0199   2C78 - 12                    STAX D
0200   2C79 - CD FB 2F              CALL GETCHR
0201   2C7C - F5            STR     PUSH PSW        ;save current (next) char
0202   2C7D - E5                    PUSH H          ;save pointer
0203   2C7E - CD 97 2C              CALL STORV      ;store variable in VLIST
0204   2C81 - E1                    POP H           ;restore pointer
0205   2C82 - F1                    POP PSW         ;restore (next) char
0206   2C83 - 11 08 2B              LXI D VBUF      ;reset buffer pointer
0207   2C86 - C3 F2 2B              JMP NXCHR+3     ;continue
0208
0209                        ; Create linked lists of cross references
0210
0211   2C89 - 21 ED 30      STORL   LXI H LLIST+1   ;preset backup link
0212   2C8C - 22 1C 2B              SHLD BKLNK
0213   2C8F - EB                    XCHG            ;save it
0214   2C90 - 2A EC 30              LHLD LLIST      ;addr of 1st entry
0215   2C93 - AF                    CLA             ;flag for line numbers
0216   2C94 - C3 A3 2C              JMP STR1
0217   2C97 - 21 EB 30      STORV   LXI H VLIST+1   ;preset backup link
0218   2C9A - 22 1C 2B              SHLD BKLNK
0219   2C9D - EB                    XCHG
0220   2C9E - 2A EA 30              LHLD VLIST      ;addr of 1st entry
0221   2CA1 - 3E 01                 MVI A 1         ;flag for variables
0222   2CA3 - 32 1E 2B      STR1    STA VLFLG       ;save flag
0223   2CA6 - 7C                    MOV AH          ;list empty?
0224   2CA7 - B5                    ORA L
0225   2CA8 - C2 B2 2C              JNZ NEXT        ;no
0226   2CAB - 2A 1A 2B              LHLD AVAIL      ;yes, preset reg's
0227   2CAE - EB                    XCHG
0228   2CAF - C3 E3 2C              JMP AD1         ;  & store 1st entry
0229   2CB2 - E5            NEXT    PUSH H          ;save pointer to link
0230   2CB3 - 23                    INX H           ;point to data
0231   2CB4 - 23                    INX H
0232   2CB5 - 3A 1E 2B              LDA VLFLG       ;lines or variables?
0233   2CB8 - 3D                    DCR A
0234   2CB9 - CA C5 2C              JZ NX1          ;variables
0235   2CBC - 5E                    MOV EM          ;line numbers,
0236   2CBD - 23                    INX H           ;  read line #
0237   2CBE - 56                    MOV DM          ;  into D/E
0238   2CBF - CD E9 2E              CALL CPRDB      ;compare
0239   2CC2 - C3 C8 2C              JMP NX2
0240   2CC5 - CD BE 2E      NX1     CALL CPRV       ;cmpr variables
0241   2CC8 - E1            NX2     POP H           ;restore ptr to link
0242   2CC9 - CA 36 2D              JZ STOR         ;new = current entry
0243   2CCC - D2 ED 2C              JNC INSR        ;new < current entry
```

```
0244  2CDF - 5E              MOV EM      ;new > current entry
0245  2CD0 - 23              INX H       ;read link into D/E
0246  2CD1 - 56              MOV DM
0247  2CD2 - 7A              MOV AD      ;end of list?
0248  2CD3 - B3              ORA E
0249  2CD4 - CA DE 2C        JZ ADD      ;yes, add to end
0250  2CD7 - 22 1C 2B        SHLD BKLNK  ;no, save ptr for back-up
0251  2CDA - EB              XCHG        ;H/L=pointer to next entry
0252  2CDB - C3 B2 2C        JMP NEXT
0253
0254  2CDE - EB        ADD   XCHG        ;get next available addr
0255  2CDF - 2A 1A 2B        LHLD AVAIL  ;  into D/E
0256  2CE2 - EB              XCHG
0257  2CE3 - 72        AD1   MOV MD      ;store it as link
0258  2CE4 - 2B              DCX H
0259  2CE5 - 73              MOV ME
0260  2CE6 - C5              PUSH B
0261  2CE7 - 01 00 00        LXI B 0     ;set for end-of-list marker
0262  2CEA - C3 FA 2C        JMP INS1    ;add new data to list
0263
0264  2CED - 2A 1A 2B  INSR  LHLD AVAIL  ;get next available addr
0265  2CF0 - EB              XCHG        ;  into D/E
0266  2CF1 - 2A 1C 2B        LHLD BKLNK  ;H/L=back-up pointer
0267  2CF4 - C5              PUSH B
0268  2CF5 - 46              MOV BM      ;read old link into B/C
0269  2CF6 - 72              MOV MD      ;store next addr as new link
0270  2CF7 - 2B              DCX H
0271  2CF8 - 4E              MOV CM
0272  2CF9 - 73              MOV ME
0273  2CFA - EB        INS1  XCHG        ;store old link
0274  2CFB - 71              MOV MC      ;  into new entry
0275  2CFC - 23              INX H
0276  2CFD - 70              MOV MB
0277  2CFE - C1              POP B       ;restore data
0278  2CFF - 3A 1E 2B        LDA VLFLG   ;lines or variables?
0279  2D02 - 3D              DCR A
0280  2D03 - CA 10 2D        JZ INS2     ;variables
0281  2D06 - 23              INX H       ;lines,
0282  2D07 - 71              MOV MC      ;  store line number
0283  2D08 - 23              INX H       ;  as data
0284  2D09 - 70              MOV MB
0285  2D0A - 23              INX H       ;point to good/bad flag
0286  2D0B - 36 00           MVI M 0     ;set for good
0287  2D0D - C3 22 2D        JMP INS4-1
0288  2D10 - 11 08 2B  INS2  LXI D VBUF  ;variable, set ptr to buf
0289  2D13 - 1A        INS3  LDAX D      ;xfr variable into list
0290  2D14 - A7              SF          ;  as data
0291  2D15 - CA 1E 2D        JZ ENDV
0292  2D18 - 23              INX H
0293  2D19 - 77              MOV MA
0294  2D1A - 13              INX D
0295  2D1B - C3 13 2D        JMP INS3
0296  2D1E - 7E        ENDV  MOV AM      ;set high bit high
0297  2D1F - F6 80           ORI 80H     ;  as end marker
0298  2D21 - 77              MOV MA
0299  2D22 - 23              INX H
0300  2D23 - 36 00     INS4  MVI M 0     ;store end-of-list marker
0301  2D25 - 23              INX H
0302  2D26 - 36 00           MVI M 0
0303  2D28 - EB              XCHG        ;get current line number
0304  2D29 - 2A 18 2B        LHLD CURLN  ;  into D/E
0305  2D2C - EB              XCHG
0306  2D2D - 23              INX H       ;store it as referencing
0307  2D2E - 73              MOV ME      ;  line number
0308  2D2F - 23              INX H
0309  2D30 - 72              MOV MD
0310  2D31 - 23              INX H       ;update next available addr
0311  2D32 - 22 1A 2B        SHLD AVAIL  ;  storage
0312  2D35 - C9              RET
0313
0314  2D36 - 23        STOR  INX H       ;advance pointer
0315  2D37 - 23              INX H       ;  to data
0316  2D38 - 3A 1E 2B        LDA VLFLG   ;lines or variables?
0317  2D3B - 3D              DCR A
0318  2D3C - CA 45 2D        JZ ST1      ;variables
0319  2D3F - 23              INX H       ;skip over line #
0320  2D40 - 23              INX H
0321  2D41 - 23              INX H       ;  & good/bad flag
0322  2D42 - C3 54 2D        JMP ST2+1
0323  2D45 - 7E        ST1   MOV AM      ;skip over variable name
0324  2D46 - FE 90           CPI FN
0325  2D48 - C2 4C 2D        JNZ $+4
0326  2D4B - 23              INX H
0327  2D4C - 7E              MOV AM
0328  2D4D - 23              INX H
0329  2D4E - A7              SF
0330  2D4F - F2 4C 2D        JP $-3
0331  2D52 - FE              SKP1
0332  2D53 - EB        ST2   XCHG        ;read link
0333  2D54 - 5E              MOV EM      ;  into D/E
0334  2D55 - 23              INX H
0335  2D56 - 56              MOV DM
0336  2D57 - 7A              MOV AD      ;end of list?
0337  2D58 - B3              ORA E
0338  2D59 - C2 53 2D        JNZ ST2     ;no, continue searching
0339  2D5C - E5              PUSH H      ;save pointer
0340  2D5D - 23              INX H       ;read line #
0341  2D5E - 5E              MOV EM      ;  into D/E
0342  2D5F - 23              INX H
0343  2D60 - 56              MOV DM
0344  2D61 - 2A 18 2B        LHLD CURLN  ;H/L=current line
0345  2D64 - CD E3 2E        CALL CPRDH  ;compare them
0346  2D67 - D1              POP D       ;restore ptr into D/E
0347  2D68 - C8              RZ          ;done if numbers equal
0348  2D69 - 2A 1A 2B        LHLD AVAIL  ;else add new reference
0349  2D6C - EB              XCHG        ;  to end of list
0350  2D6D - 72              MOV MD      ;update previous end marker
0351  2D6E - 2B              DCX H       ;  to be new link
0352  2D6F - 73              MOV ME
0353  2D70 - EB              XCHG
0354  2D71 - C3 23 2D        JMP INS4    ;continue
0355
0356                    ; Second pass to check for bad line numbers
0357
0358  2D74 - 2A 22 2B  CHKLN LHLD FDA    ;1st disk addr
0359  2D77 - 22 20 2B        SHLD DA
0360  2D7A - CD 14 30        CALL SCT    ;get 1st sector
0361  2D7D - DA 3B 30        JC ERR
0362  2D80 - 21 00 2A        LXI H BUF   ;set buffer pointer
0363  2D83 - 22 1C 2B        SHLD BPTR
0364  2D86 - CD 14 2F        CALL RDBLN  ;get 1st BASIC line #
0365  2D89 - 2A EC 30        LHLD LLIST  ;pointer to 1st entry
0366  2D8C - 7C              MOV AH      ;list empty?
0367  2D8D - B5              ORA L
0368  2D8E - CA BD 2D        JZ PRINT    ;then go to print routine
```

(Continued on page 38.)

(Continued from page 36.)

```
0369  2D91 - 4E          CHK1    MOV CM          ;read link
0370  2D92 - 23                  INX H           ;  into B/C
0371  2D93 - 46                  MOV BM
0372  2D94 - 23                  INX H
0373  2D95 - 5E                  MOV EM          ;read line number
0374  2D96 - 23                  INX H           ;  into D/E
0375  2D97 - 56                  MOV DM
0376  2D98 - 23                  INX H
0377  2D99 - E5                  PUSH H          ;save ptr to good/bad flag
0378  2D9A - 2A 18 2B     CHK2    LHLD CURLN      ;current BASIC line #
0379  2D9D - CD E3 2E             CALL CPRDH      ;compare list to BASIC
0380  2DA0 - CA B2 2D             JZ OK           ;match, good line #
0381  2DA3 - DA AC 2D             JC BAD          ;BASIC > list, no good
0382  2DA6 - CD 14 2F             CALL RDBLN      ;BASIC < list, continue search
0383  2DA9 - C3 9A 2D             JMP CHK2        ;  continue search
0384  2DAC - E1           BAD     POP H           ;restore good/bad ptr
0385  2DAD - 36 01                MVI M 1         ;flag for bad
0386  2DAF - C3 B6 2D             JMP OK+4
0387  2DB2 - CD 14 2F      OK     CALL RDBLN      ;get next BASIC line #
0388  2DB5 - E1                   POP H
0389  2DB6 - 60                   MOV HB          ;link into H/L
0390  2DB7 - 69                   MOV LC
0391  2DB8 - 78                   MOV AB          ;end of list?
0392  2DB9 - B1                   ORA C
0393  2DBA - C2 91 2D             JNZ CHK1        ;no, continue comparing
0394
0395                        ; Print lists
0396
0397  2DBD - 21 AF 2E      PRINT  LXI H AGN       ;set artificial return addr
0398  2DC0 - E5                   PUSH H
0399  2DC1 - 21 44 30             LXI H WCH       ;ask lines or variables
0400  2DC4 - CD 48 2F             CALL VMSG
0401  2DC7 - CD F1 2F      PT1    CALL CHIN       ;get answer
0402  2DCA - E6 5F                ANI 5FH         ;be sure it is upper case
0403  2DCC - FE 56                CPI "V"         ;variables?
0404  2DCE - CA D6 2D             JZ PT2          ;yes
0405  2DD1 - FE 4C                CPI "L"         ;lines?
0406  2DD3 - C2 C7 2D             JNZ PT1         ;ignore anything else
0407  2DD6 - CD A3 2F      PT2    CALL VOUT       ;print answer
0408  2DD9 - FE 56                CPI "V"         ;lines?
0409  2DDB - 3E 01                MVI A 1         ;set for variables
0410  2DDD - CA E1 2D             JZ $+4          ;yes, variables
0411  2DE0 - AF                   CLA             ;no, lines
0412  2DE1 - 32 1E 2B             STA VLFLG       ;store flag
0413  2DE4 - 3E 0D                MVI A CR        ;then print crlf
0414  2DE6 - CD A3 2F             CALL VOUT
0415  2DE9 - 21 61 30             LXI H PRTM      ;want hard copy?
0416  2DEC - CD 48 2F             CALL VMSG
0417  2DEF - CD 22 30             CALL YN         ;get answer
0418  2DF2 - 3E 00                MVI A 0         ;set for printer off
0419  2DF4 - C2 F9 2D             JNZ $+5
0420  2DF7 - 3E 01                MVI A PCODE     ;set for printer on
0421  2DF9 - 32 0A 2B             STA OCODE       ;store code
0422  2DFC - 3A 06 2B             LDA VLINES      ;line count for video
0423  2DFF - C2 05 2E             JNZ $+6
0424  2E02 - 3A 07 2B             LDA PLINES      ;line count for printer
0425  2E05 - 32 0B 2B             STA LINES       ;save count
0426  2E08 - 32 09 2B             STA LCTR        ;set counter
0427  2E0B - 3A 1E 2B             LDA VLFLG       ;lines or variables?
0428  2E0E - 3D                   DCR A
0429  2E0F - CA 1B 2E             JZ PRV          ;variables
0430  2E12 - 2A EC 30             LHLD LLIST      ;starting addr of list
0431  2E15 - 11 C2 30             LXI D LM        ;"Lines" msg
0432  2E18 - C3 21 2E             JMP PRV+6
0433  2E1B - 2A EA 30      PRV    LHLD VLIST      ;set for variable list
0434  2E1E - 11 CD 30             LXI D VM        ;"Variables" msg
0435  2E21 - 7C                   MOV AH          ;empty?
0436  2E22 - B5                   ORA L
0437  2E23 - C8                   RZ              ;yes, quit
0438  2E24 - EB                   XCHG            ;print heading
0439  2E25 - CD 52 2F             CALL PMSG
0440  2E28 - 21 D5 30             LXI H CM
0441  2E2B - CD 52 2F             CALL PMSG
0442  2E2E - 21 0D 2B             LXI H NBUF
0443  2E31 - CD 52 2F             CALL PMSG
0444  2E34 - CD EF 2E             CALL PCRLF
0445  2E37 - EB                   XCHG
0446
0447                        ; Now print the list
0448
0449  2E38 - 4E           NXT     MOV CM          ;read link
0450  2E39 - 23                   INX H           ;  into B/C
0451  2E3A - 46                   MOV BM
0452  2E3B - 23                   INX H
0453  2E3C - 3E 07                MVI A WIDTH     ;set line width ctr
0454  2E3E - 32 08 2B             STA CTR
0455  2E41 - 3A 1E 2B             LDA VLFLG       ;lines or variables?
0456  2E44 - 3D                   DCR A
0457  2E45 - CA 65 2E             JZ PT3          ;variables
0458  2E48 - 5E                   MOV EM          ;lines, read line #
0459  2E49 - 23                   INX H           ;  into D/E
0460  2E4A - 56                   MOV DM
0461  2E4B - 23                   INX H
0462  2E4C - 7E                   MOV AM          ;read good/bad flag
0463  2E4D - 23                   INX H
0464  2E4E - 3D                   DCR A
0465  2E4F - C2 57 2E             JNZ $+8         ;good
0466  2E52 - 3E 2A                MVI A "*"       ;bad
0467  2E54 - CD 98 2F             CALL POUT       ;print asterisk
0468  2E57 - EB                   XCHG            ;print line #
0469  2E58 - CD 5C 2F             CALL BINDEC
0470  2E5B - 21 99 30             LXI H DSHM      ;followed by a dash
0471  2E5E - CD 52 2F             CALL PMSG
0472  2E61 - EB                   XCHG
0473  2E62 - C3 79 2E             JMP NXLN#
0474  2E65 - 7E           PT3     MOV AM          ;read 1st char of var
0475  2E66 - FE 90                CPI FN          ;function?
0476  2E68 - C2 76 2E             JNZ NFN         ;no
0477  2E6B - 3E 46                MVI A "F"       ;yes, print FN
0478  2E6D - CD 98 2F             CALL POUT
0479  2E70 - 3E 4E                MVI A "N"
0480  2E72 - CD 98 2F             CALL POUT
0481  2E75 - 23                   INX H           ;point to next char
0482  2E76 - CD 52 2F      NFN    CALL PMSG       ;print var name
0483  2E79 - 5E            NXLN#  MOV EM          ;read link from ref list
0484  2E7A - 23                   INX H           ;  into D/E
0485  2E7B - 56                   MOV DM
0486  2E7C - D5                   PUSH D          ;and save it
0487  2E7D - 23                   INX H           ;read line #
0488  2E7E - 5E                   MOV EM          ;  into D/E
0489  2E7F - 23                   INX H
0490  2E80 - 56                   MOV DM
0491  2E81 - 3E 09                MVI A TAB       ;print tab
0492  2E83 - CD 98 2F             CALL POUT
0493  2E86 - EB                   XCHG
```

```
0494  2E87 - CD 5C 2F          CALL BINDEC     ;print line number
0495  2E8A - 3A 08 2B          LDA CTR         ;count entries per line
0496  2E8D - 3D                DCR A
0497  2E8E - C2 96 2E          JNZ PT4         ;line not full
0498  2E91 - CD EF 2E          CALL PCRLF      ;line full, print crlf
0499  2E94 - 3E 07             MVI A WIDTH     ;reset counter
0500  2E96 - 32 08 2B    PT4   STA CTR         ;save count
0501  2E99 - E1                POP H           ;restore link
0502  2E9A - 7C                MOV AH          ;end of ref list?
0503  2E9B - B5                ORA L
0504  2E9C - C2 79 2E          JNZ NXLN#       ;no, print next
0505  2E9F - 3A 08 2B          LDA CTR         ;suppress crlf if nec
0506  2EA2 - FE 07             CPI WIDTH
0507  2EA4 - C4 EF 2E          CNZ PCRLF
0508  2EA7 - 78                MOV AB          ;end of whole list?
0509  2EA8 - B1                ORA C
0510  2EA9 - C8                RZ              ;yes
0511  2EAA - 60                MOV HB          ;no, put link
0512  2EAB - 69                MOV LC          ;  into H/L
0513  2EAC - C3 38 2E          JMP NXT         ;print next line or variable
0514
0515                     ; Check if more lists are wanted
0516
0517  2EAF - 21 70 30    AGN   LXI H AGNM      ;ask again?
0518  2EB2 - CD 48 2F          CALL VMSG
0519  2EB5 - CD 22 30          CALL YN         ;get answer
0520  2EB8 - CA BD 2D          JZ PRINT        ;print another list
0521  2EBB - C3 41 30          JMP ABRT        ;or return to DOS
0522
0523                     ;       ******** SUBROUTINES ********
0524
0525                     ; Compare variable in VBUF and in VLIST
0526                     ;   On return, zero set if equal.
0527                     ;   Carry clear if buf<list,
0528                     ;   carry set if buf>list.
0529
0530  2EBE - EB          CPRV  XCHG            ;list pointer into D/E
0531  2EBF - 21 08 2B          LXI H VBUF      ;buffer pointer into H/L
0532  2EC2 - 1A                LDAX D          ;1st char
0533  2EC3 - FE 90             CPI FN          ;function code?
0534  2EC5 - C2 CC 2E          JNZ CPLP        ;no
0535  2EC8 - BE                CMP M           ;yes, buffer too?
0536  2EC9 - C0                RNZ             ;done if not also function
0537  2ECA - 23                INX H           ;else continue comparing
0538  2ECB - 13                INX D
0539  2ECC - 1A          CPLP  LDAX D          ;get char from list
0540  2ECD - E6 7F             ANI 7FH         ;be sure bit 7 is low
0541  2ECF - BE                CMP M           ;compare to char in buffer
0542  2ED0 - C0                RNZ             ;done if not same
0543  2ED1 - 1A                LDAX D          ;get char again
0544  2ED2 - A7                SF              ;chk bit 7
0545  2ED3 - 23                INX H           ;increment pointers
0546  2ED4 - 13                INX D           ;  for next chars
0547  2ED5 - 7E                MOV AM          ;char from buffer
0548  2ED6 - FA DF 2E          JM ENDL
0549  2ED9 - A7                SF              ;end mark in buffer?

0550  2EDA - C2 CC 2E          JNZ CPLP        ;no, continue comparing
0551  2EDD - 3C                INR A           ;buf<list, clear zero flag
0552  2EDE - C9                RET             ;  and return
0553  2EDF - A7          ENDL  SF              ;end mark in buffer?
0554  2EE0 - C8                RZ              ;yes, buf=list
0555  2EE1 - 37                STC             ;no, buf>list
0556  2EE2 - C9                RET
0557
0558                     ; 16 bit unsigned comparison of H/L and D/E.
0559                     ; If H/L=D/E then zero set, if H/L<=D/E carry clear,
0560                     ; if H/L>D/E carry set.
0561
0562  2EE3 - 7A          CPRDH MOV AD
0563  2EE4 - BC                CMP H
0564  2EE5 - C0                RNZ
0565  2EE6 - 7B                MOV AE
0566  2EE7 - BD                CMP L
0567  2EE8 - C9                RET
0568
0569                     ; Same comparison, but between B/C and D/E.
0570
0571  2EE9 - 7A          CPRDB MOV AD
0572  2EEA - B8                CMP B
0573  2EEB - C0                RNZ
0574  2EEC - 7B                MOV AE
0575  2EED - B9                CMP C
0576  2EEE - C9                RET
0577
0578                     ; Print cr, lf and count lines
0579                     ;  pause if end of page.
0580
0581  2EEF - 3E 0D       PCRLF MVI A CR        ;print crlf
0582  2EF1 - CD 98 2F          CALL POUT
0583  2EF4 - 3A 09 2B          LDA LCTR        ;get line counter
0584  2EF7 - A7                SF              ;zero?
0585  2EF8 - C8                RZ              ;  then no pause
0586  2EF9 - 3D                DCR A           ;count lines
0587  2EFA - C2 10 2F          JNZ CONT        ;continue if not end of page
0588  2EFD - E5                PUSH H          ;save pointer
0589  2EFE - 21 7E 30          LXI H PAUSE     ;print pause message
0590  2F01 - CD 48 2F          CALL VMSG       ;to video only
0591  2F04 - CD F1 2F          CALL CHIN       ;keyboard input
0592  2F07 - 3E 0D             MVI A CR        ;no, print another crlf
0593  2F09 - CD A3 2F          CALL VOUT       ;  to video
0594  2F0C - E1                POP H           ;restore pointer
0595  2F0D - 3A 0B 2B          LDA LINES       ;reset counter
0596  2F10 - 32 09 2B    CONT  STA LCTR
0597  2F13 - C9                RET
0598
0599                     ; Read line numbers from BASIC program
0600                     ; to check for bad line number references.
0601
0602  2F14 - E5          RDBLN PUSH H          ;save registers
0603  2F15 - D5                PUSH D
0604  2F16 - 2A 1C 2B          LHLD BPTR       ;ptr to line length
0605  2F19 - 11 FF FF          LXI D 0FFFFH    ;set line # to max
0606  2F1C - CD FB 2F          CALL GETCHR     ;read line length
0607  2F1F - 3D                DCR A           ;end of program?
0608  2F20 - CA 41 2F          JZ HI           ;yes, use max line #
0609  2F23 - F5                PUSH PSW        ;no, save length-1
0610  2F24 - CD FB 2F          CALL GETCHR     ;read line number
0611  2F27 - 5F                MOV EA          ;  into D/E
0612  2F28 - CD FB 2F          CALL GETCHR
0613  2F2B - 57                MOV DA
0614  2F2C - F1                POP PSW         ;restore length
0615  2F2D - D6 02             SUI 2           ;compensate for reading
0616  2F2F - 85                ADD L           ;calculate ptr to next line
0617  2F30 - 6F                MOV LA
0618  2F31 - 22 1C 2B          SHLD BPTR       ;save it
```

ending

```
0619  2F34 - D2 41 2F          JNC HI          ;done if in same sector
0620  2F37 - D5                PUSH D          ;else get next sector
0621  2F38 - C5                PUSH B
0622  2F39 - CD 0D 30          CALL SECT
0623  2F3C - DA 3B 30          JC ERR
0624  2F3F - C1                POP B
0625  2F40 - D1                POP D
0626  2F41 - EB          HI    XCHG            ;line number into H/L
0627  2F42 - 22 18 2B          SHLD CURLN      ;save it
0628  2F45 - D1                POP D           ;restore registers
0629  2F46 - E1                POP H
0630  2F47 - C9                RET
0631
0632                     ; Print messages
0633
0634  2F48 - 7E          VMSG  MOV AM          ;get char
0635  2F49 - 23                INX H           ;point to next
0636  2F4A - CD A3 2F          CALL VOUT       ;print char
0637  2F4D - A7                SF              ;check for end mark
0638  2F4E - F2 48 2F          JP VMSG         ;loop if not done
0639  2F51 - C9                RET
0640  2F52 - 7E          PMSG  MOV AM          ;same routine as above
0641  2F53 - 23                INX H           ;  except output to printer
0642  2F54 - CD 98 2F          CALL POUT       ;  if it is enabled
0643  2F57 - A7                SF
0644  2F58 - F2 52 2F          JP PMSG
0645  2F5B - C9                RET
0646
0647                     ; Print H/L registers as decimal number.
0648                     ;  All registers except accumulator saved.
0649
0650  2F5C - E5          BINDEC PUSH H         ;save all registers
0651  2F5D - D5                PUSH D
0652  2F5E - C5                PUSH B
0653  2F5F - 01 00 00          LXI B 0
0654  2F62 - 11 F0 D8          LXI D -10000
0655  2F65 - CD 83 2F          CALL DISP       ;print 10000's digit
0656  2F68 - 11 18 FC          LXI D -1000
0657  2F6B - CD 83 2F          CALL DISP       ;print 1000's digit
0658  2F6E - 11 9C FF          LXI D -100
0659  2F71 - CD 83 2F          CALL DISP       ;  etc.
0660  2F74 - 11 F6 FF          LXI D -10
0661  2F77 - CD 83 2F          CALL DISP
0662  2F7A - 7D                MOV AL          ;and finally,
0663  2F7B - CD 96 2F          CALL DEC        ;  print the units digit
0664  2F7E - C1                POP B           ;restore all registers
0665  2F7F - D1                POP D
0666  2F80 - E1                POP H
0667  2F81 - C9                RET
0668  2F82 - 0C                INR C           ;calculate digit
0669  2F83 - 19          DISP  DAD D           ;  by repeated subtraction
0670  2F84 - DA 82 2F          JC DISP-1       ;  until result is negative
0671  2F87 - 7B                MOV AE          ;negate D/E
0672  2F88 - 2F                CMA             ;  to compensate
0673  2F89 - 5F                MOV EA          ;  for last subtraction
0674  2F8A - 7A                MOV AD          ;  which was one to far
0675  2F8B - 2F                CMA
0676  2F8C - 57                MOV DA
0677  2F8D - 13                INX D
0678  2F8E - 19                DAD D
0679  2F8F - 78                MOV AB          ;suppress leading zeros
0680  2F90 - B1                ORA C           ;  if B is zero, all digits
0681  2F91 - 47                MOV BA          ;  have been zeros so far
0682  2F92 - 79                MOV AC          ;get digit
0683  2F93 - 0E 00             MVI C 0         ;reset digit counter
0684  2F95 - C8                RZ              ;no printing if zero
0685  2F96 - C6 30      DEC   ADI "0"         ;convert digit to ASCII
0686                           ;  and fall into POUT
0687
0688                     ; Character output, using standard Northstar
0689                     ; DOS output routines.  Will handle tabs and
0690                     ; adds auto line feed following a carriage return.
0691                     ; Saves and restores processor flags as well as
0692                     ; original character.
0693
0694  2F98 - F5          POUT  PUSH PSW        ;save original char
0695  2F99 - C5                PUSH B          ;save B/C
0696  2F9A - E6 7F             ANI 7FH         ;strip bit 7
0697  2F9C - 47                MOV BA          ;char into B
0698  2F9D - 3A 0A 2B          LDA OCODE       ;get output code
0699  2FA0 - C3 A9 2F          JMP OUT
0700  2FA3 - F5          VOUT  PUSH PSW        ;save original char
0701  2FA4 - C5                PUSH B          ;save B/C
0702  2FA5 - E6 7F             ANI 7FH         ;strip bit 7

0703  2FA7 - 47                MOV BA          ;char into B
0704  2FA8 - AF                CLA             ;output code = 0
0705  2FA9 - F5          OUT   PUSH PSW        ;save code
0706  2FAA - 78                MOV AB          ;char back into A
0707  2FAB - FE 09             CPI TAB         ;tab character?
0708  2FAD - C2 CA 2F          JNZ NTAB        ;no
0709  2FB0 - 3A 0C 2B          LDA TCT         ;yes, get tab counter
0710  2FB3 - 4F                MOV CA          ;  into C
0711  2FB4 - 06 20             MVI B " "       ;space into B
0712  2FB6 - F1          TOUT  POP PSW         ;restore output code
0713  2FB7 - F5                PUSH PSW        ; & resave
0714  2FB8 - CD 0D 20          CALL COUT       ;output space
0715  2FBB - 0C                INR C           ;count tabs
0716  2FBC - 3E 07             MVI A 7
0717  2FBE - A1                ANA C
0718  2FBF - 4F                MOV CA
0719  2FC0 - C2 B6 2F          JNZ TOUT        ;continue if not done
0720  2FC3 - 32 0C 2B          STA TCT         ;reset tab counter
0721  2FC6 - F1                POP PSW
0722  2FC7 - C3 EE 2F          JMP CDONE
0723  2FCA - FE 0D       NTAB  CPI CR          ;not tab, then cr?
0724  2FCC - C2 E1 2F          JNZ NCR         ;no
0725  2FCF - F1                POP PSW         ;yes, print cr
0726  2FD0 - F5                PUSH PSW
0727  2FD1 - CD 0D 20          CALL COUT
0728  2FD4 - 06 0A             MVI B LF        ;print line feed also
0729  2FD6 - F1                POP PSW
0730  2FD7 - CD 0D 20          CALL COUT
0731  2FDA - AF                CLA             ;reset tab counter
0732  2FDB - 32 0C 2B          STA TCT
0733  2FDE - C3 EE 2F          JMP CDONE
0734  2FE1 - F1          NCR   POP PSW         ;not cr, print character
0735  2FE2 - CD 0D 20          CALL COUT
0736  2FE5 - 3A 0C 2B          LDA TCT         ;increment tab counter
0737  2FE8 - 3C                INR A
0738  2FE9 - E6 07             ANI 7           ;keep 3 low bits only
0739  2FEB - 32 0C 2B          STA TCT
0740  2FEE - C1          CDONE POP B           ;restore B/C
0741  2FEF - F1                POP PSW         ;restore original char & flags
0742  2FF0 - C9                RET
0743
```

(Continued on page 42.)

(Continued from page 40.)

```
                        ; Character input, abort if Ctrl-C
0745
0746    2FF1 - AF       CHIN    CLA             ;standard Northstar
0747    2FF2 - CD 10 20         CALL CIN        ;  char input
0748    2FF5 - FE 03            CPI CTLC        ;control C?
0749    2FF7 - C0               RNZ             ;ok if not
0750    2FF8 - C3 41 30         JMP ABRT        ;abort if so
0751
0752                    ; Get next character from disk buffer,
0753                    ;   get next disk sector if necessary


0754
0755    2FFB - 7E       GETCHR  MOV AM          ;get character
0756    2FFC - 2C               INR L           ;point to next
0757    2FFD - C0               RNZ             ;done if same sector
0758    2FFE - E5               PUSH H          ;need to get next sector,
0759    2FFF - D5               PUSH D          ;  save all registers
0760    3000 - C5               PUSH B
0761    3001 - F5               PUSH PSW
0762    3002 - CD 0D 30         CALL SECT       ;xfr sector to buffer
0763    3005 - DA 3B 30         JC ERR          ;abort if problem
0764    3008 - F1               POP PSW         ;restore everything
0765    3009 - C1               POP B
0766    300A - D1               POP D
0767    300B - E1               POP H
0768    300C - C9               RET
0769
0770                    ; Transfer next disk sector to buffer
0771
0772    300D - 2A 20 2B SECT    LHLD DA         ;get disk addr
0773    3010 - 23               INX H           ;set to next sector
0774    3011 - 22 20 2B         SHLD DA         ;  & save
0775    3014 - 11 00 2A SCT     LXI D BUF       ;set registers
0776    3017 - 06 01            MVI B READ      ;  for DCOM
0777    3019 - 3A 1F 2B         LDA D#
0778    301C - 4F               MOV CA
0779    301D - 3E 01            MVI A 1
0780    301F - C3 22 20         JMP DCOM        ;xfr sector and return
0781
0782                    ; Yes or no?  Accepts only Y or N, ignores
0783                    ;  anything else.  On return, zero set if Y,
0784                    ;  zero clear if N.  Requires that processor
0785                    ;  flags are saved by the output routine.
0786
0787    3022 - CD F1 2F YN      CALL CHIN       ;get keyboard input
0788    3025 - E6 5F            ANI 5FH         ;be sure it is upper case
0789    3027 - FE 59            CPI "Y"         ;answer yes?
0790    3029 - CA 32 30         JZ YES
0791    302C - FE 4E            CPI "N"         ;answer no?
0792    302E - C2 22 30         JNZ YN          ;ignore anything else
0793    3031 - A7               SF              ;clear zero flag
0794    3032 - CD A3 2F YES     CALL VOUT       ;print Y or N
0795    3035 - 3E 0D            MVI A CR        ;  followed by crlf
0796    3037 - CD A3 2F         CALL VOUT
0797    303A - C9               RET
0798
0799                    ; Disk error abort
0800
0801    303B - 21 BC 30 ERR     LXI H ERM       ;print error msg
0802    303E - CD 48 2F         CALL VMSG
0803    3041 - C3 28 20 ABRT    JMP DOS         ;and abort back to DOS
0804
```

```
0805                    ;       ******** MESSAGES ********
0806
0807    3044 - 56 61 72 WCH     DC "Variables or Lines (V or L)? "
        3047 - 69 61 62
        304A - 6C 65 73
        304D - 20 6F 72
        3050 - 20 4C 69
        3053 - 6E 65 73
        3056 - 20 28 56
        3059 - 20 6F 72
        305C - 20 4C 29
        305F - 3F A0
0808    3061 - 57 61 6E PRTM    DC "Want printout? "
        3064 - 74 20 70
        3067 - 72 69 6E
        306A - 74 6F 75
        306D - 74 3F A0
0809    3070 - 41 6E 6F AGNM    DC "Another list? "
        3073 - 74 68 65
        3076 - 72 20 6C
        3079 - 69 73 74
        307C - 3F A0
0810    307E - 07               PAUSE   DB BEL
0811    307F - 48 69 74         DC "Hit any key to continue..."
        3082 - 20 61 6E
        3085 - 79 20 6B
        3088 - 65 79 20
        308B - 74 6F 20
        308E - 63 6F 6E
        3091 - 74 69 6E
        3094 - 75 65 2E
        3097 - 2E AE
0812    3099 - 20 AD            DSHM    DC " -"
0813    309B - 50 72 6F NAM     DC "Program name? "
        309E - 67 72 61
        30A1 - 6D 20 6E
        30A4 - 61 6D 65
        30A7 - 3F A0
0814    30A9 - 4E 6F 20 NFILE   DB "No file",CR+80H
        30AC - 66 69 6C
        30AF - 65 8D
0815    30B1 - 54 79 70 TPER    DB "Type error",CR+80H
        30B4 - 65 20 65
        30B7 - 72 72 6F
        30BA - 72 8D
0816    30BC - 45 72 72 ERM     DB "Error",CR+80H
        30BF - 6F 72
        30C1 - 8D
0817    30C2 - 4C 69 6E LM      DC "Line number"
        30C5 - 65 20 6E
        30C8 - 75 6D 62
        30CB - 65 F2
0818    30CD - 56 61 72 VM      DC "Variable"
        30D0 - 69 61 62
        30D3 - 6C E5
0819    30D5 - 20 63 72 CM      DC " cross reference for "
        30D8 - 6F 73 73
        30DB - 20 72 65
        30DE - 66 65 72
        30E1 - 65 6E 63
        30E4 - 65 20 66
        30E7 - 6F 72 A0
0820
```

```
0821                            ;       ******** LISTS START HERE ********
0822
0823    30EA - (2)              VLIST   DS 2    ;pointer to 1st variable entry
0824    30EC - (2)              LLIST   DS 2    ;pointer to 1st line # entry
0825    30EE - (1)              LIST    DS 1    ;lists start here
0826
0827                            ; Equates
0828
0829            = 0003          CTLC    = 3
0830            = 0007          BEL     = 7
0831            = 0009          TAB     = 9
0832            = 000A          LF      = 0AH
0833            = 000D          CR      = 0DH
0834            = 005F          BSP     = 5FH
0835            = 007F          DLT     = 7FH
0836            = 008F          REM     = 143
0837            = 0090          FN      = 144
0838            = 00E0          PAR     = 224
0839            = 009A          L#      = 154
0840            = 200D          COUT    = 200DH
0841            = 2010          CIN     = 2010H
0842            = 201C          OLOOK   = 201CH
0843            = 2022          DCOM    = 2022H
0844            = 2028          DOS     = 2028H
0845            = 0001          READ    = 1     ;read command for DCOM
0846            = 0001          PCODE   = 1     ;printer output code
0847            = 0007          WIDTH   = 7     ;# entries per line
```

the end

Solution of Three First Order Differential Equations with an SR-52 Programmable Calculator

BY S. ANDREW YAKUSH PH.D.
Department of Biomathematics
School of Medicine
Loma Linda University
Loma Linda, CA 93250

Abstract

This paper presents a program written for the Texas Instruments SR-52 programmable calculator to solve three first order ordinary differential equations using a 4th order Runga-Kutta method. General instructions for use of the program and a program listing are given and a typical sample problem done.

Introduction

In mathematical modeling and espe cially the area of biological modeling, effective models have been developed using three first order ordinary differential equations of the form:

$$\begin{aligned} x' &= f(t, x, y, z) & x(t_0) &= x_0 \\ y' &= g(t, x, y, z) & y(t_0) &= y_0 \\ z' &= h(t, x, y, z) & z(t_0) &= z_0 \end{aligned}$$

It is often helpful to be able to do some preliminary examination of the model before employing a computer to do the major simulation. This paper presents a program written for the Texas Instruments SR-52 programmable calculator to solve this set of equations using a 4th order Runga-Kutta method (*1*).

Program Description

The user is required to define the three functions f, g and h using the labels C, D and E. There are 61 program steps (162-223) and 9 data registers (11-19) available for user in defining the functions. For use in the functions f, g and h the current values of x, y and z are found in the registers R_{02}, R_{03} and R_{04}. The initial time t_0 is stored in register R_{01} and the initial states x_0, y_0 and z_0 are stored in registers R_{05}, R_{06} and R_{07} respectively. One half the integration step size H/2 is stored in register R_{98} and the number of integration steps is stored in register R_{00}.

The program is designed to use a PC 100 printer to print the values t_i, $x(t_i)$, $y(t_i)$ and $z(t_i)$ for $i = 1, \ldots, N$ where $t_i = iH$. If a printer is not available, program steps 147, 151, 155 and 159 can be changed from PNT to HLT (see Table 2). This will cause the program to halt after each integration step and display the appropriate values, then continue when RUN is pressed. Each integration step requires about 18 seconds in addition to the time required to execute each of the functions four times.

Table 1 contains the general instructions for use of the program. Table 2 is a listing of the general program.

Typical Sample Run

A model for the spread of an epidemic and its control is described by the following equations(2):

$$\begin{aligned} S' &= -cSI - v(t) & S(0) &= S_0 \\ I' &= cSI - rI & I(0) &= I_0 \\ R' &= rI & R(0) &= 0 \\ V' &= v(t) & V(0) &= 0 \end{aligned}$$

where S is the proportion of susceptibles in the population, I is the proportion of infectives, R is the proportion of recovered, V is the proportion who have been vaccinated, c is the contact rate, r the recovery rate and v(t) the vaccination rate and $S_0 + I_0 = 1$. Since $S + I + R + V = 1$, we need solve only the first three equations, hence this program is applicable. For the sake of simplicity we will assume that v(t) is a constant.

Forty nine program steps were required to define the three functions (see Table 3). Table 4 shows excerpts of the sample run which required a total of about 6 minutes for execution which is about 20 seconds per step. The parameters chosen for this example are those of the 6th case in Table 1 of (2).

Comments

The execution time for this program is quite long even when compared to a minicomputer. However, the convenience can often outweigh the slowness, or make it possible to solve a problem even when a computer is not available. Another advantage is that the 12 digit accuracy of the SR-52 is equivalent to double precision on a 16 bit minicomputer. Since there are only 63 program steps available and 9 of these are used for label definition and returns, only relatively simple models can be simulated. But as the sample shows, there are meaningful models which fit this criterion.

Programs to solve two first order equations, a second order equation, or a third order equation using the 4th order Runge-Kutta method are available from the author.

References

1. Bekey, G. A., and Karplus, W. J., "Hybrid Computation." John Wiley and Sons, New York, 1968.
2. Hethcote, H.W., and Waltman, P. "Optimal Vaccination Schedules in a Deterministic Epidemic Model," *Math. Bios.* 18, 365 (1973)

S. Andrew Yakush received the A.B. degree from Andrews University in 1966, the A.M. degree from the University of Southern California in 1972 and the Ph.D. degree from the same institution in 1976.

Dr. Yakush joined the faculty of the Biomathematics Department in the School of Medicine of Loma Linda University in 1972 where he is an Assistant Professor and has taught graduate courses in Biological Control Systems and Biological Modeling. He has also participated in several research projects pertaining to the applications of computers and mathematics to the medical and biologcal world, and is now project manager of the Image Processing and Pattern Recognition facility.

Table 1.
General Instructions.

Step	Procedure	Enter	Press	Display
	$x' = f(t,x,y,z)$ $y' = g(t,x,y,z)$ $z' = h(t,x,y,z)$			
1	Enter program (A & B)			
2	Enter $f(R_{01},R_{02},R_{03},R_{04})$ into C $g(R_{01},R_{02},R_{03},R_{04})$ into D and $h(R_{01},R_{02},R_{03},R_{04})$ into E		GTO 163 LRN . . . LRN	
3	Enter half integration step	H/2	STO 98	H/2
4	Enter number of steps	N	STO 00	N
5	Enter initial conditions	t_o x_o y_o z_o	STO 01 STO 05 STO 06 STO 07	t_o x_o y_o z_o
				Printed
6	Start program		A	t_i x_i y_i z_i
	Available: 61 program steps (163-223) 9 data registers (11-19)			etc.

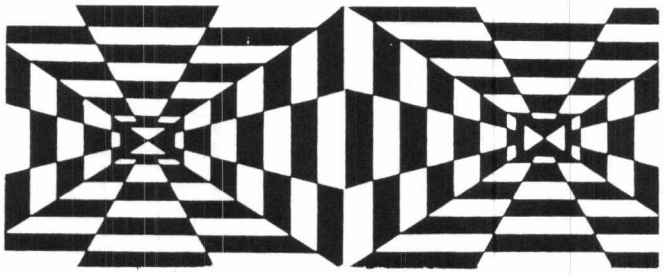


Table 2.
Program Listing.

LOC	CD	KEY	LOC	CD	KEY	LOC	CD	KEY	LOC	CD	KEY	LABELS
000	46	LBL	041	43	RCL	082	19	D'	123	09	9	A Start
001	19	D'	042	00	0	083	44	SUM	124	17	B'	B
002	55	÷	043	06	6	084	06	6	125	43	RCL	C f(t,x,y,z)
003	43	RCL	044	95	=	085	09	9	126	06	6	D g(t,x,y,z)
004	09	9	045	42	STO	086	56	RTN	127	07	7	E h(t,x,y,z)
005	09	9	046	00	0	087	46	LBL	128	44	SUM	A' used
006	95	=	047	03	3	088	11	A	129	00	0	B' used
007	56	RTN	048	43	RCL	089	25	CLR	130	05	5	C' used
008	46	LBL	049	01	1	090	42	STO	131	43	RCL	D' used
009	18	C'	050	00	0	091	00	0	132	06	6	E'
010	65	X	051	85	+	092	08	8	133	08	8	
011	43	RCL	052	43	RCL	093	42	STO	134	44	SUM	
012	09	9	053	00	0	094	00	0	135	00	0	REGISTERS
013	08	8	054	07	7	095	09	9	136	06	6	00 N
014	95	=	055	95	=	096	42	STO	137	43	RCL	01 t_i
015	56	RTN	056	42	STO	097	01	1	138	06	6	02 x_i+K/2
016	46	LBL	057	00	0	098	00	0	139	09	9	03 y_i+L/2
017	17	B'	058	04	4	099	03	3	140	44	SUM	04 z_i+M/2
018	43	RCL	059	13	C	100	42	STO	141	00	0	05 x_i
019	09	9	060	18	C'	101	09	9	142	07	7	06 y_i
020	08	8	061	42	STO	102	09	9	143	99	PAP	07 z_i
021	44	SUM	062	00	0	103	16	A'	144	43	RCL	08 K/2
022	00	0	063	08	8	104	02	2	145	00	0	09 L/2
023	01	1	064	19	D'	105	22	INV	146	01	1	10 M/2
024	46	LBL	065	44	SUM	106	49	PRD	147	98	PNT(HLT)	
025	16	A'	066	06	6	107	09	9	148	43	RCL	67 sum K/2
026	43	RCL	067	07	7	108	09	9	149	00	0	68 sum L/2
027	00	0	068	14	D	109	17	B'	150	05	5	69 sum M/2
028	08	8	069	18	C'	110	16	A'	151	98	PNT(HLT)	
029	85	+	070	42	STO	111	02	2	152	43	RCL	98 H/2
030	43	RCL	071	00	0	112	49	PRD	153	00	0	99 3 or 1.5
031	00	0	072	09	9	113	00	0	154	06	6	
032	05	5	073	19	D'	114	08	8	155	98	PNT(HLT)	
033	95	=	074	44	SUM	115	49	PRD	156	43	RCL	
034	42	STO	075	06	6	116	00	0	157	00	0	
035	00	0	076	08	8	117	09	9	158	07	7	
036	02	2	077	15	E	118	49	PRD	159	98	PNT(HLT)	
037	43	RCL	078	18	C'	119	01	1	160	58	DSZ	
038	00	0	079	42	STO	120	00	0	161	11	A	
039	09	9	080	01	1	121	49	PRD	162	81	HLT	
040	85	+	081	00	0	122	09	9				

Table 3.
Sample Program.

LOC	CD	KEY	LOC	CD	KEY	LOC	CD	KEY	LOC	CD	KEY
163	46	LBL	176	85	+	189	01	1	202	15	E
164	13	C	177	42	STO	190	04	4	203	43	RCL
165	43	RCL	178	01	1	191	75	-	204	01	1
166	01	1	179	04	4	192	43	RCL	205	02	2
167	01	1	180	43	RCL	193	01	1	206	65	X
168	65	X	181	01	1	194	02	2	207	43	RCL
169	43	RCL	182	03	3	195	65	X	208	00	0
170	00	0	183	95	=	196	43	RCL	209	03	3
171	02	2	184	94	+/-	197	00	0	210	95	=
172	65	X	185	56	RTN	198	03	3	211	56	RTN
173	43	RCL	186	46	LBL	199	95	=			
174	00	0	187	14	D	200	56	RTN			
175	03	3	188	43	RCL	201	46	LBL			

REGISTERS
11 c
12 r
13 v
14 cSI

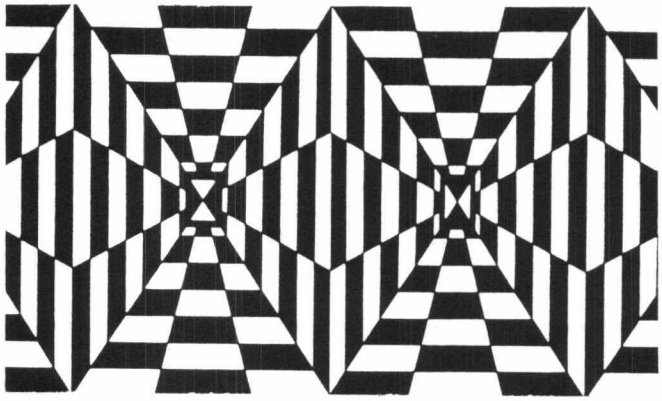


Table 4.
Typical Sample Run.

Step	Procedure	Enter	Press	Display
1	Enter program (A & B)			
2	Enter definitions of f, g and h (see Table 3.)		GTO 163 LRN . . . LRN	
3	Enter model parameters			
	Contact rate c	10	STO 11	10.
	Recovery rate r	6	STO 12	6.
	Vaccination rate v	.4	STO 13	0.4
4	Enter initial conditions			
	t_o	0	STO 01	0.
	S_o	.99	STO 05	.99
	I_o	.01	STO 06	.01
	R_o	0	STO 07	0.
5	Enter integration paramaters			
	H/2	.05	STO 98	.05
	N	10	STO 00	10.
				Printed
6	Begin integration		A	0.1
	S(.1)			.9383331529
	I(.1)			.0143969329
	R(.1)			.0072699143
				.
				.
				.
	t			0.5
	S(.5)			.6974981485
	I(.5)			0.034952619
	R(.5)			.0675492326
				.
				.
				.
	t			1.
	S(1.)			0.402482465
	I(1.)			.0263862156
	R(1.)			.1711313194
7	Change vaccination rate	0	STO 13	0.
8	Enter additional steps	5	STO 00	5.
9	Resume integration		A	1.1
	S(1.1)			.3929749397
	I(1.1)			.0215503779
	R(1.1)			.1854746824
				.
				.
				.
	t			1.5
	S(1.5)			.3709168282
	I(1.5)			.0089478071
	R(1.5)			.2201353647

Letters

IMPROVEMENTS ON CP/M AND CDOS

Dear Suzanne,

Thank you for the great, all CP/M issue of *DDJ* #41. I have a few comments to say regarding Jim Warner's article "CP/M and CDOS Program Compatibility." There are two small bugs involved with this program.

The first problem is that the function call 134 (FORMAT STRING TO FILE CONTROL BLOCK) builds the FCB correctly, but does not set the Next Record field to zero. This means that when the file is read in, CP/M will begin reading at the Record Number of the last record read of the previous file. This is normally only noticed under DEBUG when reading in multiple files. The fix for this is simple, and only adds 7 extra bytes:

```
NAM2:   CALL    DUMMY       ;THIS IS PART OF OLD CODE
        XOR     A
        LD      (DE),A      ;ZERO FILE EXTENT
;
; THIS IS THE ADDED CODE
;
        PUSH    HL          ;SAVE HL REG PAIR
        LD      HL,20       ;OFFSET FROM EXTENT TO NEXT REC.
        ADD     HL,DE       ;HL POINTS TO NEXT RECORD FIELD
        LD      (HL),A      ;ZERO NEXT RECORD
        POP     HL          ;RESTORE HL
; BACK TO ORIGINAL CODE
        POP     DE
        RET
```

The second, very small, problem concerns another function call. My version of EDIT (V00.10) does, as its first function call, a GET VERSION AND RELEASE NUMBERS (8DH). As far as I can tell, EDIT does not care what values come back as the version and release numbers. There are two possible solutions to this.

1. Patch EDIT to eliminate the call.

The offending code (for EDIT V00. 10) is at location 021CH. The original code is

```
021C    LD      C,8DH
021E    JP      5
```

If location 021CH is changed to a 'RET' instruction (C9) this eliminates the function call, and EDIT appears to operate correctly.

2. Change FILTER to ignore any undecoded function calls.

This may seem to be drastic, but I can report no problems with it in my experience. Change the instruction in FILTER

```
JP    NZ,IGOR   ;NOT DECODED, ** ERROR **
                      to
RET   NZ        ;NOT DECODED, IGNORE AND RETURN
```

Apart from these two small problems, I found Mr. Warren's article both enjoyable and very useful, and so I thank both him and you for it. Keep up the good work.

Sincerely,
Leigh F. Fiddes

25 Buena Vista Drive.
Montmorency. 3094.
AUSTRALIA.

MORE ON INDIRECT ADDRESSING FOR 8080/Z80

Dear *DDJ* :

With reference to "Indirect addressing for the 8080 and Z80" (*DDJ* #47), an alternative way to implement LHLI is

```
;  LHLI performs HL:=(HL)
LHLI:  PUSH PSW
       MOV  A,M
       INX  H
       MOV  H,M
       MOV  L,A
       POP  PSW
       RET
```

This is more efficient than the "miniprogram" technique described and avoids the gaucherie of self-modifying code.

Similarly,

```
;  IXMOV performs A:=(A+X)
IXMOV: PUSH D
       PUSH H
       PUSH X
       POP  H
       MVI  D,0
       MOV  E,A
       DAD  D
       MOV  A,M
       POP  H
       POP  D
       RET
```

Obviously, macro versions of this could be written to handle indexed moves to other registers. Also, the PUSH-POP pairs may be omitted when the application does not need the corresponding registers saved.

Sincerely,
David Ecklein

Tesseract Associates
Stinson Lake Road
Rumney, New Hampshire 03266

INTERFACING H-14 TO HORIZON

Dear Editor:

May I make the following suggestion in response to John R. Dye, (*DDJ* #47), regarding interfacing a Heathkit printer H-14 to a North Star Horizon.

Instead of trading away a very nice printer, why not make it run full speed? The North Star Horizon ports are extremely flexible and can interface to almost anything. You may wish to publish the following 3-wire configuration setup to assist others who wish to wed these two devices.

On the Motherboard header at location 4D, solder a wire from pin 7 to pin 8 and also a wire from pin 9 to pin 10. Then solder a wire from pin 12 of header 4D over to pin 12 of the header at location 2C. These 3 connections should be the only ones on these headers.

This allows the printer to run at full speed by having the printer's buffer full signal (interface pin 15) drop Clear to Send to the second serial port USART. No software changes are necessary.

Sincerely,
Warren B. Saunders

3417 Hickory Hills Drive
Oakton, Virginia 22124

. . . . . .

Dear Editor:

I am writing in reference to J. R. Dye's letter in *DDJ* #47. I agree with him that you should be aware of the complexity of integrating different manufacturers' hardware. Mainframe manufacturers' suggested configurations in many cases do not represent the best alternative to a user's needs.

There is no substitute for knowledge of your hardware's capabilities. The North Star Horizon has the needed hardware and software to perform the simple handshaking the H14 printer requires. I've successfully interfaced a Texas Instrument's 810 printer to the Horizon's serial ports.

This printer buffers incoming data until whole lines have been received. The printer then prints the buffer while holding pin 20 of the EIA interface low (DNB option in the 810 printer must be selected). The serial output routine must be able to detect this low signal and hold further output as the printer logic is BUSY printing the buffer.

The STANDARD serial output routine in the NORTH STAR DOS tests the least significant bit (TxRdy) of the serial port status byte. This bit is ON if the serial port is ready to receive a character from the CPU for output processing. This bit is affected by the CTS (clear to send) input to the 8251 USARTs that control the serial ports. The CTS input is terminated at pin 12 of the serial port config header.

In my application it was a simple matter to rewire the serial config header to jump (EIA pin 20/header pin 1) to (header pin 12/8251 input CTS). This turns OFF TxRdy bit in the serial port status byte when the busy printer is holding EIA pin 20 low. Note that a low on header pin 12 results in a high at the 8251 due to inverting EIA receiver.

Be aware that more is involved than just the above details in making the interface work. You must carefully scrutinize hardware documents in building the config. header and I/O cable. Your objective should be to get a low signal to pin 12 of the serial config. header signifying the printer is busy.

Pitfalls such as inverting EIA receivers/drivers, strapable hardware options and special I/O cables can make a maze out of the best documentation. I have personally experienced the excellence of North Star's hardware manual as well as the T.I. 810's. I would suggest you find a more knowledgable Horizon dealer, friendly or not.

Sincerely,
John T. McCarron

2021 N. 22nd Street
Phoenix, Arizona 85006

ADDENDUM TO "ZX65"

Dear Editor:

A potentially serious error in "ZX65: Simulating a Micro" (*DDJ* #47) has been pointed out to me by a user. It is system dependent and, fortunately, fairly rare. In those cases where the problem exists, however, ZX65 simply won't accept input from the console keyboard, and thus will not respond to commands once loaded.

The problem is that ZX65 expects the user's console print routine CONOUT to return with the character just printed still in the accumulator. In those few instances where this doesn't happen, the following patch will solve the problem:

ADDRESS	OBJ CODE	INSTRUCTION
7A28	C3 80 7D	JP 7D80
•		
•		
•		
7D80	F5	PUSH AF
7D81	CD 0C 7E	CALL CONOUT
7D84	F1	POP AF
7D85	C9	RET

This patch can be inserted by loading ZX65.COM under DDT, making the changes using the "S" function, and storing the modified code back to disk, following warm start, by typing "SAVE 15 ZX65.COM"

Respectfully,
Richard M. Kruse

6221 Woodlow
Wichita, KS 67220

WHEN MINIMUM VALUE EQUALS MAXIMUM COST

DDJ:

Most readers probably feel, as I do, that the goal of all human enterprise is to provide maximum value and service at minimum cost. This minimum can never be zero, since someone must pay the piper. I was therefore interested to learn that someone at Prentice-Hall did not agree that the numerical value of price is always positive. They sent me, earlier this year, no less than two announcements of a new book, *Microcomputer Interfacing*, by Bruce Artwick. Each had a postcard saying: "Yes, please send me a FREE 15-day examination copy. After I've seen (it) . . .I'll send you my check for $18.95 plus postage and handling. Otherwise, I'll return the book during the 15-day trial period and owe nothing." My normal tactic for new books is to wait until they show up in a bookstore, and to look them over very carefully before arriving at a decision (usually negative, since truly fine books are rare). So I ignored the first P-H message, but when the book failed to appear I sent in the second postcard. So that my tiny circle of admirers will not feel that I have lapsed into senility, let me add how I interpreted the word FREE as used by P-H. Since the retail markup of a book is 40-50%, I assumed that P-H was not really expecting to reap this for itself, but wished to provide extraordinarily early availability—a special service—while absorbing the extra costs (trivial handling charge, etc.).

The P-H interpretation of the word FREE is significantly different. I had sent the card in May. Nothing happened. In late June, the book appeared in a local bookstore. A very small book, technically OK, not worth the price ($20.18, including sales tax.) My examination of it was *really* FREE for me, though of course the principle that nothing is free

was not violated—the overhead costs to the store are ultimately paid, by me and others, as part of the price of the books we buy. That's fairness.

Yesterday, the P-H computer acknowledged my card of last May by sending me a bill for $23.09, itemizing the base price plus $2.73 postage and handling, plus $1.41 sales tax. This tax is 18 cents higher than the local tax, since the computer is programmed to tax the postage-and-handling too. The computer also knows that the Berkeley tax is 6.5%, not 6% as in most of the state. It doesn't know that the card it was answering forgot to mention a tax, or that there is as yet no book. It is probably now counting down from 15 days, when it will emit a second bill. As to the book itself, the mills of the gods (notoriously slow) may even now be grinding away at this trivial problem. For the scam to be truly perfect, it ought to appear on day −1 of the countdown! The FREE offer would then become an unconditional sale, delivered 4+ weeks later than I could have got it in Berkeley and at a price $2.91 higher. This idiotic rigmarole perhaps makes sense for a potential buyer in the middle of the Sahara desert. The P-H computer, so deeply versed in the minutiae of taxation, is unaware that Berkeley is awash with books.

Fairness compels me to acknowledge that the P-H organization is not made up 100% of nitwits (except perhaps at its selling end). Last May I had a phone call from the Bay Area P-H representative, Howard Welt, an intelligent and courteous man. He had seen my articles on Instruction Sets in *Kilobaud Microcomputing*, and wondered whether I had enough material for a book. I said no, explained why not, and suggested names of people who might be interested. Later, I wrote him a letter with supplementary information. No "handling charge" (P-H) since I'm a true believer in the dissemination of information at minimum cost. That's the major role of a university. I don't expect book publishers to operate at a loss. Still, they also are disseminators of information. I do expect a high level of integrity and efficiency. The cost of the Artwick book could have been reduced by using paperback instead of hardcover. Since such books begin to obsolesce before the ink is dry, softcover makes better sense.

Sincerely,
Hal Gordon

College of Natural Resources
University of California
Berkeley, CA 94720

TOO MUCH Z80, 8080 AND CP/M

Dear *Dr. Dobb's*:

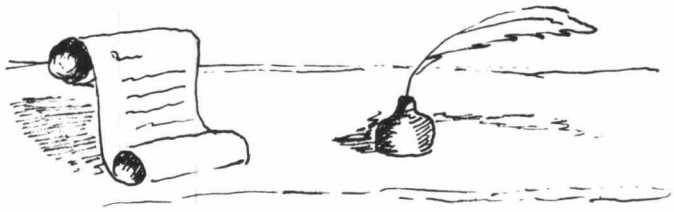
Thank you for the issue on the Language C (*DDJ* #45). It is nice to see some recognition for alternatives to PASCAL.

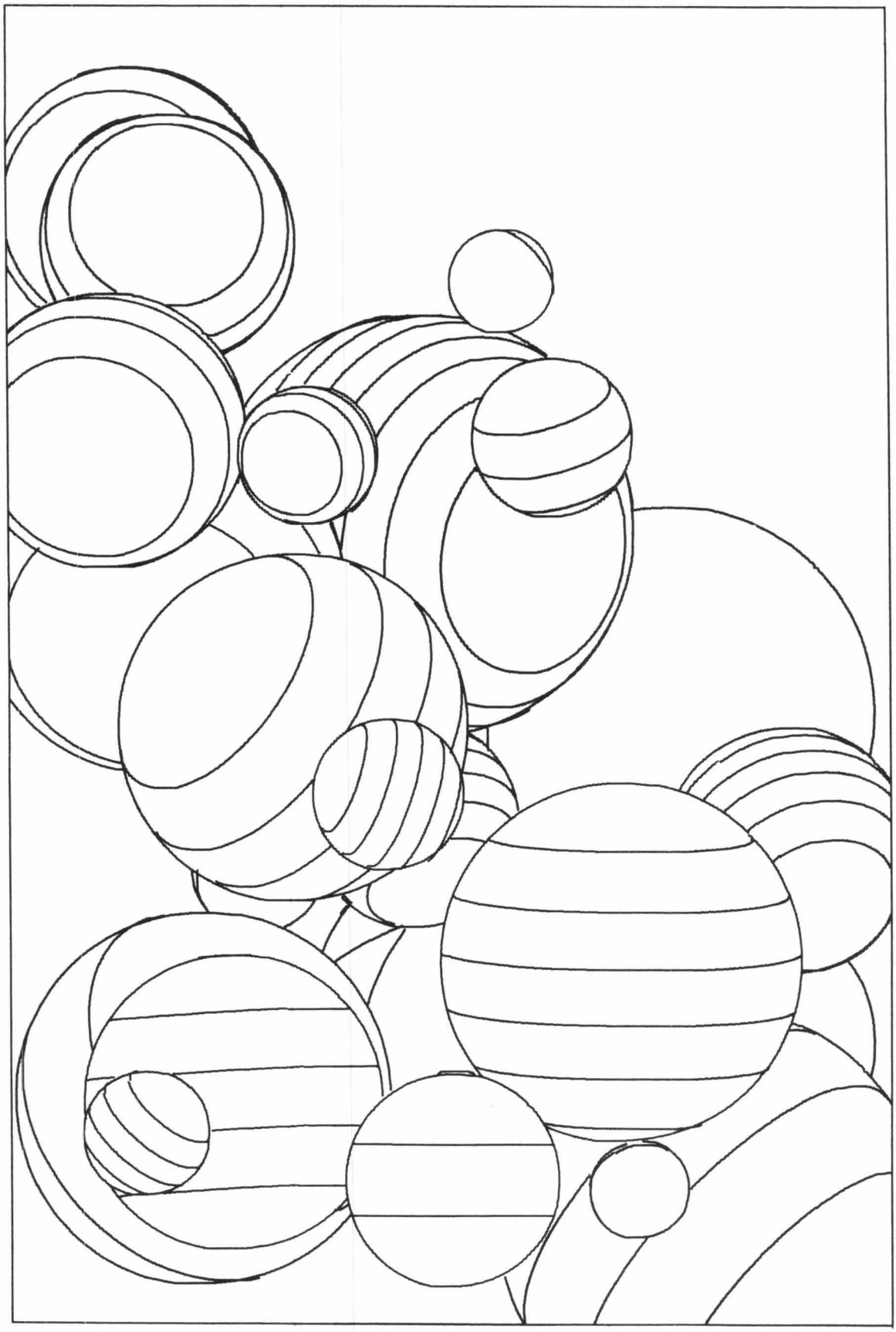
Tim Pugh's article in *DDJ* #46 on MCALL-C, while interesting, contains an instance of the much-too-prevalent attitude that everybody who has a disk system is using it with a Z-80 or 8080 and is running CP/M. For a column and a half, the article, which is about a communications protocol for sending files from one personal computer to another, could apply to any computer with a modem and a disk. Then he answers the question of why it takes an indeterminate amount of time to open a file by saying that it's because of the wide variety of disks and controllers that can be used with CP/M! I was expecting something more like: it's because of the wide variety of disks, controllers, operating systems, and processors that are in use. Instead, we are apparently intended to assume that the only disk operating system (DOS) that exists is CP/M (and by implication, that the only microprocessors that exist are the 8080 and Z-80). To be fair to Mr. Pugh, he has developed the protocol on a CP/M system and is apparently selling this CP/M implementation of it. However, there is nothing about the protocol that requires CP/M. It could be implemented for any DOS on any processor.

I would like to clarify the remarks I made in my letter that appeared in *DDJ* #46 (page 55). My statement that CP/M will not be "universal" until there are versions of it for many other processors (6800, 6502, etc.) was not intended to imply that I advocate such implementations. CP/M has a number of limitations and shortcomings that should not be perpetuated in other operating systems. The biggest such problem is probably the lack of device-independent I/O, as well as the fixed and rather limited number of devices supported. CP/M keeps track of the size of disk files in sectors rather than in bytes. Thus, a special data character (control-Z) is required to mark the end of a text file, and files containing "binary" information must be a multiple of 128 bytes in length. These features, resulting to some degree in two file types or formats, are at best unattractive. The orientation toward disks with 128-byte sectors, toward terminals with upper case only, its encouragement of assembly language for almost all programming; the restriction, removed in the latest release, to disks of 256K bytes or less; the rather clumsy method of random access to disk files, improved in the latest release; and the fact that processing and I/O cannot be overlapped are also problem areas. Perhaps it is because of some of CP/M's limitations that it can run on as many different systems as it does.

Sincerely,
Jim Howell

5472 Playa Del Rey
San Jose, CA 95123





© David Dameron 1979

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 50 | November/December 1980 | Volume 5, Issue 10

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

N-Logs: A New Number Language for Scientific Computers **4**
Keneth A. Simons

Las Vegas Super Slot: A CP/M Game Machine Program Using Flashwriter I Graphics **10**
David W. Freidman

Quick-Key: A North Star BASIC Statement Generator **23**
Leonard E. Garcia

TUTOR: A Foreign Language Vocabulary Trainer **29**
Gary D. Gaugler

Interfacing the WH14 Printer to Your Microcomputer's Serial Port Is Simpler Than You Think **30**
Lachman Sippy

and more ...

N-LOGS

A NEW NUMBER LANGUAGE FOR SCIENTIFIC COMPUTERS

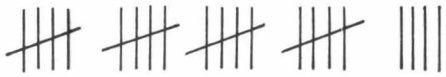
by Keneth A. Simons

SUMMARY: The N-log is so defined that it is a simple positive integer used to represent any number, with any required precision. No suffixes, prefixes, exponents or decimal points are needed to allow symbolizing the entire range of numbers encountered in scientific computations. N-logs are particularly useful with digital computers, eliminating the need for floating point routines and exponential notation. A procedure is described that allows finding the N-log of the sum or difference of two numbers, when the N-log of each is known, without converting from N-logs to numbers or vice-versa.

INTRODUCTION: Since men first started numbering and measuring, the results have been written symbolically. There is, for example, not the remotest relationship beween the symbol '2', or the word 'two', and a pair of things as they exist in nature. We have all become accustomed to the fact that one represents the other, but the whole idea is strictly man-made. Since this is true, it is perfectly allowable to look for other, and possibly more convenient, ways of representing numbers. A good example is found in the hexadecimal system where, because it is convenient in the computer context, we use, for example, the symbol A to represent the number that has been symbolized by 10 for at least a thousand years! This paper describes a new way of representing numbers which is based on simple fundamental principles and has many advantages over conventional methods.

THE DEVELOPMENT OF SYMBOLS FOR NUMBERS: When a prehistoric farmer asked his slave how many hogs were in a field, the man might have held up nine fingers, or made nine scratches in the dirt, like this:

This system was direct, but became cumbersome for the wealthy with lots of pigs! So a better system evolved wherein one counted four pigs, then scratched diagonally across to show the fifth. With these symbols twenty-four pigs would appear as:



One simply counted up how many fives there were, added the odd ones and arrived at the number.

ROMAN NUMERALS: As history progressed it became necessary to deal with larger numbers, and a simpler system was needed. The Romans came up with a scheme where letter symbols were used for numbers. Thus I represented one, V was used for five, X for ten, L for fifty, C for a hundred, and so on. Any number could be portrayed by writing the appropriate symbols in a row, with the smaller ones on the right. The number represented was understood to be the sum of the numbers indicated by the symbols. Thus CCLXI meant two hundreds (CC) plus a fifty (L) plus a ten (X) plus one (I), or two hundred and sixty-one. Unfortunately, at about this time, mankind became interested in calculation. With the rise of commerce and the early beginnings of the sciences it became necessary not only to be able to write numbers but also to use them in calculations. The processes of multiplication,

Keneth A. Simons – 2035 Willowbrook Drive, Huntingdon Valley, PA 19006

division and the expression of fractions were extremely clumsy using the Roman system. As Dr. Jacob Shekel once said, referring to the failure of the Romans to recognize the usefulness of a symbol for zero, "Their trouble with numbers was that they didn't know nothing."

ARABIC NUMERALS: Fortunately a system had been in use in the Arabian countries for centuries based on three revolutionary and incredibly useful concepts. This system was to be adopted by the Western world, and it played an important part in making possible the rapid development of the physical sciences during and after the Renaissance. These concepts were:

1. Nothing was given a name (zero) and a symbol (0).
2. The next nine numbers were given both names and symbols with the English names: one (1), two (2), three (3), four (4), five (5), six (6), seven (7), eight (8), and nine (9).
3. The weight of a given digit was indicated by its position. Thus the right-most digit was assumed to be multiplied by 1, the next digit to the left by 10, the next by 100, and so on.

Using the Arabian system the symbol 9870 would be interpreted as (9 X 1000) + (8 X 100) + (7 X 10) + (0 X 1). This way of symbolizing numbers greatly simplified the algorithms for the needed mathematical calculations so that, compared with earlier systems, these calculations were accomplished much more rapidly.

FRACTIONS: With the acceptance of Arabic numerals, an additional need developed – the need to represent numbers less than one. Unfortunately a solution arose which originated centuries of confusion from which we still suffer: the fraction was introduced! Numbers less than one were represented as the quotient of one integer divided by another. These symbols were clumsy to write, and, as any grade school graduate knows, calculations involving them were complex and cumbersome. To make matters worse the British embraced the idea and saddled us with a system of measurement that is incredibly inefficient. They did, in retrospect, the strangest things, avoiding the number 10, the base of their number system, as if it were a plague! A mile was divided by 1760 (where did that come from?) to get the yard. The yard was divided by 3 to get the foot (why 3 rather than 5 or 10?) The foot was split into 12 parts (they couldn't use 10, it might simplify things) to get the inch. Then they ran out of names, so smaller sizes were expressed as fractional parts of an inch, right on down to 1/64s. The whole mess has become part of our culture, and probably has wasted more man-hours of intellectual effort than all of the World Wars combined!

DECIMAL FRACTIONS: A much simpler way of representing numbers less than one followed the invention of the decimal point, and the discovery that a self-consistent system resulted when the first digit to the right of that point was assumed to be divided by 10, the second by 100, the third by 1000, and so on. With this approach the symbol .235 represented the sum of 2/10 + 3/100 + 5/1000. The great advantage of presenting numbers this way lay in the fact that all of the calculating algorithms that worked with integers were equally useful with numbers containing decimal fractions, as long as care was used to place the decimal point in the right position in the result. They had the further advantage that they were easier to write since there were no numbers above or below the line. Decimal fractions are almost universally used in today's pocket calculators, which generally don't display ordinary fractions It seems possible that, between these calculators and the almost world-wide adoption of the metric system, the world may (except for some back-water areas like our own country!) at long last get rid of the English system and the ordinary fraction.

EXPONENTS AND EXPONENTIAL NUMBERS: With the rapid development of astronomy, physics and the other natural sciences in the 16th and 17th centuries, came the need for improved ways of handling very large and very small numbers. Astronomical measurements showed, for example, that the distances to even the nearer stars were so great that they could conveniently be expressed by using a 'light year,' the distance travelled by light in one year. One light year is about 6000000000000 miles. In physics it was found that the rest mass of an electron is something like 0.000000000000000 0000000000000001 kilograms. It was obvious that it was impractical to express numbers like this with ordinary numerical notation. A more compact and convenient numerica language was needed.

It had been customary for some time to express the number of times a number is multiplied by itself by the use of an 'exponent,' a little numeral written above and to the right. Thus 2 times itself, called "two squared," was written 2^2. 2 X 2 X 2, called 2 cubed, was written 2^3, and so on. The use of the number 10 with various exponents turned out to be a very effective answer to the problem stated above. Since 10^{12} = 1 followed by 12 zeroes, an awkward numeral like 6000000000000 could be broken up into the product of 6 by 1000000000000, written simply as 6×10^{12}. This technique provided a much simpler and more satisfactory way of dealing with very large numbers. To deal with small numbers the fact was used that a number with a negative exponent is equal to the inverse of the same number with the same exponent having a positive sign. Thus $10^{-4} = 1/10^4 = 0.0001$. The rest mass of the electron, written above with 30 zeroes followed by a 1, could be expressed much more compactly as 1×10^{-31} kilograms. Numbers expressed in this way, i.e., using an exponent, are called "exponential numbers" and this technique for handling very large or very small numbers is called "scientific notation." It is used almost universally in scientific work where large or small numbers are involved, and is available in most calculators and computer languages intended for use by those concerned with scientific calculations.

LOGARITHMS: Another important numerical language that is closely related to the exponent is the logarithm. The basis for logarithms is the fact that any number can be raised not only to a power which is an integer (e.g. 8^6=262144 or 3^4=81), but also to a power that includes a decimal fraction (e.g. $2^{1.89}$ = approximately 3.706, $10^{1.1}$ = approximately 12.59). The logarithm of any number is simply the power to which some other number (called the "base") must be raised to equal it. Thus, referring to the examples above, the logarithm of 262144 to the base 8 is seen to be 6 (since this is the power to which 8 must be raised to equal 262144); and the logarithm of 12.59 to the base 10 must equal 1.1 for the same reason.

Logarithms were originally discovered by Napier, who used the number "e" (=2.71828. . .) as his base. His reason for this choice was that the logarithm of a number to this base (called its "natural" logarithm) is relatively easy to compute. This is also the reason why many computer languages give the natural logarithm. Briggs later discovered that, since our numerical language, the decimal system, is based on the number 10, 10 also makes a very convenient base for logarithms. He, and others who followed him, compiled tables of these logarithms which formed, for almost 250 years, the basis for almost all accurate computations in the sciences and in surveying.

The advent of the electronic computer in the middle of the 20th century changed all that. Its speed and facility are so much superior to that achieved with logarithms that the use of

the latter for computation has practically vanished. In the 1967 edition of the *Encyclopedia Brittanica* (Ref. 1) K. O. May states that ". . . logarithms (are) virtually obsolete for large-scale computation, though they are used in the 1960's for a few special purposes and indirectly as the basis of the slide rule." (In 1980 even this use has disappeared, although the logarithm still remains as the basis for the decibel, widely used in communications and acoustics.)

It is a strange quirk of history that today, when the computation of a logarithm requires only the operation of an inexpensive pocket calculator, the need for them has almost completely disappeared! Almost all the computations that formerly required logarithms are now done by these same calculators, or their big brothers, the electronic computers. The object of this paper is to propose a new idea, inspired by the almost universal acceptance of the decibel. What is proposed is the use of the logarithm as the basis for a new numerical language, called "N-logs" which, being integers, are more convenient for people to handle, and especially suited for use as input and output numbers for computers.

PRECISION AND ACCURACY: Since one of the major advantages of N-logs has to do with their precision, it is important to establish clearly what that word means. *The International Dictionary of Applied Mathematics* (Ref. 2) defines "accuracy":

Accuracy*: The degree of exactness actually possessed by an approximation, measurement, etc. It may be contrasted with* precision *which is the degree of exactness with which a quantity is expressed. For example, as a value for* π *the number 3.1428 is more precise than accurate.*

This is, of course, because that number is in error by about .04% (as compared with the correct value, 3.1416), whereas it is stated with a precision of plus or minus 0.0016% (the maximum range of numbers implied by the digits 3.1428). As another example, suppose that a digital voltmeter reads 8.19271 when measuring a certain voltage, but it had been calibrated against a voltage standard which had an error of 0.1%. The accuracy of this measurement, limited by the standard, would be only 0.1%, while the precision of the reading would be about 1 part per million, 1000 times better.

THE PRECISION OF RATIONAL NUMBERS: All of the numbers that report the results of measurement in engineering and the sciences are "rational numbers" since they must be written with a limited number of digits, and, for the same reason almost all of them represent approximations (the single exception being integers that result from counting a few items, which are the only numbers we deal with that can be exact). Because of this it is important to ask what factors affect the precision of numbers as they are usually written. It is common to speak of "three figure accuracy," implying that the precision is affected only by the number of digits. Unfortunately this is not true. The precision with which a number is written depends not only on the number of digits that are used, but on what those digits are. As an example the number 9821, as an approximation, represents the result of rounding any number between the limits 9820.5 and 9821.5. With four significant figures, the number has a precision of 51 PPM (the error of 9821.5 compared with 9821). The number 1032, on the other hand, with the same number of digits, is written with a precision of only 484 PPM (the error of 1032.5 compared with 1032), almost 10 times poorer than 9821! This problem of variable precision is inherent in our number system, and is a result of the fact that it was developed for counting, rather than for measuring ratios, so that each successive number in a series, as we write it, is arrived at by adding a fixed small number to the previous one.

THE PRECISION OF LOGARITHMS: By contrast consider the LOGs (see NOTE, p. 36) of the two numbers written above. The LOG of 9821, written with three digits to the right of the decimal point, is 3.992. Since this number approximates all LOGs between 3.9915 and 3.9925, it represents all numbers between 9806 (EXP 3.9915) and 9829 (EXP 3.9925) and it represents these numbers with a precision of about 0.12%. Compare this with the LOG of 1032, which is 3.014. This represents the spread of numbers between about 1032 (EXP 3.0135) and 1034 (EXP 3.0145), also showing a precision of about 0.12%. This illustrates a fact that is basic to this presentation: the precision with which a LOG represents a number is determined only by the number of digits in the LOG to the right of the decimal point, and is not affected by what those digits are. It is for this reason that logs are called "the language of ratios," and make an excellent numerical language for reporting ratio measurements.

PEOPLE PREFER INTEGERS: Most people dislike fractions (including the decimal kind) and avoid using them whenever possible. A person who goes into a lumber yard asks for a "2 by 4," although the actual dimensions of the board he wants are close to 1-3/4 by 3-3/4 inches. He asks for "Number 22" wire at the radio store, when the actual diameter of the wire he wants is around 0.025 inches. Although he actually pays 0.17 times the principle on his mortgage, he calls this "17%." The whole complicated series of prefixes for metric measurements is chiefly aimed at keeping the numbers used as close to integers as possible. We'd much rather say 83 millimeters than "0.083 meters;" 102 microhenries is less of a mouthful than 0.000102 henries. One rather silly place where this breaks down is in the measurement of capacitors. Through long-standing custom we almost always say ".005 microfarads" rather than "5 nanofarads," which is just as correct and much easier to state.

THE DECIBEL: One of the cleverest dodges for turning complicated numbers into integers was invented at Bell Laboratories in 1924. They multiplied the LOG of any power ratio by 10 and obtained a beautifully simple series of integers to describe the quantities which most concerned them: gain, loss and power levels in telephone systems. Unfortunately they made two mistakes that severely limited the spread of their idea. In the first place they insisted that the process could only be used with power and power ratios (which limitation has no basis in fact since, obviously, the LOG of any number can be taken, and the result multiplied by 10, whether the number represents power, resistance, length or the population of Malaysia!) (see Ref. 3) The only sensible question is "Does this process yield a convenient result?" If it does, it should be used. The other mistake the Bell people made was to adopt a very confusing terminology. They said, for example, that the attenuation of a section of telephone line was measured and expressed in terms of a unit, the decibel. They considered attenuation as a measurable Physical Quantity (which it is, being a measurable property of a Physical System), but they also considered the dB as the Physical Unit in terms of which this quantity was measured. One dB is no more a Physical Unit than the number 3! The statement "The ratio is one decibel" means nothing more nor less than "10 times the LOG of the ratio = 1," i.e., the LOG of the ratio is 0.1 and, since EXP 0.1 = 1.259, the ratio is approximately 1.259. The number 1.259 can, by no stretch of the imagination, be called a unit of anything!

N-LOGS—EXPRESSING ANY NUMBER AS AN INTEGER: Since it is clear that people prefer integers, it makes good sense to look for a number language that allows expression of any number in that form. The decibel approach points the way, 10 times the LOG of any number can be rounded to

form an integer but when this is done the precision of the expression is restricted. This can be seen if we consider the range of numbers represented by, for example, 3 decibels. The corresponding nominal ratio is found from 10 X LOG X = 3, LOG X = .3 and X = EXP .3, i.e., X = 1.995. At the upper limit 10 X LOG Y = 3.5, and Y = EXP .35 = 2.239. Since 2.239 is 12% higher than 1.995, and LOGs with the same number of significant figures to the right of the decimal point represent the corresponding number with the same precision, it follows that dBs, rounded to integers, have a basic precision of plus or minus 12%. Since this precision is inadequate for many applications, the next question is "How can the precision be improved without losing the important advantage of the integer presentation?" The answer is very simple. As with any other decimal number, multiplying a LOG by 10 has the effect of moving the decimal point one place to the right. Thus 3 dB is simply the LOG .3, with the decimal point shifted. To move that point two places would require multiplication by 100, three places would require 1000, and in general, to move the decimal point in a LOG "N" places requires multiplying the LOG by EXP N, where N is 1, 2, 3, 4, 5, 6 and so on, as required. From this follows the definition of N-logs:

> *The N-log of a number is that integer most closely approximating the product of EXP N and the LOG of the number.*

It should be noted that N is not necessarily an integer. In fact the multiplier, defined above as 10^N, can be any number. The implications of this fact are very broad, showing that one has a wide choice of multipliers so that the use of the N-log concept is not restricted to the decimal system, but can be used with any number language. However our calculators and our brains are both more at home with the 10-based system, so the general case will not be explored further in this paper.

Since N-logs as defined above, where N is a small integer, are easily found with a pocket calculator, it makes sense to use them. An understanding of the characteristics of each "family" (i.e., the N-logs resulting from each choice of N) is essential if the idea is to be applied to practical situations. Table I shows how the choice of N determines the number of digits in the N-log, and the precision with which it can express a number. It can be seen that these families provide a very wide range of capabilities.

TABLE I

The Properties of the First Seven Families of N-logs

Family	N	Multiplier	# of Digits*	Precision
1-Logs	1	10	3	+ 12 %
2-Logs	2	100	4	+ 1.2 %
3-Logs	3	1,000	5	+ 0.11 %
4-Logs	4	10,000	6	+ 0.011 %
5-Logs	5	100,000	7	+ 11 PPM
6-Logs	6	1,000,000	8	+ 1.1 PPM
7-Logs	7	10,000,000	9	+ 0.11 PPM

* The number of digits is that required to represent numbers between 10^{-99} and 10^{+99}.

N-LOGS SIMPLIFY EXPRESSIONS: The simplicity achieved by the use of N-logs can best be illustrated by a few examples: The mean distance from earth to the sun is 9.3×10^7 miles. The Two-log of this distance in miles is 797, expressed with a precision of ±1.2%. Instead of reporting the result of measuring a certain capacitor as "986 picofarads" it could be expressed by stating the Three-log of the capacitance in farads, which is -9006, with a precision of ±0.11%. One light year, stated as the Six-log of the number of miles, is 12768973, expressed with a precision of about 1 PPM. The rest mass of the electron, in kilograms, is expressed, including the accuracy with which it is known, by the Six-log -30040-502±6. Since a change of 1 in the right-hand digit of a Six-log corresponds to about 1PPM (regardless of what the digits are) this states, without the use of any additional symbolism (like PPM or %), that this quantity is known with an accuracy corresponding to ±6 PPM. This simplicity in the statement of tolerance is one of the several advantages of N-logs.

AN N-LOG IS SIMPLY A LOGARITHM TO A DIFFERENT BASE: The bases that have been used for systems of logarithms in the past have been a small group. They include 10 as the base for Common Logarithms; the number "e" (2.71828 . . .) as the base for Natural Logarithms; and more recently 2 as the base for logarithms relating to computer operation. This paper is an attempt to show that many other bases are not only possible, but sometimes very convenient. To show that an N-log, as defined above, is simply a logarithm to a base which is different from 10, e or 2, some new terms are needed. In what follows the word "log" will be used to mean a logarithm in general, i.e., one taken to any base. With a subscript ("$\log_x$") it will indicate a logarithm to the base indicated by the subscript, e.g., $\log_2$ will mean the logarithm to the base 2.

The formula for changing the base of a logarithm can be found in any elementary Algebra text. One way to state it is:

$$\log_x a = \log_y a \,/\, \log_y x$$

In this equation x and y are constants (the two bases) and a is a variable, thus dividing $\log_y a$ by $\log_y x$ is the same as multiplying it by the constant $1 \,/\, \log_y x$. So every N-log, which is found by multiplying the $\log_{10}$ of a number by a constant, is simply a log, and all the rules that apply to logs apply equally to N-logs, including the four basic rules:

$$\log(XY) = \log(X) + \log(Y)$$
$$\log(X/Y) = \log(X) - \log(Y)$$
$$\log(X^Y) = Y \times \log(X)$$
$$\log(\sqrt[Y]{X}) = \log(X) \,/\, Y$$

FINDING LOG(X+Y) WHEN LOG(X) AND LOG(Y) ARE KNOWN: To allow all the numbers in a computer program to be represented by N-logs, and to avoid the need for changing N-logs to ordinary numbers, or the converse, a procedure is needed that allows finding the N-log of the sum or difference of two numbers directly when the N-log of each one is known. Although it is not found in many modern textbooks, such a procedure exists. It was invented in 1803 by an Italian physicist named Leonelli, but was popularized by Gauss, and called "Gaussian Logarithms" (see Ref. 1). Actually the logic involved is quite simple, and the computer operations that are required are reasonably fast and easy to implement. The mathematical development of the formulas for accomplishing this is shown in Appendix I, and the results for one family, the Two-logs, are shown in Tables II, III and IV. Similar tables for Three-logs would contain 300 lines each, 3000 for Four-logs and so on.

TABLE II

Sum Factor for Two-logs

When A-B is between:	S equals:
0 - 1	30
2 - 3	29
4 - 5	28
6 - 7	27
8 - 9	26
10 - 12	25
13 - 14	24
15 - 16	23
17 - 19	22
20 - 21	21
22 - 24	20
25 - 27	19
28 - 30	18
31 - 33	17
34 - 36	16
37 - 40	15
41 - 43	14
44 - 47	13
48 - 51	12
52 - 56	11
57 - 61	10
62 - 66	9
67 - 72	8
73 - 79	7
80 - 86	6
87 - 96	5
97 - 107	4
108 - 122	3
123 - 145	2
146 - 193	1
Above 194	0

TABLE III

Difference Factor for Two-logs. A-B < 30

When A-B equals:	D equals:
0	infinity
1	164
2	135
3	118
4	106
5	96
6	89
7	83
8	77
9	73
10	69
11	65
12	62
13	59
14	56
15	53
16	51
17	49
18	47
19	45
20	43
21	42
22	40
23	39
24	37
25	36
26	35
27	33
28	32
29	31
30	30

TABLE IV

Difference Factor for Two-logs. A-B > 30

When A-B is between:	D equals:
30 - 30	30
31 - 31	29
32 - 32	28
33 - 34	27
35 - 35	26
36 - 36	25
37 - 37	24
38 - 39	23
40 - 40	22
41 - 42	21
43 - 44	20
45 - 45	19
46 - 47	18
48 - 50	17
51 - 52	16
53 - 54	15
55 - 57	14
58 - 60	13
61 - 63	12
64 - 66	11
67 - 70	10
71 - 75	9
76 - 79	8
80 - 85	7
86 - 92	6
93 - 100	5
101 - 111	4
112 - 125	3
126 - 146	2
147 - 194	1
Above 194	0

USING THESE TABLES: A "Sum Factor," "S" and a "Difference Factor," "D" are defined by the equations:

A = Two-log(X)
B = Two-log(Y) where A > B.
S = Two-log(X+Y) - A
D = A - Two-log(X-Y)

With these definitions the Two-log of X+Y = A+S and the Two-log of X-Y = A-S. To find the Two-log of X+Y, subtract B from A, find the value of S in Table II which corresponds to this difference, and add it to A. Similarly, to find the Two-log of X-Y, subtract B from A, find the corresponding value of D in Table III (if A-B < 30) or Table IV (if A-B > 30), and subtract D from A.

Examples: Some examples will show how this works out. Suppose $X = 1.479 \times 10^{12}$ and $Y = 0.602 \times 10^{12}$ then A = 1217, B = 1178, and A-B = 39. Since 39 is between 37 and 40, Table II shows that, in this case, S = 15, so Two-log(X+Y) = A+S = 1217 + 15 = 1232. Since A-B > 30, D is found in Table IV, and equals 23. Thus Two-log(X-Y) = A-D = 1217 - 23 = 1194. As a check add X and Y: $1.479 \times 10^{12} + 0.602 \times 10^{12} = 2.081 \times 10^{12}$, which has the log 12.32 and the Two-log 1232. Similarly $X-Y = 0.877 \times 10^{12}$ with the log 11.94, and the Two-log 1194, which results agree with those obtained using the tables.

As a second example let A = -328 and B = -354 (-328 > -354), A-B = 26. From Table II: D = 19, so Two-log(X+Y) = -328 + 19 = -309. From Table III: D = 35, so Two-log(X-Y) = -363. Checking this: X= 0.0005248, Y = 0.0002884, X+Y = 0.0008132 with the Two-log -309, and X-Y = 0.0002364 with the Two-log -363.

These examples clearly show the disadvantage of N-logs, i.e., that addition and subtraction are more difficult when using them, so that they would not be useful in contexts, as in accounting, where these operations predominate. However,

in most scientific calculations their advantages outweigh this disadvantage. 1217 is easier to write and to pronounce than is 1.479×10^{12}. To find the Two-log corresponding to $1.279 \times 10^{12} \times 0.602 \times 10^{12}$, 1217 is added to 1178, giving 2395. The Two-log of 0.0008132^{8} is $8 \times -309 = -2472$ and so on.

CAN LOGARITHMS BE USED TO REPRESENT NEGATIVE NUMBERS?: When it is proposed that N-logs be used to represent all of the numbers in a series of calculations, an objection that will immediately be raised by anyone familiar with logarithms is the fact that, since there is no power to which a base can be raised that gives a negative result, it is impossible to take the logarithm of a negative number. While this is certainly true it does not interfere with the use of logarithms to *represent* negative numbers. Consider several comparable situations. The simplest relates to the fact that decimal digits *alone* cannot represent a negative number. Any group of symbols including only 012345678 and 9 will be interpreted as a positive integer! Mathematical tradition, to get around this, provides for the representation of negative numbers by the use of an eleventh symbol, the minus sign. What's sauce for the goose is sauce for the gander! If extra symbols are allowed, they can be used with logarithms just as well as with ordinary numbers. Since the minus sign is already used to show whether the logarithm is positive or negative, some other symbol, for example the letter N, must be chosen to indicate that the number represented by the logarithm is negative. Thus, since the Three-log of 5678 is written 3754, the symbol for the Three-log of −5678 could be written 3754N. The Three-log of −0.3492 could be written −2457N, and so on. Of course the N would have to be taken into account when using the S or D routines described in the preceding section.

Another situation, very familiar to computer experts, relates to the fact that any combination of the binary digits 0 and 1, taken as written, must be interpreted as a positive integer. To get around this it is common practice in designing and programming binary computers to represent a negative number as the two's complement of the corresponding positive number. Thus the decimal number +55 would be represented in binary by the symbol 110111, while −55 would be 001001. This way of symbolizing negative numbers has two advantages. Not only does it change them into a form the computer can handle, it provides the added advantage that, within the computer, the addition of the binary number representing a positive quantity to that representing a negative one will give a result which is the difference of the two quantities (if the overflow is neglected). This basic idea can be applied to N-logs, as will be demonstrated in the next section.

N-LOGS FOR COMPUTERS: While there are many possible ways in which a number language for computers could be built around the N-log concept, one particular way of doing this seems like a good one. It begins by assuming a 16-bit binary word. This means that the largest number that can be represented is FFFF (Hex). Thus, including 0 there are 10000 (Hex), or 65536 (Decimal) possible numbers. There are four possible kinds of N-logs to be represented; i.e., the + N-log (i.e., positive N-log) of a + number; the − N-log (i.e., negative N-log) of a + number; the + N-log of a − number; and the − N-log of a − number. This means that the total number of symbols allowable must be divided by four to find the largest N-log that can be represented. 65536 divided by 4 is 16384 (Dec.) (= 4000 Hex)). If Three-logs were used this means that the range of numbers that could be handled would be the ones whose Three-logs extended from −16384 to +16384, i.e., the numbers between 4.130×10^{-17} and 2.4210×10^{-16}. The precision of all numbers would be that characteristic of Three-logs, i.e., plus or minus 0.115%. This range of numbers and this degree of precision is adequate for most scientific calculations, so this approach seems to have practical appeal.

A SPECIFIC PROPOSAL: One way to implement this idea would be to select the lowest group available in a 16-bit word (i.e., the numbers from 0000 to 3FFF (Hex)) to represent the positive Three-logs of positive numbers. The two's complements of these (i.e., the numbers from FFFF down to C001 (Hex)) would then logically represent negative Three-logs of positive numbers. Adding 4000 (Hex) to the lowest group would set aside the numbers between 4000 and 7FFF (Hex) to represent positive Three-logs of negative numbers, and their two's complements, the numbers from C000 down to 8001 (Hex), to represent the negative Three-logs of negative numbers. Such an approach would provide a simple way of arriving at the initial Three-log, by the use of a scientific calculator, simple algorithms within the computer for converting these input numbers to the corresponding binary numbers, and extreme simplification of the operations of multiplying, dividing, powers and roots by the computer. Operations corresponding to the addition or subtraction of the original numbers, or the intermediate results, would be a little more complicated, requiring look-up tables for S and D, but would be perfectly feasible and quite fast.

CONCLUSION—A PERSONAL NOTE: It must be obvious to anyone who reads through this paper that the missing step is the creation of a program, which must be in machine language to be effective, demonstrating how well these ideas work. I have not attempted this for a very good reason. I have been trying for somewhat more than ten years to get across the (to me) obvious advantages of a wider use of logarithms, and am fully conversant with the mathematical side of the story. My acquaintance with computers, on the other hand, is limited to building my Digital Group unit, using it extensively for computing, preparation of text (including the manuscript for this paper) and plotting. All of this has been programmed in Basic. Up to now I have not found the time to get into machine language programming. In short, if the subject matter of this paper is to be proved properly, I need help. If any reader with the appropriate skill finds the subject appealing, I would be most anxious to work jointly on proving these ideas. The fundamental concept of N-logs is sound, and there is no doubt in my mind that they will make a valuable addition to computer techniques, but this needs to be proved. □

APPENDIX I

The Mathematical Basis for the Sum and Difference Factors

Note: In what follows it is assumed that X>Y. Antilog(a) is used to mean the result of raising the logarithmic base to the power "a", i.e., the inverse operation to log(a).

Let: A=log(X) B=log(Y) and C=A−B
Then: antilog(C)=X/Y
Let: H=Y/X=antilog(B−A)=antilog(−C)

The Development of the Sum Factor "S":

log(X−Y)=log[X(1−Y/X)]=log(X)+log(1−Y/X)
Let: S=log(1+Y/X)=log(1+H)
Then: log(X+Y)=A+S

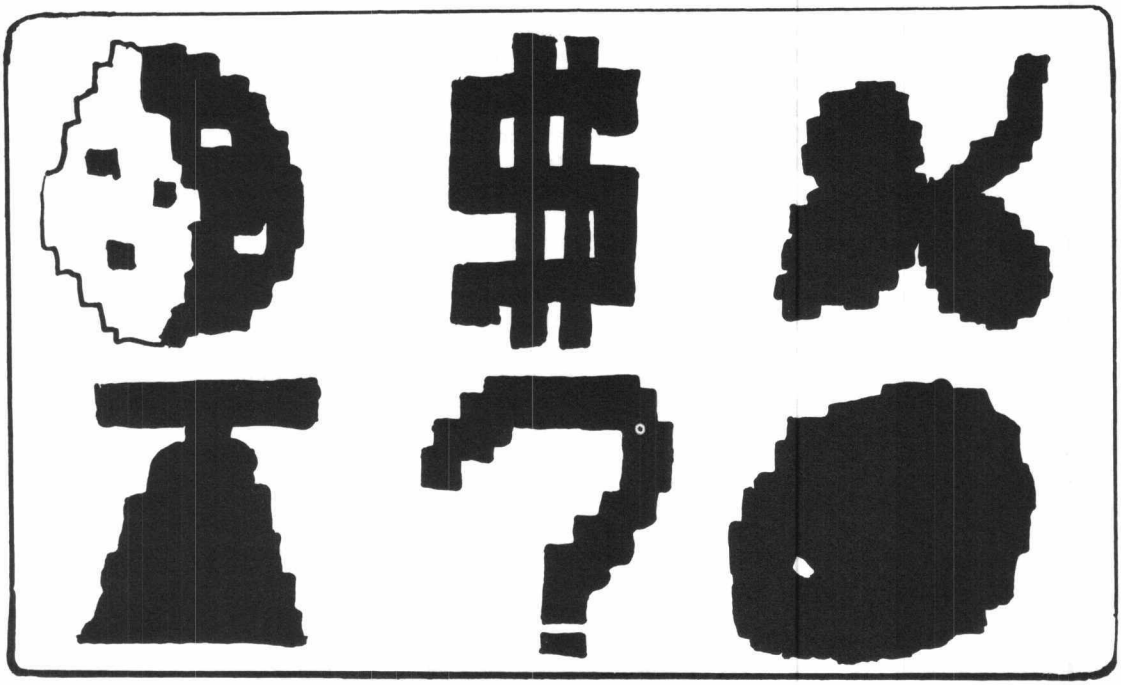
The Development of the Difference Factor "D":

log(X+Y)=log[X(1+Y/X)]=log(X)+log(1+Y/X)
Let: D=−log(1−Y/X)=−log(1−H)
Then: log(X−Y)=A−D

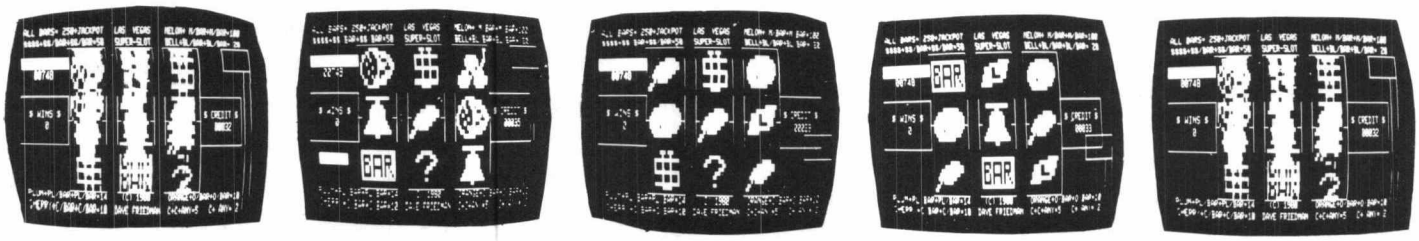
(continued on page 36)

$ $ $ $ $ $ $ $ $ $

LAS VEGAS SUPER-SLOT



a CP/M game machine program using Flashwriter I graphics



© by David W. Friedman

Eiko and I were spending the year-end holidays quietly at home with my word-processing-oriented North Star patiently waiting for the next major copywriting assignment which is how I earn our keep. We had been recounting the fun of playing the slot machines in Macau and the Philippines when she challenged me to put the game up on the computer. I agreed on the condition that she help with the graphics and off we went.

THE HARDWARE

A North Star Horizon II double-density system with 56K of RAM, a Vector Graphics Flashwriter I video board at E000, a 12V TV converted to a monitor and an 80-column friction-feed Teletype 43. Other video boards may have the same or similar graphics provisions but I know of at least two which don't, so you'll have to check carefully or else rewrite the graphics routines to match what you have.

THE SOFTWARE

Las Vegas Super-Slot is programmed as a standard CP/M transient at 100H and requires CP/M both for loading and for internal file manipulations as well as input/output standardization. I have been running VEGAS.COM using a modified version of the MENU.COM program by James Frantz which appeared in Creative Computing's December 1979 issue. On both cold and warm starts, MENU is invoked so the user need only type a 2-digit number to access VEGAS.

THE GRAPHICS

Flashwriter I is an excellent video board for alphanumerics with limited graphic capabilities. Characters can be displayed as 16 lines of 64 characters per line. The graphics capability can be invoked randomly for any desired character positions which further subdivides each character location into a 3 high by 2 wide pixel (picture element) array. This yields an overall resolution of 128 pixels horizontally by 48 pixels vertically.

There are also four other graphics features each controlled by an isolated bit: reverse video, low intensity video, horizontal thin lines and vertical thin lines. Access to the twelve video bits for each character position is segregated into two memory areas which can be called video and attributes (VID and ATTB) such that each character-position's attributes are always exactly four RAM pages above the corresponding video byte. For example, if the board is located at E000H, the first character position's video byte will be at E000H while the four excess attribute bits will be located in the byte at E400H.

David W. Friedman – 2-8-24 Okubo, Shinjuku-Ku, Tokyo, Japan 160

VIDEO BIT DIAGRAM

```
ATTB=VID+400H          VID
 76543210              76543210
 1111XXXX              XXXXXXXX
     ||||_pixels on/off |||____|___ graphics pixel bits
     |||__horizontal    ||_________ non-graphic bit (40H)
     ||     |thin lines |__________ reverse video bit
     ||___vertical
     |____half-intensity shading
```

The designation of each bit is shown. Note that bit 6 (40H) of the video byte is undefined in the graphics mode; this bit has two graphics coding uses in the VEGAS program. The attribute bytes are actually equipped with RAM only for the lower nibble such that, for normal alphanumeric character usages, reading this memory will give an F0 hex byte. This is slightly inefficient usage of address space but, for an 8-bit microprocessor, yields efficient access to the remaining four attribute bits.

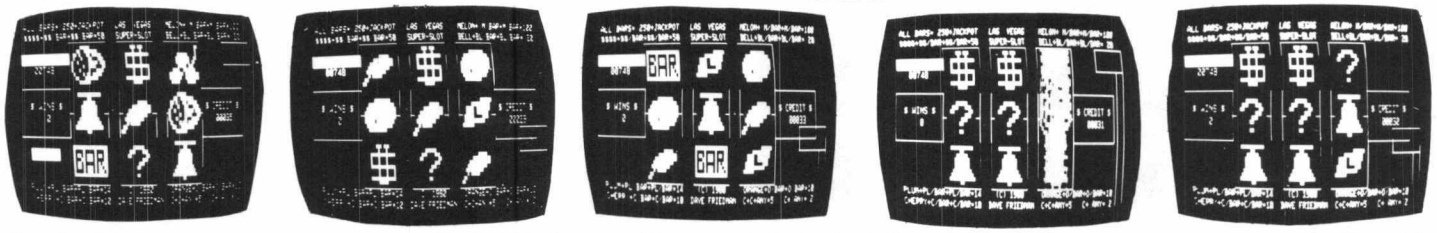
USING THE GRAPHICS

VEGAS begins by setting its own stack pointer, thus discarding the entry stack pointer, and calling the routine INITIZ which handles all necessary initialization, including graphics. VEGAS provides the user with a 3-reel/3-position down-rotating-reel presentation of a slot-machine plus a moving crank arm at the right and various status displays including winning combinations, current credits, current winning amount, current jackpot amount, ready indicator and others depending upon the current operating mode: TEST, PLAY, ZERO or READY. The initialization of the graphics functions is in the following order:

+ ERASE the screen to all spaces, 20H
+ apply the PAYOFF strings at top and bottom
+ apply left WIN and right CREDIT boxes
+ apply shading to top and bottom reel positions
+ apply graphics bits and REEL boxes at three positions
+ apply TICKS at centers of boxes
+ apply initial reel patterns using SPIN subroutine
+ apply operating arm using CRANK subroutine
+ show JACKPOT amount retrieved from separate file or create file and load it with 100 credits

There are several *very* generalized routines used to accomplish the above and usage of these is described here:

+ at entry, the B/C register must contain: either BLKOFF =0 or BLKON=8080H plus 100H times the vertical height (to B) plus the horizontal width (to C) defining an area.



+ at entry, the H/L register must contain: the beginning character position byte which is: either VID or ATTB plus the vertical line (0 to 15) times the linelength (LNLGTH=64) plus the horizontal character position (0 to 63).
+ BOXY requires at ATTB address in H/L to apply or remove a rectangular box dimensioned by B/C. Exceeding right or bottom limits does not wrap around nor write into non-graphic RAM addresses. BOXY uses the GRAPH and GPHRHZ routines and may be used for zero width or height boxes (lines only) although VEGAS does not contain such usages.
+ GRAPH requires an ATTB address in H/L and the bit of interest in the A register to apply or remove that bit from the rectangular area defined by B/C. GRAPH uses the GRAPHER routine.
+ GRAPHON is the same as GRAPH with A being set by the routine to the graphics bit, 0001B.
+ GRAPHER applies bits to one horizontal line at a time
+ GPHRHZ is the same as GRAPHER with A being set by the routine to the horizontal line bit, 0010B.

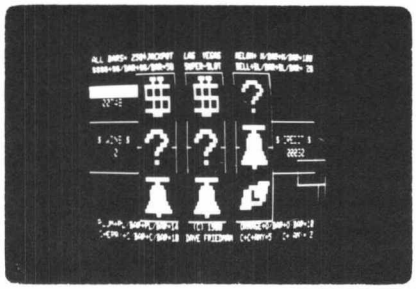
In addition to these generalized routines, two specialized routines are used:

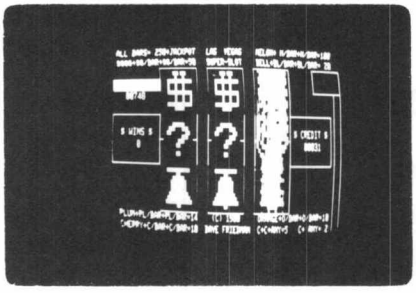
+ CRANK which calls HANDON and HANDOF both of which call the generalized BOXY routines.
+ MVPTN2 which requires the area to be specified in B/C and the upper-left byte to be specified in H/L. MVPTN is set for areas 10 characters wide by 12 high and thus moves 120 bytes to the block of video memory specified.

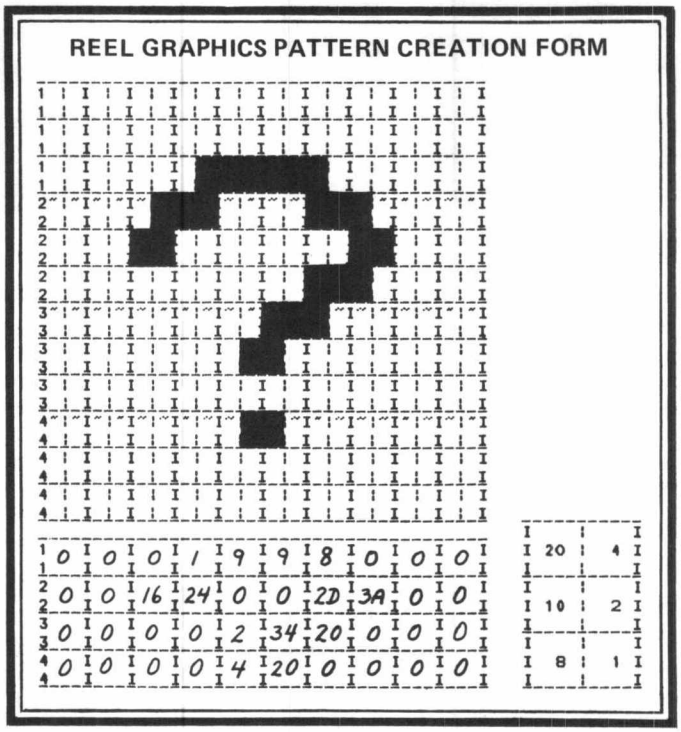
These routines provide access to every graphics capability of the Flashwriter I video board and *are readily applicable to use within other, unrelated programs.*

FORMING THE GRAPHIC PATTERNS

As an aid to rapid creation of the reel-graphics, the form shown was very efficient. It incorporates a pixel aspect ratio of 1.7:1 compared with about 2:1 actually obtained (depending upon monitor adjustment) such that penciling-in the desired blocks yields a very slightly squat but proportional black on white representation of what the final pixel pattern will look like when shown on the video screen. However, it was found that short video display routine to test the final appearance was essential to finalize the individual patterns.







Having settled upon the graphic presentation desired, it is only necessary to code the individual bytes by referring to the form's coding diagram wherein each pixel's bit value is given in hexadecimal; adding gives the byte value. VEGAS uses 3FH for all white pixels within a character position rather than the optional reverse video function 80H. Bit 6 which has value 40H is reserved for identification coding of the patterns and is added to the first four bytes of the pattern for this purpose. In this way, the final pattern's code can be read out of the video memory after the reels have stopped.

THE MAIN PROGRAM

After "READY" is shown, the KEYTEST routine looks for any keypress while incrementing the B/C register. When a keypress is obtained, C is saved as a RANDOM number before returning. The key is read to register A and tested against '$' (credit entry) and the first letter of the four keywords EXIT, TEST, ZERO and PLAY before testing for a CR. Any of these routines, if called, will return with a CR in A if the call was inappropriate, thus executing an ACCEPT/OPERATE cycle. Keywords are control key oriented by ANDing each reference string letter with CTRL=1FH and represent passwords to be known only to the user and not to the players.

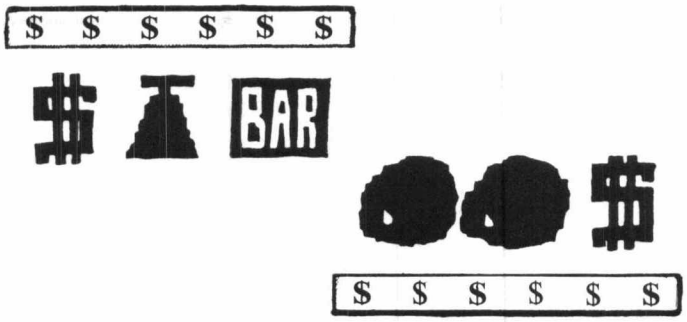
+ '$' issues credits only if no credits remain. If credit is issued, the jackpot is increased by about 15% of the credited amount up to a maximum of 15 jackpot credits while receipts are increased by the full credited amount.
+ EXIT erases the entire video memory and saves the jackpot and receipts before branching to a CP/M warm start.
+ TEST applies 100 credits and automatically runs the game until credits reach 0 or until the CR key is pressed. Jackpots are not paid in TEST and it returns with credits set to 0 in either case and can be used to cancel a winning player's credits.
+ ZERO sets receipts accumulated from '$' collections to 0, shows 0 receipts, saves the jackpot and receipts and, upon pressing the CR key, restores the READY condition. This feature allows the user to leave a player in charge with means to check the receipts when he later returns. *(continued on page 14)*

+ PLAY shows current receipts from '$' collections, saves the receipts and jackpot and, upon pressing the CR key, restores a READY condition without having affected the current credits.

ACCEPT checks to make sure credits are available before permitting OPERATE to be called. If credit is available, the READY sign is removed and OPERATE is called. If OPERATE returns with the carry set, the jackpot is paid out and a zeroed jackpot is saved to disk. All internal branches are back to MAIN which restores the READY mode.

THE OPERATE ROUTINE

One credit is CHARGEd for each play, the winning value is zeroed, the arm is CRANKed with sound effects, SPEED sets maximum reel running times based on the RANDOM number generated during KEYTEST, the reels are spun with stopping sound effects and options for individual reel–stopping control using keys '1', '2' and '3', a WINCHeck is performed which increases the jackpot for certain high payoff combinations, the winning amount is shown and, if not zero, is CREDitedOUT through one–by-one adding to the credits with sound effects. CREDOUT returns with the carry set if the win was three bars in order to pay out the jackpot.



THE SPIN ROUTINE

Here, speed of 'rotation' is very important. The use of a fast pattern stuffing routine and video pattern internal indentification coding were necessary to obtain sufficient speed even with the North Star's 4MHz Z-80, although no attempt was made to use any of the special Z-80 move instructions since these are not compatible with CP/M's assembler. If running on a 2 MHz 8080A, this routine's code can be expanded to eliminate internal subroutine calls which will help somewhat.

Initial entry is at SPIN1 with 12 in the A register to insure the movement of at least 3 patterns before STOP-KEY checking. SPINCouNT and each of the REELnS global variables must be maintained at multiples of 4 to keep the patterns centered on the TICKS at the middle position of each reel. SPEED presets the globals prior to SPIN entry and multiples of 4 pattern movements insure synchronization.

If any reel is stopped while others continue to spin, the extra time must be taken up to avoid speed increases of the remaining spinning reels and this is the function of the calls to TIMER1 in the SPINER routine. The last 16 counts of the REELnS globals are sub-divided into four sets of which three invoke TIMER1 to fill time and one outputs a space to the printer for sound effects in addition to advancing the pattern by 1/4 which gives the slowing-stop with sound effects. The pattern for the CHERRY has non-graphic 40H start byte and signals the RESETP section of SPINER's code to reset the pattern to CHERRYY in order to close the pattern loop similarly to what would exist on an actual slot machine reel.

STOPCMP checks the REELnS globals for the required all-zero stopped return condition. STOPKEY checks for a keypress of '1', '2' or '3' and adjusts the respective REELnS global variable to bring about a stop with slowing and with 'slip' determined both by the reel number and the RANDOM number generated at the start of each cycle by MAIN's KEYTEST. Multiples of 4 must be maintained and register A is set to 8 to introduce a minimum time interval between acceptance of STOPKEY presses prior to jumping back to SPIN1.

THE WINCHECK ROUTINE

WINCHK 'reaches' into the video memory and extracts the 2nd through the 4th byte's 40H bit identification codes with BYTINA. The 3 reel's bytes are stored in E, D and A respectively. Comparisons and BAR substitutions are used to determine winning combinations with winning values taken from the PAYTaBLe. The winning amount is saved as the VALUE global and, depending upon the amount, an addition is made to the jackpot.

BIOS AND BDOS CALLS

All BIOS console calls and printer calls are made directly to the BIOS branching table whose high byte is obtained from the warm start address stored in the jump at 0H. Disk operations to create and manipulate the incidental jackpots.lot file, use the normal BDOS calling conventions except, upon return from every BDOS call, a dental jackpots.lot file, use the normal BDOS calling conventions except, upon return from every BDOS call, a check is made to see if an error occurred. In case of a BDOS error, the graphics initialization is destroyed so an error message is shown indicating return to CP/M via a warm start.

SHOWING STRINGS, NUMBERS AND MATCHING KEYWORDS

All strings are terminated with nulls. The SHOSIG routines detects the null for return after advancing the D/E string pointer so that a subsequent call can be made to show another string following the null without resetting the pointer. Reverse video is built-into the strings for READY and JACKPOT.

All changing numbers are shown as 5-digit integers using SHO5DIG which can suppress leading zeros if entered with b = 0. Leading zeros are suppressed for WINS but are retained for all other changing numbers.

All MATCH strings are ANDed with CTRL = 1FH to invoke the need of pressing the CTRL key with all of the keyword letters and, each string is terminated with a null. The MATCH routine returns with the zero flag set if successful or not set if there was no MATCH.

LISTING AND AVAILABILITY OF DISKS

The complete CP/M assembly listing follows except for the data bytes representing the reel patterns which are printed as a memory dump using CP/M's DDT in order to save printing space. If you'd like to save yourself the typing time and associated errors, I can make the VEGAS.ASM and VEGAS.COM files available but only on North Star 10-hard-sectored disks for $50 and, I'll assemble VEGAS.COM to your selected video location of the Flashwriter I board if you mention it.

CONCERNING WINNING STATISTICS

If one discounts the effect of using the three number keys (1, 2 and 3) to control the stopping of the reels, the 'payback' expectations are as summarized here. However, it is possible to 'help' things along by using these 3 keys.

EXPECTATION TABLE

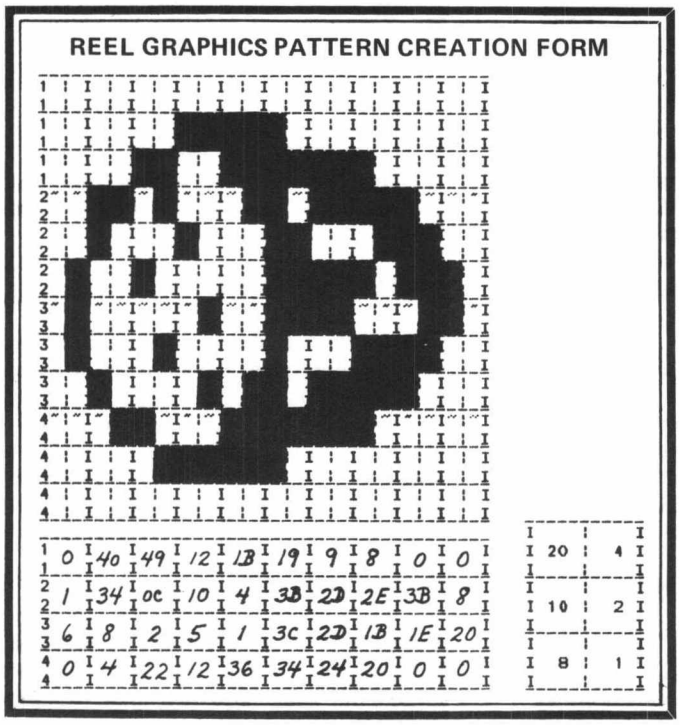
ITEM	#	COMB.	PAY	EXP.	JBONUS	JKPOT
BARS	2	8	250+J	2000	0	0
MELONS	2	24	100	2400	13	312
DOLLARS	2	24	50	1200	7	168
BELLS	4	112	20	2240	3	336
PLUMS	4	112	14	1568	2	224
ORANGES	5	185	10	1850	0	0
CHERRIES	3	57	10	570	0	0
first 2		207	5	1035	0	0
first 1		1380	2	2760	0	0
LEMONS	5	185	0	0	1	185
CHARGES	0	(146)	0	0	3	438
STOTAL	27	2294	--	15623	--	1663
NO WINS	--	17389	0	0		
JACKPOTS	--	(8)	208	1664		
TOTALS	27	19683	--	17287		

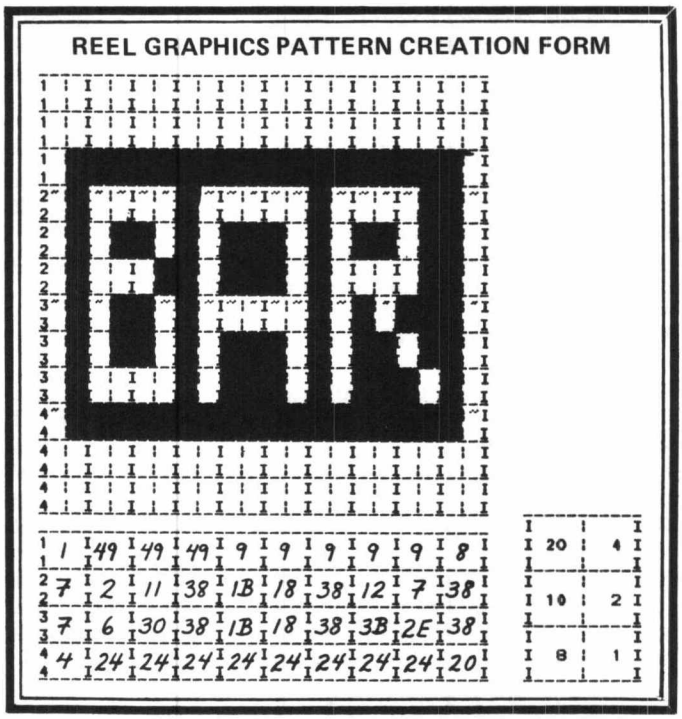
$$\text{PAYBACK} = \frac{17287}{19683} = 87.8 \text{ percent}$$

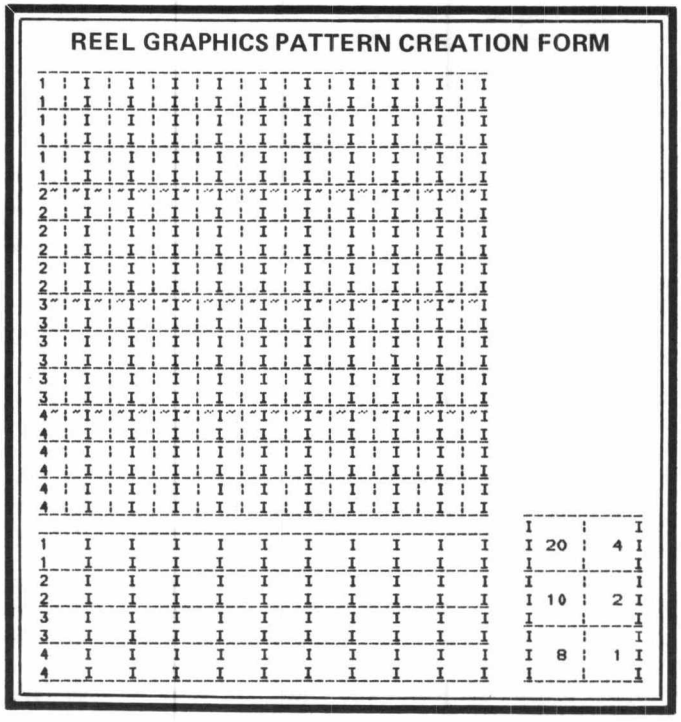
(skill not considered)

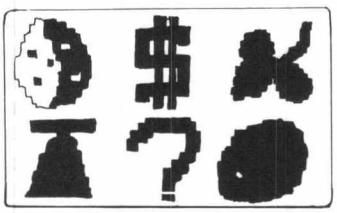
While our experience has shown that practice time tends to yield a higher percentage of bigger wins, at the same time the number of credits required to obtain these wins increases! This is caused by the player trying to lineup the reels, as they fly by, using readily recognizable patterns such as bars and dollar signs but, because of the reel 'slip' factor introduced by STOPKEY as a function of the RANDOM number, he is generally one pattern away from success. Not wanting to disturb his progress in hand, the player keeps trying to adjust for the single offset of the patterns and is thus, barring good luck, more often than not defeated. It would be blatantly untruthful to claim that this was not a planned perversity of the VEGAS game machine.

In view of the above, it can be said that there is the possibility of skillful manipulation of the game but, just as in real life, the most skillful approach may not yield the most profitable result. Eiko and I do hope you will enjoy playing the LAS VEGAS SUPER-SLOT game machine as much as we do. □









Listing

```
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                ; LAS  VEGAS  SUPER-SLOT  GAME MACHINE PROGRAM
                ;       (C) 1980 , DAVE FRIEDMAN
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                ;       USES FLASHWRITER I VIDEO @ E000
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
0100            VEGAS   ORG     100H    ;STD CPM TRANSIENT
                ;+++++++++++++++++++++++++++++++++++++++++++++++++
E000 =          VID     EQU     0E000H
00E0 =          VIDH    EQU     VID/100H
E400 =          ATTB    EQU     VID+400H
00E4 =          ATTBH   EQU     ATTB/100H
00E8 =          VIDNDH  EQU     VIDH+8
0040 =          LNLGTH  EQU     64
0010 =          SCRNHT  EQU     16
8080 =          BLKON   EQU     8080H
0000 =          BLKOFF  EQU     0
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
0000 =          WMSTT   EQU     0       ;CP/M WARM START ADDRESS
0005 =          BDOS    EQU     5       ;DISK OPERATIONS
005C =          FCB     EQU     5CH     ;FILE CONTROL BLOCK
0080 =          DMAD    EQU     80H     ;DEFAULT DMA ADDRESS
000D =          SETBDOS EQU     13      ;DRWA,DEFAULT DMA
000F =          OPENF   EQU     15      ;FCB OUT,ADDR OR FFH RETD
0010 =          CLOSEF  EQU     16      ;SAME DATA AS OPENF
0013 =          DELETEF EQU     19      ;FCB OUT
0014 =          READF   EQU     20      ;FCB OUT,0=OK,1=FAR,2=BAD
0015 =          WRITEF  EQU     21      ;0=OK,1=BAD,2=NODSK,FF=NODCTY
0016 =          CREATF  EQU     22      ;FCB OUT,ADDR OR FFH=NODCTY
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
001F =          CTRL    EQU     00011111B       ;ASSEMBLER CTRL KEY MASK
000D =          CRMK    EQU     CTRL AND 'M'
0008 =          BKSPMK  EQU     CTRL AND 'H'
0007 =          BELMK   EQU     CTRL AND 'G'
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
0100 31570D     START:  LXI     SP,STKPTR
0103 CD2202             CALL    INITIZ
                ;+++++++++++++++++++++++++++++++++++++++++++++++
0106 CD5706     MAIN    CALL    READY
0109 CDE602             CALL    KEYTEST
010C CD7906             CALL    PCREAD
010F FE24               CPI     '$'
0111 CCF302             CZ      COLLECT
0114 FE05               CPI     CTRL AND 'E
0116 CC4501             CZ      EXIT
0119 FE14               CPI     CTRL AND 'T'
011B CC9A01             CZ      TEST
011E FE1A               CPI     CTRL AND 'Z'
0120 CCDB01             CZ      ZERO
0123 FE10               CPI     CTRL AND 'P'
0125 CCED01             CZ      PLAYS
0128 FE0D               CPI     CRMK
012A C20601             JNZ     MAIN
012D CD4D03             CALL    ACCEPT
0130 CA0601             JZ      MAIN
0133 CD5106             CALL    NREADY
0136 CD5701             CALL    OPERATE
0139 D20601             JNC     MAIN
013C CD1003             CALL    PAYJACK
013F CD0606             CALL    SVJACK
0142 C30601             JMP     MAIN
                ;+++++++++++++++++++++++++++++++++++++++++++++++
0145 216907     EXIT    LXI     H,EXITSTG+1
0148 CD1602             CALL    MATCH
014B 3E0D               MVI     A,CRMK
014D C0                 RNZ
014E CD6606             CALL    ERASE
0151 CD0606             CALL    SVJACK
0154 C30000             JMP     WMSTT   ;BACK TO CP/M
                ;+++++++++++++++++++++++++++++++++++++++++++++++++
0157 CD5905     OPERATE CALL    CHARGE
015A AF                 XRA     A
015B 32610D             STA     VALUE
015E CD7405             CALL    SHOWIN
0161 CD5303             CALL    CRANK
0164 CDF603             CALL    SPEED
0167 3E0C               MVI     A,12    ;MIN INITIAL SPIN X4
0169 CD1904             CALL    SPIN1
016C CDE804             CALL    WINCHK
016F CD7405             CALL    SHOWIN
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
0172 3A610D     CREDOUT LDA     VALUE
0175 B7                 ORA     A       ;CY=0
0176 C8                 RZ
0177 5F                 MOV     E,A
0178 2A630D             LHLD    CREDTV
017B 23         BUMP    INX     H
017C 22630D             SHLD    CREDTV
017F CD8906             CALL    BKSOUT
0182 E5D5               PUSH H! PUSH D
0184 CD6005             CALL    SHOCRED
0187 D1E1               POP D!  POP H
0189 CDD504             CALL    TIMERX
018C 1D                 DCR     E
018D C27B01             JNZ     BUMP
0190 CD8406             CALL    SPCOUT  ;AT LEAST 1 SPACE IF BIG PAY
0193 3A610D             LDA     VALUE
0196 D6FA               SUI     250
0198 3F                 CMC
0199 C9                 RET             ;CY=1 ,PAY JACKPOT
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
019A 217307     TEST    LXI     H,TSTMCH+1
019D CD1602             CALL    MATCH
01A0 3E0D               MVI     A,CRMK
01A2 C0                 RNZ
01A3 114107             LXI     D,TSTSTG
01A6 CD5A06             CALL    READY1
01A9 216400             LXI     H,100
01AC CD5D05             CALL    CHARGE1
01AF 260A       TEST1   MVI     H,10
01B1 CDD504     TEST2   CALL    TIMERX
01B4 25                 DCR     H
01B5 C2B101             JNZ     TEST2
01B8 21650D             LXI     H,RANDOM
01BB 34                 INR     M       ;SINCE USER NOT IN CONTROL
01BC CD7306             CALL    PCIN
01BF B7                 ORA     A
01C0 CAD101             JZ      TEST3
01C3 CD7906             CALL    PCREAD
01C6 FE0D               CPI     CRMK
01C8 C2D101             JNZ     TEST3
01CB 210000             LXI     H,0
01CE C35D05             JMP     CHARGE1
01D1 CD4D03     TEST3   CALL    ACCEPT
01D4 C8                 RZ
01D5 CD5701             CALL    OPERATE
01D8 C3AF01             JMP     TEST1
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
01DB 216E07     ZERO    LXI     H,ZEROSTG+1
01DE CD1602             CALL    MATCH
01E1 3E0D               MVI     A,CRMK
01E3 C0                 RNZ
01E4 210000             LXI     H,0
01E7 22690D             SHLD    RECIPT
01EA C3F601             JMP     PLAYS1
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
01ED 217807     PLAYS   LXI     H,PLAYSTG+1
01F0 CD1602             CALL    MATCH
01F3 3E0D               MVI     A,CRMK
01F5 C0                 RNZ
01F6 115407     PLAYS1  LXI     D,RCPTSTG
01F9 21F1E1             LXI     H,VID+7*LNLGTH+49
01FC CD5D06             CALL    SHOSTG
01FF CD5106             CALL    NREADY
0202 2A690D             LHLD    RECIPT
0205 CD6C05             CALL    SHOCRD1
```

```
0208 CD0606             CALL    SVJACK
020B CD7906     PLAYS2  CALL    PCREAD
020E FE0D               CPI     CRMK
0210 C20B02             JNZ     PLAYS2
0213 C36005             JMP     SHOCRED
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
0216 CD7906     MATCH   CALL    PCREAD
0219 BE                 CMP     M
021A 23                 INX     H
021B C0                 RNZ
021C 7E                 MOV     A,M
021D B7                 ORA     A
021E C21602             JNZ     MATCH
0221 C9                 RET
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
0222 CD6606     INITIZ  CALL    ERASE
0225 2100E0             LXI     H,VID+0*LNLGTH+0
0228 118407             LXI     D,PAYOFF
022B CD5D06             CALL    SHOSTG
022E 2180E3             LXI     H,VID+14*LNLGTH+0
0231 CD5D06             CALL    SHOSTG
0234 018C84             LXI     B,BLKON+100H*4+12       ;4 X 12 AREA
0237 2180E5             LXI     H,ATTB+6*LNLGTH+0
023A CD8C03             CALL    BOXY
023D 21B0E5             LXI     H,ATTB+6*LNLGTH+48
0240 CD8C03             CALL    BOXY
0243 01A484             LXI     B,BLKON+100H*4+36       ;4 X 36 AREA
0246 218CE6             LXI     H,ATTB+10*LNLGTH+12
0249 3E08               MVI     A,8     ;HALF-INTENSITY SHADING
024B CDB503             CALL    GRAPH
024E 218CE4             LXI     H,ATTB+2*LNLGTH+12
0251 E5                 PUSH    H
0252 CDB503             CALL    GRAPH
0255 E1E5               POP H!  PUSH H
0257 018A8C             LXI     B,BLKON+100H*12+10      ;12 X 10 AREA
025A CDB303             CALL    GRAPHON
025D E1                 POP     H
025E CD8C03             CALL    BOXY
0261 110D00             LXI     D,(25-12) OR (38-25)    ;CORNER DELTA
0264 19E5               DAD D!  PUSH H
0266 CDB303             CALL    GRAPHON
0269 E1                 POP     H
026A CD8C03             CALL    BOXY
026D 19E5               DAD D!  PUSH H
026F CDB303             CALL    GRAPHON
0272 E1                 POP     H
0273 CD8C03             CALL    BOXY
0276 210CE6             LXI     H,ATTB+8*LNLGTH+12
0279 010203             LXI     B,0302H ;B=3 COLS, C=HZ LINE GPX
027C E5         TICKS   PUSH    H
027D 79                 MOV     A,C
027E B6                 ORA     M
027F 77                 MOV     M,A
0280 7D                 MOV     A,L
0281 C609               ADI     9       ;10-1=BOX WIDTH-1
0283 6F                 MOV     L,A
0284 79                 MOV     A,C
0285 B6                 ORA     M
0286 77                 MOV     M,A
0287 E119               POP H!  DAD D
0289 05                 DCR     B
028A C27C02             JNZ     TICKS
028D 21CF08             LXI     H,BELL  ;INITIAL REEL DISPLAY
0290 225F0D             SHLD    REEL3P
0293 21C70B             LXI     H,BELL1X
0296 225D0D             SHLD    REEL2P
0299 21AF0A             LXI     H,BELLX
029C 225B0D             SHLD    REEL1P
029F 211010             LXI     H,1010H ;LAST 16 COUNTS=4 LINES
02A2 22580D             SHLD    REEL1S
02A5 22590D             SHLD    REEL2S  ;INCLUDES REEL3S
02A8 CD1704             CALL    SPIN    ;APPLIES PATTERNS
02AB CD5303             CALL    CRANK   ;APPLIES HANDLE
02AE 210000             LXI     H,0
02B1 22610D             SHLD    VALUE
02B4 22630D             SHLD    CREDTV
02B7 22690D             SHLD    RECIPT
02BA CD7405             CALL    SHOWIN
02BD CD6005             CALL    SHOCRED
02C0 CDD905             CALL    LDJACK
02C3 F2CC02             JP      SHOJACK
02C6 216400             LXI     H,100
02C9 22670D     INIT2   SHLD    JACKPOT
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
02CC CD7F06     SHOJACK CALL    BELOUT
02CF CDD504             CALL    TIMERX
02D2 111307             LXI     D,JACKSTG
02D5 21C0E0             LXI     H,VID+3*LNLGTH+0
02D8 CD5D06             CALL    SHOSTG
02DB 1103E1             LXI     D,VID+4*LNLGTH+3
02DE 2A670D             LHLD    JACKPOT
02E1 0601               MVI     B,1
02E3 C38505             JMP     SHO5DIG
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
02E6 CD7306     KEYTEST CALL    PCIN    ;RETURNS A=1 IF KEY WAITING
02E9 B7                 ORA     A
02EA 03                 INX     B
02EB CAE602             JZ      KEYTEST
02EE 79                 MOV     A,C
02EF 32650D             STA     RANDOM
02F2 C9                 RET
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
02F3 CD4D03     COLLECT CALL    ACCEPT
02F6 3E0D               MVI     A,CRMK
02F8 C0                 RNZ             ;CREDIT REMAINING
02F9 111400             LXI     D,20
02FC 2A690D             LHLD    RECIPT
02FF 19                 DAD     D
0300 22690D             SHLD    RECIPT
0303 2A630D             LHLD    CREDTV
0306 19D5               DAD D!  PUSH D
0308 CD5D05             CALL    CHARGE1
030B D1                 POP     D
030C 7B                 MOV     A,E
030D C33505             JMP     PLJACK
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
0310 2A670D     PAYJACK LHLD    JACKPOT
0313 11F6FF             LXI     D,-10
0316 19                 DAD     D
0317 D24703             JNC     PAID
031A 22670D             SHLD    JACKPOT
031D 2A630D             LHLD    CREDTV
0320 110A00             LXI     D,10
0323 19                 DAD     D
0324 CD5D05             CALL    CHARGE1
0327 CDCC02             CALL    SHOJACK
032A 111307             LXI     D,JACKSTG
032D 2100E2             LXI     H,VID+8*LNLGTH+0
0330 E5                 PUSH    H
0331 CD5D06             CALL    SHOSTG
0334 CD8406             CALL    SPCOUT
0337 CD8906             CALL    BKSOUT
033A CD8906             CALL    BKSOUT
033D CDD504             CALL    TIMERX
0340 E1                 POP     H
0341 CD5D06             CALL    SHOSTG  ;NJAKSTG FOLLOWS JACKSTG
0344 C31003             JMP     PAYJACK
0347 210000     PAID    LXI     H,0
034A C3C902             JMP     INIT2
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
034D 2A630D     ACCEPT  LHLD    CREDTV
0350 7C                 MOV     A,H
0351 B5                 ORA     L
0352 C9                 RET
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
0353 21B7E4     CRANK   LXI     H,ATTB+2*LNLGTH+55
0356 E5                 PUSH    H
0357 CD7F03             CALL    HANDOF
```

(continued)

```
035A 2177E6            LXI     H,ATTB+9*LNLGTH+55
035D E5                PUSH    H
035E CD6F03            CALL    HANDON
0361 E1                POP     H
0362 CDD504            CALL    TIMERX
0365 CD8E06            CALL    CROUT
0368 CDD504            CALL    TIMERX
036B CD7F03            CALL    HANDOF
036E E1                POP     H
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
036F 018882     HANDON LXI     B,BLKON+100H*2+8        ;2 X 8 BOX
0372 CD8C03            CALL    BOXY
0375 118600            LXI     D,2*LNLGTH+6    ;LEVER RELATIVE POS
0378 19                DAD     D
0379 01828F            LXI     B,BLKON+100H*15+2       ;FULL X 2 BOX
037C C38C03            JMP     BOXY
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
037F 010802     HANDOF LXI     B,BLKOFF+100H*2+8
0382 CD8C03            CALL    BOXY
0385 118600            LXI     D,2*LNLGTH+6
0388 19                DAD     D
0389 01020F            LXI     B,BLKOFF+100H*15+2
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
038C D5E5E5C5   BOXY   PUSH D! PUSH H! PUSH H! PUSH B
0390 CDD903            CALL    GPHRHZ  ;HZ LINES
0393 C1C5              POP B!  PUSH B
0395 B7                ORA     A
0396 CCA803            CZ      BOXY2   ;IF NOT PAST RIGHT
0399 C1E1C5            POP B!  POP H!  PUSH B
039C CDA803            CALL    BOXY2   ;LEFT VERT LINES
039F C1C5              POP B!  PUSH B
03A1 CDD903            CALL    GPHRHZ  ;BTM HZ LINES
03A4 C1E1D1            POP B!  POP H!  POP D
03A7 C9                RET
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
03A8 3E80       BOXY2  MVI     A,80H
03AA A1                ANA     C       ;PICKUP BLKON/OFF CONDITION
03AB F601              ORI     1       ;ONLY 1 WIDE NEEDED
03AD 4F                MOV     C,A
03AE 3E04              MVI     A,0100B ;VERT LINE GPX
03B0 C3B503            JMP     GRAPH
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
03B3 3E01       GRAPHON MVI    A,0001B ;GRAPHICS BIT
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
03B5 D5C5F5     GRAPH  PUSH D! PUSH B! PUSH A
03B8 114000            LXI     D,LNLGTH
03BB 78         GRAPH1 MOV     A,B
03BC E60F              ANI     SCRNHT-1
03BE CAD503            JZ      GRAPHED ;IF NO MORE
03C1 F1F5              POP A!  PUSH A
03C3 C5E5              PUSH B! PUSH H
03C5 CDDB03            CALL    GRAPHER
03C8 E1C1              POP H!  POP B
03CA 19                DAD     D
03CB 7C                MOV     A,H
03CC FEE8              CPI     VIDNDH
03CE D2D503            JNC     GRAPHED ;IF PAST BTM
03D1 05                DCR     B
03D2 C3BB03            JMP     GRAPH1
03D5 F1C1D1     GRAPHED POP A! POP B!  POP D
03D8 C9                RET
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
03D9 3E02       GPHRHZ MVI     A,0010B ;HZ LINE GPX
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
03DB 47         GRAPHER MOV    B,A
03DC 79         GRAPHR1 MOV    A,C
03DD E63F              ANI     LNLGTH-1
03DF C8                RZ              ;IF NO MORE
03E0 78                MOV     A,B
03E1 2F                CMA             ;CREATE BIT MASK
03E2 A6                ANA     M
03E3 77                MOV     M,A     ;ZERO CURRENT SETTING
03E4 79                MOV     A,C
03E5 B7                ORA     A
03E6 F2EC03            JP      GRPHOF  ;IF BLKOFF THEN DONE
03E9 78                MOV     A,B
03EA B6                ORA     M
03EB 77                MOV     M,A     ;SET BIT TO 1
03EC 23         GRPHOF INX     H       ;NEXT BYTE
03ED 7D                MOV     A,L
03EE E63F              ANI     LNLGTH-1
03F0 7C                MOV     A,H     ;A REG. NON-ZERO ON RETURN
03F1 C8                RZ              ;IF PAST RIGHT EDGE
03F2 0D                DCR     C
03F3 C3DC03            JMP     GRAPHR1
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
03F6 11580D     SPEED  LXI     D,REEL1S
03F9 2A650D            LHLD    RANDOM  ;IN L
03FC 3E30              MVI     A,00110000B
03FE A5                ANA     L
03FF F680              ORI     80H
0401 12                STAX    D       ;RANGE 80,90,A0,B0
0402 13                INX     D       ;REEL2S
0403 3E18              MVI     A,00011000B
0405 A5                ANA     L
0406 F6C0              ORI     0C0H
0408 12                STAX    D       ;RANGE C0,C8,D0,D8
0409 13                INX     D       ;REEL3S
040A 3E0C              MVI     A,00001100B
040C A5                ANA     L
040D F610              ORI     10H
040F 67                MOV     H,A     ;RANDOM+1 RANGE 10,14,18,1C
0410 22650D            SHLD    RANDOM  ;SPEC IN H (RANDOM+1)
0413 F6F0              ORI     0F0H
0415 12                STAX    D       ;RANGE F0,F4,F8,FC
0416 C9                RET
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
0417 3E04       SPIN   MVI     A,4     ;4 LINES/PATTERN
0419 32570D     SPIN1  STA     SPINCNT
041C 01580D            LXI     B,REEL1S
041F 2A5B0D            LHLD    REEL1P
0422 118CE0            LXI     D,VID+2*LNLGTH+12
0425 CD8F04            CALL    SPINER
0428 225B0D            SHLD    REEL1P
042B 01590D            LXI     B,REEL2S
042E 2A5D0D            LHLD    REEL2P
0431 1199E0            LXI     D,VID+2*LNLGTH+25
0434 CD8F04            CALL    SPINER
0437 225D0D            SHLD    REEL2P
043A 015A0D            LXI     B,REEL3S
043D 2A5F0D            LHLD    REEL3P
0440 11A6E0            LXI     D,VID+2*LNLGTH+38
0443 CD8F04            CALL    SPINER
0446 225F0D            SHLD    REEL3P
0449 3A570D            LDA     SPINCNT
044C 3D                DCR     A
044D C21904            JNZ     SPIN1
0450 CD7306            CALL    PCIN
0453 B7                ORA     A
0454 C26304            JNZ     STOPKEY
0457 3A5A0D     STOPCMP LDA    REEL3S
045A 2A580D            LHLD    REEL1S  ;REEL2S IN H
045D B4                ORA     H
045E B5                ORA     L
045F C21704            JNZ     SPIN
0462 C9                RET
0463 CD7906     STOPKEY CALL   PCREAD
0466 2A660D            LHLD    RANDOM+1        ;SPEC IN L
0469 11580D            LXI     D,REEL1S
046C D631              SUI     '1'
046E 2600              MVI     H,0
0470 CA8104            JZ      STOPKY1
0473 3D                DCR     A
0474 13                INX     D       ;REEL2S
0475 2604              MVI     H,4
0477 CA8104            JZ      STOPKY1 ;IF '2'
047A 3D                DCR     A
047B C21704            JNZ     SPIN    ;IF NOT '3'
```

```
047E 13              INX     D       ;REEL3S
047F 2608            MVI     H,8
0481 1A        STOPKY1 LDAX    D
0482 FE11            CPI     11H
0484 DA1704          JC      SPIN    ;IF NOW STOPPING
0487 7C              MOV     A,H
0488 B5              ORA     L
0489 12              STAX    D       ;RANGES FROM 10H TO 1CH
048A 3E08            MVI     A,8     ;MIN TIME BETWEEN STOP KEYS
048C C31904          JMP     SPIN1
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
048F 0A        SPINER  LDAX    B
0490 B7        SPINER1 ORA     A
0491 CADA04          JZ      TIMER1  ;KEEP SPEEDS CONSTANT
0494 3D              DCR     A
0495 02              STAX    B
0496 FE11            CPI     11H
0498 D2A504          JNC     RESETP  ;IF NOT SLOWING
049B E603            ANI     3
049D FE01            CPI     1
049F D2DA04          JNC     TIMER1  ;FILL 3/4THS OF TIME
04A2 CD8406          CALL    SPCOUT  ;SOUND EFFECTS
04A5 7E        RESETP  MOV     A,M
04A6 FE40            CPI     40H     ;CHERRY MARKER
04A8 DAAE04          JC      RESETP1
04AB 21B70C          LXI     H,CHERRYY
04AE D5        RESETP1 PUSH    D
04AF 11F6FF          LXI     D,-10   ;10 BYTES/LINE
04B2 19              DAD     D
04B3 D1E5EB          POP D!  PUSH H! XCHG
04B6 CDBB04          CALL    MVPTN
04B9 E1              POP     H       ;RESTORE NEW PATN POS FOR SAVE
04BA C9              RET
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
04BB 010C0A    MVPTN   LXI     B,100H*10       ;10 WIDE X 12 HIGH
04BE C5E5      MVPTN2  PUSH B! PUSH H
04C0 1A        MVPTN1  LDAX    D
04C1 77                MOV     M,A
04C2 2313              INX H!  INX D
04C4 05                DCR     B
04C5 C2C004            JNZ     MVPTN1
04C8 E1C1D5            POP H!  POP B!  PUSH D
04CB 114000            LXI     D,LNLGTH
04CE 19D1              DAD D!  POP D   ;NEXT LINE, SAME HZ POS
04D0 0D                DCR     C
04D1 C2BE04            JNZ     MVPTN2
04D4 C9                RET
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
04D5 3EB4      TIMERX  MVI     A,180   ;ABOUT RIGHT FOR ME
04D7 C3DC04            JMP     TIMER
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
04DA 3E08      TIMER1  MVI     A,8     ;RUN A BIT FASTER;9 IS EXACT
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
04DC E5        TIMER   PUSH    H
04DD 67                MOV     H,A
04DE 3D        TIMEA   DCR     A
04DF C2DE04            JNZ     TIMEA
04E2 25                DCR     H
04E3 C2DE04            JNZ     TIMEA
04E6 E1                POP     H
04E7 C9                RET
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
04E8 AF        WINCHK  XRA     A
04E9 32610D            STA     VALUE
04EC 218DE1            LXI     H,VID+6*LNLGTH+13
04EF CD4605            CALL    BYTINA
04F2 5F                MOV     E,A
04F3 219AE1            LXI     H,VID+6*LNLGTH+26
04F6 CD4605            CALL    BYTINA
04F9 57                MOV     D,A
04FA 21A7E1            LXI     H,VID+6*LNLGTH+39
04FD CD4605            CALL    BYTINA
0500 FE07              CPI     7       ;BAR CODE
0502 C20A05            JNZ     NOTBAR
```



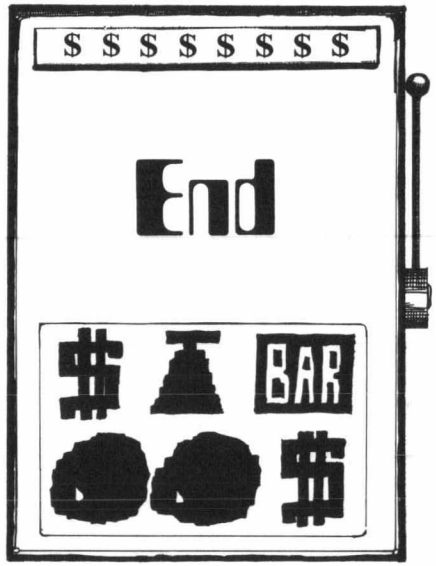
```
0505 BA                CMP     D
0506 CA1205            JZ      WINER3  ;ANY+2 BARS IS OK
0509 7B                MOV     A,E     ;REEL 3'S CODE=REEL 1'S
050A BB        NOTBAR  CMP     E       ;R3 VS. R1
050B C21C05            JNZ     LOWCHK
050E BA                CMP     D       ;R3 VS. R2
050F C21C05            JNZ     LOWCHK
0512 1600      WINER3  MVI     D,0
0514 217C07            LXI     H,PAYTBL
0517 19                DAD     D
0518 7E                MOV     A,M     ;TABLE+CODE=(WIN VALUE)
0519 C32805            JMP     STRVAL
051C 7B        LOWCHK  MOV     A,E
051D FE01              CPI     1       ;CHERRY'S CODE
051F C0                RNZ
0520 BA                CMP     D       ;R1 VS. R2
0521 3E02              MVI     A,2     ;FOR JUST FIRST CHERRY
0523 C22805            JNZ     STRVAL
0526 3E05              MVI     A,5     ;FOR BOTH CHERRY(IES)
0528 32610D    STRVAL  STA     VALUE
052B FEFA              CPI     250
052D C8                RZ
052E B7                ORA     A
052F CA3B05            JZ      ADDJAK  ;IF LEMON WON!
0532 FE0E              CPI     14
0534 D8                RC
0535 0F0F0F    PLJACK  RRC!    RRC!    RRC
0538 E60F              ANI     0FH
053A C8                RZ
053B 5F        ADDJAK  MOV     E,A
053C 1600              MVI     D,0
053E 13                INX     D
053F 2A670D            LHLD    JACKPOT
0542 19                DAD     D       ;ADDS FROM 1 TO 13 ON JACKPOT
0543 C3C902            JMP     INIT2
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
0546 3E40      BYTINA  MVI     A,01000000B     ;NON-DISPLAY CODE BIT
0548 4F                MOV     C,A
0549 A6                ANA     M
054A 07                RLC
054B 47                MOV     B,A
054C 79                MOV     A,C
054D 23                INX     H
054E A6                ANA     M
054F B0                ORA     B
0550 07                RLC
0551 47                MOV     B,A
0552 79                MOV     A,C
0553 23                INX     H
0554 A6                ANA     M
0555 B0                ORA     B
0556 0707              RLC!    RLC     ;ADJUST TO LOWEST BITS
0558 C9                RET
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
0559 2A630D    CHARGE  LHLD    CREDTV
055C 2B                DCX     H
055D 22630D    CHARGE1 SHLD    CREDTV
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
0560 114907    SHOCRED LXI     D,KREDIT
0563 21F1E1            LXI     H,VID+7*LNLGTH+49
0566 CD5D06            CALL    SHOSTG
0569 2A630D            LHLD    CREDTV
056C 1134E2    SHOCRD1 LXI     D,VID+8*LNLGTH+52
056F 0601              MVI     B,1     ;ALL ZEROS SHOW
0571 C38505            JMP     SHO5DIG
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
0574 115F07    SHOWIN  LXI     D,WINSTG
0577 21C2E1            LXI     H,VID+7*LNLGTH+2
057A CD5D06            CALL    SHOSTG
057D 1102E2            LXI     D,VID+8*LNLGTH+2
0580 2A610D            LHLD    VALUE
0583 0600              MVI     B,0     ;LEADING ZEROS SUPPRESSED
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++
0585 D5        SHO5DIG PUSH    D
```

(continued)

```
0586 11F0D8           LXI     D,-10000        ;X=10000 DIVISOR
0589 CDBE05           CALL    DIVX
058C D11213D5         POP D!  STAX D! INX D!  PUSH D
0590 1118FC           LXI     D,-1000
0593 CDBE05           CALL    DIVX
0596 D11213D5         POP D!  STAX D! INX D!  PUSH D
059A 119CFF           LXI     D,-100
059D CDBE05           CALL    DIVX
05A0 D11213D5         POP D!  STAX D! INX D!  PUSH D
05A4 11F6FF           LXI     D,-10
05A7 CDBE05           CALL    DIVX
05AA D11213D5         POP D!  STAX D! INX D!  PUSH D
05AE 11FFFF           LXI     D,-1
05B1 CDBE05           CALL    DIVX
05B4 D1               POP     D
05B5 FE20             CPI     ' '
05B7 C2BC05           JNZ     LASTD
05BA 3E30             MVI     A,'0'   ;SHOW AT LEAST 1 DIGIT IF ZERO
05BC 12       LASTD   STAX    D
05BD C9               RET
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
05BE 0E2F     DIVX    MVI     C,'0'-1
05C0 0C       DIVX1   INR     C
05C1 19               DAD     D
05C2 DAC005           JC      DIVX1
05C5 7B2F5F           MOV A,E!  CMA!  MOV E,A
05C8 7A2F57           MOV A,D!  CMA!  MOV D,A
05CB 13               INX     D       ;TWO'S COMPLEMENT IN D
05CC 19               DAD     D
05CD 79               MOV     A,C
05CE B0               ORA     B
05CF 47               MOV     B,A     ;SET SUPPRESS FLAG
05D0 FE30             CPI     '0'
05D2 C2D705           JNZ     NLZERO
05D5 0E20             MVI     C,' '   ;IF ONLY ZEROS TO HERE
05D7 79       NLZERO  MOV     A,C
05D8 C9               RET
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
05D9 3E05     LDJACK  MVI     A,5
05DB 32570D   LDJAK1  STA     TRIALS
05DE CDA306           CALL    SETDSK
05E1 CD4706           CALL    SETFCB
05E4 CDBC06           CALL    OPEN
05E7 F8               RM
05E8 CDC106           CALL    READ
05EB C2FD05           JNZ     TRYAGN
05EE 2A8000           LHLD    DMAD
05F1 22670D           SHLD    JACKPOT
05F4 2A8200           LHLD    DMAD+2
05F7 22690D           SHLD    RECIPT
05FA C3B706           JMP     CLOSE
05FD 3A570D   TRYAGN  LDA     TRIALS
0600 3D               DCR     A
0601 C2DB05           JNZ     LDJAK1
0604 3D               DCR     A
0605 C9               RET
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
0606 3E05     SVJACK  MVI     A,5
0608 32570D   SVJACK1 STA     TRIALS
060B CDA306           CALL    SETDSK
060E CD4706           CALL    SETFCB
0611 CDA806           CALL    DELETE
0614 CDAD06           CALL    CREATE
0617 FE37             CPI     55
0619 D0               RNC
061A 2A670D           LHLD    JACKPOT
061D 228000           SHLD    DMAD
0620 2A690D           LHLD    RECIPT
0623 228200           SHLD    DMAD+2
0626 CDB206           CALL    WRITE
0629 C23006           JNZ     RETRY
062C CDB706           CALL    CLOSE
062F F0               RP
```

```
0630 3A570D   RETRY   LDA     TRIALS
0633 3D               DCR     A
0634 C20806           JNZ     SVJACK1
0637 C9               RET
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
0638 012100   MVZFCB  LXI     B,21H
063B 215C00           LXI     H,FCB
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
063E E5       MVCYBM  PUSH    H
063F 70       MVCYBM1 MOV     M,B
0640 23               INX     H
0641 0D               DCR     C
0642 C23F06           JNZ     MVCYBM1
0645 E1               POP     H
0646 C9               RET
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
0647 CD3806   SETFCB  CALL    MVZFCB
064A 113507           LXI     D,JACKFL
064D 23               INX     H
064E C35D06           JMP     SHOSTG
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
0651 112507   NREADY  LXI     D,NREDY
0654 C35A06           JMP     READY1
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
0657 112D07   READY   LXI     D,REDY
065A 21C2E2   READY1  LXI     H,VID+11*LNLGTH+2
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
065D 1A       SHOSTG  LDAX    D
065E 13               INX     D       ;NOW TO SKIP STOP ON RET
065F B7               ORA     A
0660 C8               RZ
0661 77               MOV     M,A
0662 23               INX     H
0663 C35D06           JMP     SHOSTG
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
0666 3EE8     ERASE   MVI     A,VIDNDH
0668 2100E0           LXI     H,VID
066B 3620     ERASE1  MVI     M,' '
066D 23               INX     H
066E BC               CMP     H
066F C26B06           JNZ     ERASE1
0672 C9               RET
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
0673 C5       PCIN    PUSH    B
0674 0606             MVI     B,06H   ;LOW ADDR BYTE OF BIOS CONST
0676 C39306           JMP     CALER
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
0679 C5       PCREAD  PUSH    B
067A 0609             MVI     B,09H   ;LOW ADDR BYTE OF BIOS CONIN
067C C39306           JMP     CALER
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
067F 3E07     BELOUT  MVI     A,BELMK
0681 C39006           JMP     PRCHR
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
0684 3E20     SPCOUT  MVI     A,' '
0686 C39006           JMP     PRCHR
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
0689 3E08     BKSOUT  MVI     A,BKSPMK
068B C39006           JMP     PRCHR
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
068E 3E0D     CROUT   MVI     A,CRMK
              ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
0690 C5       PRCHR   PUSH    B
0691 060F             MVI     B,0FH   ;LOW ADDR BYTE OF BIOS LIST
0693 D5E5     CALER   PUSH D! PUSH H
0695 4F               MOV     C,A     ;ROUTINES EXPECT VALUE HERE
0696 2A0100           LHLD    WMSTT+1 ;H/L=BIOS WBOOT=XX03H
0699 68               MOV     L,B
069A 119F06           LXI     D,RTNBAK
069D D5               PUSH    D
069E E9               PCHL
              ; BIOS ROUTINE EXECUTES HERE
069F E1D1C1   RTNBAK  POP H!  POP D!  POP B
```

```
06A2 C9                 RET
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++
06A3 3E0D       SETDSK  MVI     A,SETBDOS
06A5 C3C306             JMP     ABDOS
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++
06A8 3E13       DELETE  MVI     A,DELETEF
06AA C3C306             JMP     ABDOS
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++
06AD 3E16       CREATE  MVI     A,CREATF
06AF C3C306             JMP     ABDOS
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++
06B2 3E15       WRITE   MVI     A,WRITEF
06B4 C3C306             JMP     ABDOS
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++
06B7 3E10       CLOSE   MVI     A,CLOSEF
06B9 C3C306             JMP     ABDOS
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++
06BC 3E0F       OPEN    MVI     A,OPENF
06BE C3C306             JMP     ABDOS
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++
06C1 3E14       READ    MVI     A,READF
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++
06C3 4F         ABDOS   MOV     C,A
06C4 115C00             LXI     D,FCB
06C7 CD0500             CALL    BDOS
06CA F5                 PUSH    A
06CB 3AC0E3             LDA     ATTB-LNLGTH
06CE FE42               CPI     'B'
06D0 C2E206             JNZ     ABDOS1
06D3 CD6606             CALL    ERASE
06D6 11E506             LXI     D,DSKERR
06D9 2189E1             LXI     H,VID+6*LNLGTH+9
06DC CD5D06             CALL    SHOSTG
06DF C30000             JMP     WMSTT
06E2 F1         ABDOS1  POP     A
06E3 B7                 ORA     A
06E4 C9                 RET
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++
06E5 50524F424CDSKERR   DB      'PROBLEM WITH DRIVE A: DISK, '
0701 5245545552         DB      'RETURNING TO CP/M',0
0713 A4A0CA     JACKSTG DB      '$' OR 80H,' ' OR 80H,'J' OR 80H
0716 C1C3CB             DB      'A' OR 80H,'C' OR 80H,'K' OR 80H
0719 D0CFD4             DB      'P' OR 80H,'O' OR 80H,'T' OR 80H
071C A0A4A000           DB      ' ' OR 80H,'$' OR 80H,' ' OR 80H,0
0720 2020202020NJAKSTG  DB      '     '
0725 2020202020NREDY    DB      '     ',0
072D A0D2C5     REDY    DB      ' ' OR 80H,'R' OR 80H,'E' OR 80H
0730 C1C4D9             DB      'A' OR 80H,'D' OR 80H,'Y' OR 80H
0733 A000               DB      ' ' OR 80H,0
0735 6A61636B70JACKFL   DB      'jackpotslot',0
0741 5445535449TSTSTG   DB      'TESTING',0
0749 2420435245KREDIT   DB      '$ CREDIT $',0
0754 2052454345RCPTSTG  DB      ' RECEIPTS ',0
075F 242057494EWINSTG   DB      '$ WINS $',0
0768 051809     EXITSTG DB      CTRL AND 'E',CTRL AND 'X',CTRL AND 'I'
076B 1400               DB      CTRL AND 'T',0
076D 1A0512     ZEROSTG DB      CTRL AND 'Z',CTRL AND 'E',CTRL AND 'R'
0770 0F00               DB      CTRL AND 'O',0
0772 140513     TSTMCH  DB      CTRL AND 'T',CTRL AND 'E',CTRL AND 'S'
0775 1400               DB      CTRL AND 'T',0
0777 100C01     PLAYSTG DB      CTRL AND 'P',CTRL AND 'L',CTRL AND 'A'
077A 1900               DB      CTRL AND 'Y',0
077C 000A0A0E14PAYTBL   DB      0,10,10,14,20,50,100,250
                        ;LEM/CHY/ORG/PLM/BEL/$$$/MEL/BAR
0784 20414C4C20PAYOFF   DB      ' ALL BARS= 250+JACKPOT '
079B 20204C4153         DB      '  LAS  VEGAS   '
07AB 4D454C4F4E         DB      'MELON+ M/BAR+M/BAR=100   '
07C4 2024242424         DB      ' $$$$+$$/BAR+$$/BAR=50 '
07DB 2020535550         DB      '  SUPER-SLOT   '
07EB 42454C4C2B         DB      'BELL+BL/BAR+BL/BAR= 20',0
0802 20504C554D         DB      ' PLUM+PL/BAR+PL/BAR=14 '
0819 2020202843         DB      '   (C) 1980    '
0829 4F52414E47         DB      'ORANGE+O/BAR+O/BAR=10    '
0842 2043484552         DB      ' CHERRY+C/BAR+C/BAR=10 '
0859 2044415645         DB      ' DAVE FRIEDMAN  '
0869 432B432B41         DB      'C+C+ANY=5   C+ ANY= 2',0
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++
087F 40         CHERRY  DB      40H
;SEE DUMP TABLE
0CB7 40         CHERRYY DB      40H
                        ;++++++++++++++++++++++++++++++++++++++++++++++++++++
0D2F =          FINAL   EQU     $
0D57 =          STKPTR  EQU     FINAL+20*2
0D57 =          TRIALS  EQU     STKPTR
0D57 =          SPINCNT EQU     STKPTR
0D58 =          REEL1S  EQU     STKPTR+1
0D59 =          REEL2S  EQU     STKPTR+2
0D5A =          REEL3S  EQU     STKPTR+3
0D5B =          REEL1P  EQU     STKPTR+4
0D5D =          REEL2P  EQU     STKPTR+6
0D5F =          REEL3P  EQU     STKPTR+8
0D61 =          VALUE   EQU     STKPTR+10
0D63 =          CREDTV  EQU     STKPTR+12
0D65 =          RANDOM  EQU     STKPTR+14
0D67 =          JACKPOT EQU     STKPTR+16
0D69 =          RECIPT  EQU     STKPTR+18
0D2F            END     END     END *   END     END     END     END
```



(continued on page 22)

(Las Vegas – continued from page 21)

A>Dump Table of Reel Pattern Graphics Coding and I.D. Coding

```
087F 40 @
0880 00 00 40 00 00 00 00 03 10 00 00 00 1F 3F 3F 3B
0890 01 1E 00 00 09 1B 1F 2D 37 1F 3F 19 08 00 26 36
08A0 36 20 04 36 36 34 20 00 40 49 12 1B 19 09 08 00
08B0 00 01 34 0C 10 04 3B 2D 2E 3B 08 06 08 02 05 01
08C0 3C 2D 1B 1E 20 00 04 22 12 36 34 24 20 00 00 00
08D0 40 12 12 13 1A 12 12 00 00 00 00 00 1F 3F 3F 3B
08E0 00 00 00 00 00 0B 3F 3F 3F 3F 19 00 00 00 04 24
08F0 24 26 34 24 24 20 00 01 49 49 49 09 09 09 09 09
0900 08 07 02 11 38 1B 18 38 12 07 38 07 06 30 38 1B
0910 18 38 3B 2E 38 04 24 24 24 24 24 24 24 24 20 00
0920 00 40 01 09 09 09 08 00 00 00 00 0B 3F 3F 3F 3F
0930 3F 3B 00 00 37 3F 3D 3F 3F 3F 3F 34 00 00 00 24
0940 24 24 24 20 00 00 00 00 00 00 00 00 00 09 09 09
0950 00 00 00 1B 1F 24 3F 3F 3F 30 00 00 0F 3F 3F 12
0960 12 16 24 00 00 00 00 00 00 00 00 00 00 00 00 00
0970 00 40 40 00 00 00 00 00 00 00 00 01 0B 1B 3F 3F
0980 3B 18 00 00 02 3E 3F 3F 3E 34 20 00 00 02 34 00
0990 00 00 00 00 00 00 00 00 00 40 01 09 09 09 08 00
09A0 00 00 0B 3F 3F 3F 3F 3F 3F 3B 00 00 37 3F 3D 3F
09B0 3F 3F 3F 34 00 00 00 24 24 24 24 20 00 00 00 00
09C0 40 00 42 1A 02 1A 00 00 00 00 00 3C 24 3C 24 3C
09D0 24 20 00 00 00 24 24 3C 24 3C 24 38 00 00 00 24
09E0 24 36 36 34 24 20 00 00 00 40 41 09 09 08 00 00
09F0 00 00 00 16 24 00 00 2D 3A 00 00 00 00 00 00 02
0A00 34 20 00 00 00 00 00 00 00 04 20 00 00 00 00 00
0A10 40 12 12 13 1A 12 12 00 00 00 00 00 1F 3F 3F 3B
0A20 00 00 00 00 00 0B 3F 3F 3F 3F 19 00 00 00 04 24
0A30 24 26 34 24 24 20 00 00 00 00 00 00 00 09 09 09
0A40 00 00 00 1B 1F 24 3F 3F 3F 30 00 00 0F 3F 3F 12
0A50 12 16 24 00 00 00 00 00 00 00 00 00 00 00 00 00
0A60 00 00 40 00 00 00 00 03 10 00 00 00 1F 3F 3F 3B
0A70 01 1E 00 00 09 1B 1F 2D 37 1F 3F 19 08 00 26 36
0A80 36 20 04 36 36 34 20 00 40 49 12 1B 19 09 08 00
0A90 00 01 34 0C 10 04 3B 2D 2E 3B 08 06 08 02 05 01
0AA0 3C 2D 1B 1E 20 00 04 22 12 36 34 24 20 00 00 00
0AB0 40 12 12 13 1A 12 12 00 00 00 00 00 1F 3F 3F 3B

0AC0 00 00 00 00 00 0B 3F 3F 3F 3F 19 00 00 00 04 24
0AD0 24 26 34 24 24 20 00 00 40 00 42 1A 02 1A 00 00
0AE0 00 00 00 3C 24 3C 24 3C 24 20 00 00 00 24 24 3C
0AF0 24 3C 24 38 00 00 00 24 24 36 36 34 24 20 00 00
0B00 00 40 40 00 00 00 00 00 00 00 00 01 0B 1B 3F 3F
0B10 3B 18 00 00 02 3E 3F 3F 3E 34 20 00 00 02 34 00
0B20 00 00 00 00 00 00 00 00 00 00 01 09 09 08 00 00
0B30 00 00 00 16 24 00 00 2D 3A 00 00 00 00 00 00 02
0B40 34 20 00 00 00 00 00 00 00 04 20 00 00 00 00 00
0B50 00 40 01 09 09 09 08 00 00 00 0B 3F 3F 3F 3F 3F
0B60 3F 3B 00 00 37 3F 3D 3F 3F 3F 3F 34 00 00 00 24
0B70 24 24 24 20 00 00 00 00 00 00 40 00 00 00 00 03
0B80 10 00 00 00 1F 3F 3F 3B 01 1E 00 00 09 1B 1F 2D
0B90 37 1F 3F 19 08 00 26 36 36 20 04 36 36 34 20 00
0BA0 00 00 00 00 00 09 09 09 00 00 00 1B 1F 24 3F 3F
0BB0 3F 30 00 00 0F 3F 3F 12 12 16 24 00 00 00 00 00
0BC0 00 00 00 00 00 00 00 00 40 12 12 13 1A 12 12 00
0BD0 00 00 00 00 1F 3F 3F 3B 00 00 00 00 00 0B 3F 3F
0BE0 3F 3F 19 00 00 00 04 24 24 26 34 24 24 20 00 01
0BF0 49 49 49 09 09 09 09 09 08 07 02 11 38 1B 18 38
0C00 12 07 38 07 06 30 38 1B 18 38 3B 2E 38 04 24 24
0C10 24 24 24 24 24 20 00 00 40 01 09 09 09 08 00
0C20 00 00 0B 3F 3F 3F 3F 3F 3F 3B 00 00 37 3F 3D 3F
0C30 3F 3F 3F 34 00 00 00 24 24 24 24 20 00 00 00 00
0C40 00 40 40 00 00 00 00 00 00 00 00 01 0B 1B 3F 3F
0C50 3B 18 00 00 02 3E 3F 3F 3E 34 20 00 00 02 34 00
0C60 00 00 00 00 00 00 00 00 00 00 00 00 00 09 09 09
0C70 00 00 00 1B 1F 24 3F 3F 3F 30 00 00 0F 3F 3F 12
0C80 12 16 24 00 00 00 00 00 00 00 00 00 00 00 00 00
0C90 00 40 01 09 09 08 00 00 00 00 00 16 24 00 00 2D
0CA0 3A 00 00 00 00 00 00 02 34 20 00 00 00 00 00 00
0CB0 00 04 20 00 00 00 00 40 00 00 40 00 00 00 00 03
0CC0 10 00 00 00 1F 3F 3F 3B 01 1E 00 00 09 1B 1F 2D
0CD0 37 1F 3F 19 08 00 26 36 36 20 04 36 36 34 20 00
0CE0 40 49 12 1B 19 09 08 00 00 01 34 0C 10 04 3B 2D
0CF0 2E 3B 08 06 08 02 05 01 3C 2D 1B 1E 20 00 04 22
0D00 12 36 34 24 20 00 00 00 40 12 12 13 1A 12 12 00
0D10 00 00 00 00 1F 3F 3F 3B 00 00 00 00 00 0B 3F 3F
0D20 3F 3F 19 00 00 00 04 24 24 26 34 24 24 20 00 00
```

QUICK-KEY

by Leonard E. Garcia

A NORTH STAR BASIC STATEMENT GENERATOR

QUICK-KEY is an 8080 assembly language program that generates North Star BASIC commands and statements from pre-stored text in memory by using a two key command.

The program also allows a user to enter and define any text that will be sent to the North Star BASIC interpreter (or any system program) upon hitting a special sequence of keys.

Commonly entered routines, answers to prompts, even entire routines can be entered into the program and sent to any program that calls for a character from the terminal by just hitting two keys. A user no longer needs to enter the same information again and again.

As written, the QUICK-KEY program contains commands and statements which are used by the North Star BASIC, but the text can be changed by the user interactively and you can make copies of the program containing text for an assembler, text editor, user program, etc.

The text is stored in blocks of 32 characters, and the statements or text are accessed by hitting the sequence:

ESC x

where ESC is the ESCAPE key and the x is a letter key in the range of A thru Z.

Once the program is executed in high memory, it will locate the address of the user's Character-In routine in the North Star DOS, patch it into its own routines, re-route the Character-In "calls" to itself and then return to the Disk Operating System. The program will then scan all keyboard entries for the ESC x sequence code to send the text to whatever program called for a character from the terminal.

Thus, the sequence:	EscA	
will send the text:	ABS	(Absolute)

to the BASIC interpreter, and display on the screen as it is being echoed by the system. A second ESC will cause the first text, ABS, to be rubbed out and the next statement that starts with "A" to be sent:

APPEND

Continuing to hit the ESC key will send the next command of the "A" series, deleting the last that was sent. Once a space or another character other than ESC is entered, the key code is then voided and another ESC x sequence may be started to generate a different text.

For example, the statement

100 IF A1 = 0 THEN GOTO 300 ELSE PRINT "TEST"

may be generated with the sequence:

100 EscI A1 = 0 EscT EscGEsc 300 EscE EscP "TEST"

Note that EscG will generate the text "GOSUB". Since the statement we wanted was "GOTO", another ESC was entered to rub out the "GOSUB" and send the "GOTO" instead.

Should you wish to use spaces to indent a program, the sequence:

Esc (Space bar)

will send 5 blank spaces to the BASIC interpreter. Another immediate ESC will rub out the first 5 and send 10, and a third will rub the last 10 and send 15 spaces. The ESC (Space bar) will in fact act as tabs to enter a formatted program listing.

CUSTOMIZING.

Keys H, J, Q, U, X, Y and Z do not have any corresponding BASIC statement or command stored under them and are available to the user for any custom text. QUICK-KEY will accept, in its customizing mode, text to be stored and defined under ANY key. This text may exceed the 32 characters/key that is allowed, but the key code for the next block of text will be overlaid and cannot be used.

When entering the customizing mode, hit "C" when the program asks for a command. It will then ask: "FOR KEY LETTER (A THRU Z)?", enter the letter and a format will appear as a guideline to entering the text:

---------- 31 char ----------

If the text = NOW IS THE TIME FOR ALL GOOD MEN TO WRITE PGMS (Ctl-C) is entered under key code "X", then hitting EscX later will send the entire text to the interpreter; while hitting EscY will only send the portion "TO WRITE PGMS", since it starts at the 33rd character. This feature may be used to send a name and address to a program or address only, etc.

Leonard E. Garcia – 3517 Herschel Ave., Dallas, TX 75219

Another method of customizing text is by use of delimiters. The semi-colon (;) is used to signify to the program the end of characters. Thus an entry formatted as:

NOW IS THE;TIME FOR ALL;GOOD MEN (Ctrl-C)

under a key code letter "Z", will send text as follows:

EscZ	=	NOW IS THE	
Esc	=	TIME FOR ALL	(Rubbing out the "NOW IS THE")
Esc	=	GOOD MEN	(Rubbing out the "TIME FOR ALL")

When customizing text in this mode, the last character you enter MUST BE A CONTROL-C. This will indicate to the program then end of text for that key letter.

More than one line of text is allowed, with the carriage return being sent along with the text to the BASIC interpreter. Thus the custom text:

```
1000 REM---COMMON USE ROUTINE---
1010 FOR A = 1 TO 10
1020 PRINT " "
1030  NEXT A
1040  RETURN
(Ctrl-C)
```

may be stored in memory under any desired key A thru Z. Of course, long text will overlay the existing text, but it is easy enough to have various copies of the QUICK-KEY program with different text.

THE PROGRAM.

. . . is assembled at 9A00H. This is the highest in memory that it can be assembled for a 32K North Star system with memory starting at the standard 2000H. The customizing routines, commands, etc., reside in the lower part of the program and can be overlaid by a user program once the QUICK-KEY is operating. However, since NS BASIC places temporary data at the highest address for which it has been set, it must be "MEMSET" after it has been loaded in memory to an address no higher than 39880 (decimal). This will prevent it from overlaying the functional part of the program. This reminder will be displayed by the program.

After any text has been entered into memory in the customizing mode, and ended with a CTRL-C, the program will return to the command mode. Once the patch is effected with the "E", (EXECUTE) command, it will re-route the user's I/O routines and cannot be jumped to or customized again unless the I/O has been restored to it's original configuration.

This is done with the sequence Esc0. This command will restore the I/O and then the QUICK-KEY program will be inoperative, no longer scanning the keyboard inputs.

So, the normal procedure is to:

1. With your North Star DOS loaded, execute the program from disk.
2. If any customizing is required hit "C".
3. If none is desired, or if you are thru customizing, hit "E" to patch the I/O.
4. After the program has returned to DOS, execute your BASIC interpreter and limit the memory it can use with the "MEMSET" command.
5. Use the program, when done hit Esc0 to restore the I/O.

A FINAL NOTE.

Please remember that this program sends data in to DOS, BASIC or any other program that uses the North Star DOS for input. Commands such as LOAD, DESTROY, NSAVE, etc., are sent by the program just as if they had been typed in at the keyboard and MUST be deleted manually if incorrect! □

listing

```
                *    QUICK-KEY
                *
                * ROUTINE TO WRITE NS BASIC
                * PROGRAMS IN SHORTHAND  BY
                * USING PRE-STORED TEXT AND
                * STATEMENTS IN HIGH MEMORY.
                *
                * BY: LEONARD E. GARCIA
                *     3517 HERSCHEL AVE.
                *     DALLAS, TX 75219
                *
                * EQUATES:
2028H           DOS     EQU     2028H           DOS RE-ENTRY POINT
2010H           CIN     EQU     2010H           DOS CHAR-IN JUMP ADR
2000H           COUT    EQU     2000H           DOS CHAR-OUT JUMP
000D            CR      EQU     0DH             CARRIAGE RETURN
000A            LF      EQU     0AH             LINE FEED
001B            ESC     EQU     01BH            ESCAPE
0001            ETX     EQU     01H             END-OF-TEXT
9A00            QUICK   ORG     9A00H
                *
9A00 CD A3 9A           CALL    MSSG0           PRINT HERALD
9A03 CD A9 9A   START   CALL    MSSG1           "CUST OR EXECUTE?"
9A06 AF                 XRA     A               CLEAR A
9A07 CD 10 20           CALL    CIN             GET REPLY
9A0A FE 43              CPI     'C'             CUSTOMIZE?
9A0C CA 1C 9A           JZ      CUST
9A0F FE 45              CPI     'E'             EXECUTE?
9A11 CA 86 9A           JZ      PATCH
9A14 FE 03              CPI     03H             CTRL-C?
9A16 CA 28 20           JZ      DOS             LOOP
9A19 C3 03 9A           JMP     START
                * THIS ROUTINE ACCEPTS CUST TEXT
                *
9A1C CD B5 9A   CUST    CALL    MSSG3           "FOR KEY (A-Z)?"
9A1F 21 00 00           LXI     H,0000H
9A22 11 A1 9C           LXI     D,TEXT
9A25 AF                 XRA     A
9A26 CD 10 20           CALL    CIN             GET ANSWER
9A29 47                 MOV     B,A
9A2A CD D6 9A           CALL    CHOUT           ECHO
9A2D FE 41              CPI     041H            LESS THAN 'A'?
9A2F DA 1C 9A           JC      CUST
9A32 FE 5B              CPI     05BH            LESS THAN 'Z'?
9A34 D2 1C 9A           JNC     CUST
9A37 D6 40              SUI     40H             MULTIPLIER
9A39 6F                 MOV     L,A             LSB
9A3A 29                 DAD     H
9A3B 29                 DAD     H
9A3C 29                 DAD     H
9A3D 29                 DAD     H
9A3E 29                 DAD     H
9A3F 19                 DAD     D
9A40 22 9C 9C           SHLD    ADDR            ADDR+OFFSET
9A43 CD BB 9A   CUST2   CALL    MSSG4           BASE ADDR
9A46 2A 9C 9C           LHLD    ADDR            "ENTER TEXT"
9A49 AF                 XRA     A
9A4A CD 10 20           CALL    CIN
9A4D FE 03              CPI     03H             CTRL-C?
9A4F CA 7D 9A           JZ      THRU
9A52 FE 5F              CPI     5FH             "RUB"?
9A54 C2 6A 9A           JNZ     CONT            BACKSPC ROUTINE
9A57 06 08              MVI     B,08H
9A59 CD D6 9A           CALL    CHOUT
9A5C 06 20              MVI     B,020H
```

```
9A5E CD D6 9A          0680        CALL  CHOUT
9A61 06 08             0690        MVI   B,08H
9A63 CD D6 9A          0700        CALL  CHOUT
9A66 2B                0710        DCX   H               DECR ADDR
9A67 C3 49 9A          0720        JMP   ACPT            GET NEW CHR
9A6A                   0730 * SAVE CHARACTER TO MEMORY
9A6A 77                0740 CONT   MOV   M,A             TO RAM
9A6B 47                0750        MOV   B,A
9A6C CD D6 9A          0760        CALL  CHOUT           ECHO IT
9A6F FE 0D             0770        CPI   CR              CR?
9A71 C2 79 9A          0780        JNZ   CONT2
9A74 06 0A             0790        MVI   B,LF            LINE FEED
9A76 CD D6 9A          0800        CALL  CHOUT
9A79 23                0810 CONT2  INX   H               INCR ADDR
9A7A C3 49 9A          0820        JMP   ACPT            GET NEXT CHR
9A7D                   0830 * TERMINATE ENTRY WITH ETX CODE
9A7D 3E 01             0840 THRU   MVI   A,ETX
9A7F 77                0850        MOV   M,A
9A80 CD CB 9A          0860        CALL  CRLF
9A83 C3 03 9A          0870        JMP   START           TO BEGINNING
9A86                   0880 *
9A86                   0890 * THIS ROUTINE PATCHES THE DOS CHARACTER-IN
9A86                   0900 * TO CALL THIS PROGRAM'S I/O ROUTINES.
9A86                   0910 *
9A86 CD AF 9A          0920 PATCH  CALL  MSSG2           "I/O PATCHED"
9A89 2A 11 20          0930        LHLD  CIN+1           USER'S CIN ADDR
9A8C 22 96 9C          0940        SHLD  CHIN+2          PGM'S CHIN ADDR
9A8F 21 CD 9B          0950        LXI   H,INPUT         NEW INPUT ROUTINE
9A92 22 11 20          0960        SHLD  CIN+1           PATCH IN JUMP TABLE
9A95 3E 00             0970        MVI   A,00H           SET FLAGS FALSE
9A97 32 99 9C          0980        STA   NEXC            FOR MORE CHARACTERS
9A9A 32 9A 9C          0990        STA   RUB             AND RUBBING FLAG
9A9D 32 A0 9C          1000        STA   COUNT           AND CHR COUNTER
9AA0 C3 28 20          1010        JMP   DOS             PGM PATCH,RET TO DOS
9AA3                   1020 *
9AA3                   1030 * THIS IS THE SYSTEM MESSAGES ROUTINE
9AA3                   1040 *
9AA3 21 DB 9A          1050 MSSG0  LXI   H,TEXT0
9AA6 C3 BE 9A          1060        JMP   PRINT
9AA9 21 0C 9B          1070 MSSG1  LXI   H,TEXT1
9AAC C3 BE 9A          1080        JMP   PRINT
9AAF 21 32 9B          1090 MSSG2  LXI   H,TEXT2
9AB2 C3 BE 9A          1100        JMP   PRINT
9AB5 21 77 9B          1110 MSSG3  LXI   H,TEXT3
9AB8 C3 BE 9A          1120        JMP   PRINT
9ABB 21 93 9B          1130 MSSG4  LXI   H,TEXT4
9ABE CD CB 9A          1140 PRINT  CALL  CRLF
9AC1 46                1150 PRIN2  MOV   B,M
9AC2 CD D6 9A          1160        CALL  CHOUT           SEND A CHR
9AC5 23                1170        INX   H
9AC6 FE 01             1180        CPI   ETX             END OF TEXT?
9AC8 C2 C1 9A          1190        JNZ   PRIN2           NO?,LOOP
9ACB 06 0D             1200 CRLF   MVI   B,CR
9ACD CD D6 9A          1210        CALL  CHOUT
9AD0 06 0A             1220        MVI   B,LF
9AD2 CD D6 9A          1230        CALL  CHOUT
9AD5 C9                1240        RET                   RETURN
9AD6 AF                1250 CHOUT  XRA   A               CLEAR ACCUM
9AD7 CD 0D 20          1260        CALL  COUT            CALL DOS COUT
9ADA C9                1270        RET                   RETURN
9ADB                   1280 *
9ADB                   1290 * THESE ARE THE SYSTEM MESSAGES
9ADB                   1300 *
9ADB 20 20 20 20 2A    1310 TEXT0  DB    '    * QUICK-KEY *'
     20 51 55 49 43
     4B 2D 4B 45 59
     20 2A
9AEC 0D                1320        DB    CR
9AED 0A                1330        DB    LF
9AEE 4E 53 20 42 41    1340        DB    'NS BASIC STATEMENT GENERATOR'
     53 49 43 20 53
     54 41 54 45 4D
     45 4E 54 20 47
     45 4E 45 52 41
     54 4F 52
9B0A 0A                1350        DB    LF
9B0B 01                1360        DB    ETX
9B0C 43 55 53 54 4F    1370 TEXT1  DB    'CUSTOMIZE (C) OR EXECUTE (E) PROG
?'
     4D 49 5A 45 20
     28 43 29 20 4F
     52 20 45 58 45
     43 55 54 45 20
     28 45 29 20 50
     52 4F 47 52 41
     4D 3F
9B31 01                1380        DB    ETX
9B32 49 2F 4F 20 50    1390 TEXT2  DB    'I/O PATCHED."MEMSET" BASIC NO HIGH
 THAN 39880'
     41 54 43 48 45
     44 2E 22 4D 45
     4D 53 45 54 22
     20 42 41 53 49
     43 20 4E 4F 20
     48 49 47 48 45
     52 20 54 48 41
     4E 20 33 39 38
     38 30
9B61 0D                1400        DB    CR
9B62 0A                1410        DB    LF
9B63 52 45 54 55 52    1420        DB    'RETURNING TO DOS...'
     4E 49 4E 47 20
     54 4F 20 44 4F
     53 2E 2E 2E
9B76 01                1430        DB    ETX
9B77 46 4F 52 20 4B    1440 TEXT3  DB    'FOR KEY LETTER (A THRU Z)?:'
     45 59 20 4C 45
     54 54 45 52 20
     28 41 20 54 48
     52 55 20 5A 29
     3F 3A
9B92 01                1450        DB    ETX
9B93 20 20 20 20 20    1460 TEXT4  DB    '        ENTER TEXT BELOW:'
     20 20 45 4E 54
     45 52 20 54 45
     58 54 20 42 45
     4C 4F 57 3A
9BAB 0D                1470        DB    CR
9BAC 0A                1480        DB    LF
9BAD 2D 2D 2D 2D 2D    1490        DB    '------------31 CHAR------------'
     2D 2D 2D 2D 2D
     2D 2D 33 31 20
     43 48 41 52 2D
     2D 2D 2D 2D 2D
     2D 2D 2D 2D 2D
     2D
9BCC 01                1500        DB    ETX
9BCD                   1510 *
9BCD                   1520 * THIS IS THE FUNCTIONAL PART OF THE PGM
9BCD                   1530 *    (DOS CHIN CALLS NOW JUMP HERE)
9BCD                   1540 *
9BCD E5                1550 INPUT  PUSH  H               SAVE USER'S REGISTERS
9BCE D5                1560        PUSH  D
9BCF C5                1570        PUSH  B
9BD0 F5                1580        PUSH  PSW
9BD1 3A 99 9C          1590        LDA   NEXC            CHK IF MORE CHAR
9BD4 FE FF             1600        CPI   0FFH            MORE CHAR TO PRINT?
9BD6 CA 4D 9C          1610        JZ    SEND            YES,KEEP SENDING
9BD9 3A 9A 9C          1620        LDA   RUB             CHK IF STILL ERASING
9BDC FE FF             1630        CPI   0FFH            MORE CHAR TO "RUB"?
9BDE CA 26 9C          1640        JZ    NEXT            YES,GO TO IT
9BE1 CD 94 9C          1650 INCHR  CALL  CHIN            NO,GET CHR
9BE4 47                1660        MOV   B,A             SAVE THE CHARACTER
9BE5 FE 1B             1670        CPI   ESC             IS IT "ESCAPE"?
9BE7 CA F5 9B          1680        JZ    PROC            YES,SKIP
9BEA 3E 00             1690 NPROC  MVI   A,00H           FLAG FALSE
9BEC 32 9B 9C          1700        STA   KEY             STORE IT
9BEF 32 A0 9C          1710        STA   COUNT           ZERO CHAR COUNTER
9BF2 C3 81 9C          1720        JMP   RSTOR           NO,GO RETURN
9BF5                   1730 *
9BF5                   1740 * THIS ROUTINE PROCESSES THE ESC SEQUENCE
9BF5                   1750 *
9BF5 3A 9B 9C          1760 PROC   LDA   KEY             CHK KEY FLAG
9BF8 FE FF             1770        CPI   0FFH            KEY ALREADY HIT?
9BFA CA 26 9C          1780        JZ    NEXT            YES,MUST WANT NEXT
9BFD 21 00 00          1790        LXI   H,0000H
9C00 11 A1 9C          1800        LXI   D,TEXT
9C03 CD 94 9C          1810        CALL  CHIN            NO,GET KEY CHAR
```

(continued)

```
9C06 FE 30          1820          CPI   '0'           ASCII 0?
9C08 CA 87 9C       1830          JZ    RPATC         YES,GO REPATCH DOS
9C0B FE 20          1840          CPI   020H          SPACE BAR?
9C0D CA 22 9C       1850          JZ    SPCS
9C10 FE 41          1860          CPI   041H          LESS THAN 'A'?
9C12 DA E1 9B       1870          JC    INCHR
9C15 FE 5B          1880          CPI   05BH          LESS THAN '['?
9C17 D2 E1 9B       1890          JNC   INCHR
9C1A D6 40          1900 LETTR    SUI   040H          MULTIPLIER
9C1C 6F             1910          MOV   L,A           LSB
9C1D 29             1920          DAD   H
9C1E 29             1930          DAD   H
9C1F 29             1940          DAD   H
9C20 29             1950          DAD   H
9C21 29             1960          DAD   H
9C22 19             1970 SPCS     DAD   D             ADDR+OFFSET
9C23 C3 42 9C       1980          JMP   SEND0
9C26                1990 *
9C26                2000 * THIS ROUTINE RUBS OUT STATEMENT,SENDS NEXT
9C26                2010 *
9C26 3E FF          2020 NEXT     MVI   A,0FFH        SET RUB FLAG
9C28 32 9A 9C       2030          STA   RUB           TRUE,ERASING
9C2B 3A A0 9C       2040          LDA   COUNT         LOAD COUNT
9C2E 3D             2050          DCR   A             DECREMENT
9C2F 32 A0 9C       2060          STA   COUNT         RE-STORE
9C32 C2 3D 9C       2070          JNZ   NEXT2
9C35 3E 00          2080          MVI   A,00H         SET FLAG FALSE
9C37 32 9A 9C       2090          STA   RUB           ALL DONE
9C3A C3 4D 9C       2100          JMP   SEND          NEXT STATEMENT
9C3D 06 5F          2110 NEXT2    MVI   B,05FH        ASCII "RUB"
9C3F C3 81 9C       2120          JMP   RSTOR         RETURN W/RUB
9C42                2130 *
9C42                2140 * THIS ROUTINE OUTPUTS THE TEXT
9C42                2150 *
9C42 22 9C 9C       2160 SEND0    SHLD  ADDR          BASE ADDR
9C45 22 9E 9C       2170          SHLD  POINT         BASE POINTER
9C48 3E FF          2180          MVI   A,0FFH        FLAG TRUE
9C4A 32 9B 9C       2190          STA   KEY           FOR KEY &
9C4D 3E FF          2200 SEND     MVI   A,0FFH        TRUE
9C4F 32 99 9C       2210          STA   NEXC          FOR MORE CHARACTERS
9C52 21 A0 9C       2220          LXI   H,COUNT       CHAR COUNTER
9C55 4E             2230          MOV   C,M           COUNT TO C
9C56 2A 9E 9C       2240          LHLD  POINT         CURRENT ADDR POINTER
9C59 46             2250          MOV   B,M           CHAR TO B
9C5A 23             2260          INX   H             CHK NEXT CHR
9C5B 7E             2270          MOV   A,M           TO A
9C5C FE 3B          2280 NEXCK    CPI   ';'           DELIMITER?
9C5E C2 6B 9C       2290          JNZ   ENDCK         NO?,CHK FOR END
9C61 3E 00          2300          MVI   A,00H         ZERO FLAG
9C63 32 99 9C       2310          STA   NEXC          NEXT CHAR FLAG
9C66 23             2320          INX   H             SKIP OVER THE ';'
9C67 0C             2330          INR   C             INCR COUNT
9C68 C3 79 9C       2340          JMP   RSET
9C6B FE 01          2350 ENDCK    CPI   ETX           END OF STATEMENTS?
9C6D C2 79 9C       2360          JNZ   RSET          NO?,SKIP
9C70 2A 9C 9C       2370          LHLD  ADDR          RE-LOAD BASE ADDR
9C73 3E 00          2380          MVI   A,00H         ZERO FLAG
9C75 32 99 9C       2390          STA   NEXC          NO NEXT CHAR
9C78 0C             2400          INR   C             INCR COUNT
9C79 22 9E 9C       2410 RSET     SHLD  POINT         NEW ADDR
9C7C 0C             2420          INR   C
9C7D 79             2430          MOV   A,C
9C7E 32 A0 9C       2440          STA   COUNT
9C81                2450 *
9C81                2460 * THIS ROUTINE RETURNS W/THE CHARACTER
9C81                2470 *
9C81 F1             2480 RSTOR    POP   PSW           RESTORE REGISTER
9C82 78             2490          MOV   A,B           CHARACTER TO A
9C83 C1             2500          POP   B
9C84 D1             2510          POP   D
9C85 E1             2520          POP   H
9C86 C9             2530          RET                 RETURN FROM CHIN CALL
9C87                2540 *
9C87                2550 * THIS ROUTINE RE-PATCHES THE DOS I/O
9C87                2560 *
9C87 2A 96 9C       2570 RPATC    LHLD  CHIN+2        USER'S CHIN ADDR
9C8A 22 11 20       2580          SHLD  CIN+1         BACK IN DOS JUMP
9C8D F1             2590          POP   PSW
9C8E C1                           POP   B             RESTORE USER'S REG
9C8F D1             2610          POP   D
9C90 E1             2620          POP   H
9C91 C3 10 20       2630          JMP   CIN           RETURN TO DOS CALL
9C94                2640 *
9C94                2650 * THIS IS THE NEW CHIN ROUTINE
9C94                2660 *
9C94 AF             2670 CHIN     XRA   A
9C95 CD 00 00       2680          CALL  0000H         DUMMY ADDRESS
9C98 C9             2690          RET
9C99                2700 *
9C99                2710 * THIS IS THE TEMPORARY DATA AREA
9C99                2720 *
9C99                2730 NEXC     DS    1             MORE CHARACTERS FLAG
9C9A                2740 RUB      DS    1             MORE CHAR TO RUB FLAG
9C9B                2750 KEY      DS    1             KEY LETTER HIT
9C9C                2760 ADDR     DS    2             BASE TEXT ADDRESS
9C9E                2770 POINT    DS    2             POINTER ADDRESS
9CA0                2780 COUNT    DS    1             NO.OF CHAR PRINTED
9CA1                2790 * THIS IS THE TEXT AREA
9CA1 20 20 20 20 20 2800 TEXT     DB    '     ;       ;
     3B 20 20 20 20
     20 20 20 20 20
     20 3B 20 20 20
     20 20 20 20 20
     20 20 20 20 20
     20 20
9CC1 01             2810          DB    ETX
9CC2                2820 TEXTA    ORG   TEXT+32
9CC1 41 42 53 3B 41 2830          DB    'ABS;APPEND;ASC;ATN;AUTO'
     50 50 45 4E 44
     3B 41 53 43 3B
     41 54 4E 3B 41
     55 54 4F
9CD8 01             2840          DB    ETX
9CD9                2850 TEXTB    ORG   TEXTA+32
9CE1 42 59 45       2860          DB    'BYE'
9CE4 01             2870          DB    ETX
9CE5                2880 TEXTC    ORG   TEXTB+32
9D01 43 41 4C 4C 3B 2890          DB    'CALL;CHAIN;CHR$;CLOSE;CONT;COS'
     43 48 41 49 4E
     3B 43 48 52 24
     3B 43 4C 4F 53
     45 3B 43 4F 4E
     54 3B 43 4F 53
9D1F 01             2900          DB    ETX
9D20                2910 TEXTD    ORG   TEXTC+32
9D21 44 41 54 41 3B 2920          DB    'DATA;DEF;DEL;DESTROY;DIM'
     44 45 46 3B 44
     45 4C 3B 44 45
     53 54 52 4F 59
     3B 44 49 4D
9D39 01             2930          DB    ETX
9D3A                2940 TEXTE    ORG   TEXTD+32
9D41 45 4C 53 45 3B 2950          DB    'ELSE;EXIT;EXAM;EXP;ERRSET;END'
     45 58 49 54 3B
     45 58 41 4D 3B
     45 58 50 3B 45
     52 52 53 45 54
     3B 45 4E 44
9D5E 01             2960          DB    ETX
9D5F                2970 TEXTF    ORG   TEXTE+32
9D61 46 4F 52 3B 46 2980          DB    'FOR;FILE;FNEND;FILL;FREE'
     49 4C 45 3B 46
     4E 45 4E 44 3B
     46 49 4C 4C 3B
     46 52 45 45
9D79 01             2990          DB    ETX
9D7A                3000 TEXTG    ORG   TEXTF+32
9D81 47 4F 53 55 42 3010          DB    'GOSUB;GOTO'
     3B 47 4F 54 4F
9D8B 01             3020          DB    ETX
9D8C                3030 TEXTH    ORG   TEXTG+32
9DA1 48 20 49 53 20 3040          DB    'H IS BLANK, ERASE THIS LINE'
     42 4C 41 4E 4B
     2C 20 45 52 41
     53 45 20 54 48
     49 53 20 4C 49
```

```
                4E 45
9DBC 01                         3050          DB    ETX
9DBD                            3060 TEXTI    ORG   TEXTH+32
9DC1 49 46 3B 49 4E             3070          DB    'IF;INPUT;INT;INP;INCHAR$'
     50 55 54 3B 49
     4E 54 3B 49 4E
     50 3B 49 4E 43
     48 41 52 24
9DD9 01                         3080          DB    ETX
9DDA                            3090 TEXTJ    ORG   TEXTI+32
9DE1 4A 20 49 53 20             3100          DB    'J IS BLANK, ERASE THIS LINE'
     42 4C 41 4E 4B
     2C 20 45 52 41
     53 45 20 54 48
     49 53 20 4C 49
     4E 45
9DFC 01                         3110          DB    ETX
9DFD                            3120 TEXTK    ORG   TEXTJ+32
9E01 4B 20 49 53 20             3130          DB    'K IS BLANK, ERASE THIS LINE'
     42 4C 41 4E 4B
     2C 20 45 52 41
     53 45 20 54 48
     49 53 20 4C 49
     4E 45
9E1C 01                         3140          DB    ETX
9E1D                            3150 TEXTL    ORG   TEXTK+32
9E21 4C 49 53 54 3B             3160          DB    'LIST;LOAD;LET;LINE;LOG'
     4C 4F 41 44 3B
     4C 45 54 3B 4C
     49 4E 45 3B 4C
     4F 47
9E37 01                         3170          DB    ETX
9E38                            3180 TEXTM    ORG   TEXTL+32
9E41 4D 45 4D 53 45             3190          DB    'MEMSET'
     54
9E47 01                         3200          DB    ETX
9E48                            3210 TEXTN    ORG   TEXTM+32
9E61 4E 45 58 54 3B             3220          DB    'NEXT;NSAVE;NOENDMARK'
     4E 53 41 56 45
     3B 4E 4F 45 4E
     44 4D 41 52 4B
9E75 01                         3230          DB    ETX
9E76                            3240 TEXTO    ORG   TEXTN+32
9E81 4F 4E 3B 4F 50             3250          DB    'ON;OPEN;OUT'
     45 4E 3B 4F 55
     54
9E8C 01                         3260          DB    ETX
9E8D                            3270 TEXTP    ORG   TEXTO+32
9EA1 50 52 49 4E 54             3280          DB    'PRINT;PSIZE'
     3B 50 53 49 5A
     45
9EAC 01                         3290          DB    ETX
9EAD                            3300 TEXTQ    ORG   TEXTP+32
9EC1 51 20 49 53 20             3310          DB    'Q IS BLANK,ERASE THIS LINE'
     42 4C 41 4E 4B
     2C 45 52 41 53
     45 20 54 48 49
     53 20 4C 49 4E
     45
9EDB 01                         3320          DB    ETX
9EDC                            3330 TEXTR    ORG   TEXTQ+32
9EE1 52 45 4D 3B 52             3340          DB    'REM;RETURN;RESTORE;READ;RND'
     45 54 55 52 4E
     3B 52 45 53 54
     4F 52 45 3B 52
     45 41 44 3B 52
     4E 44
9EFC 01                         3350          DB    ETX
9EFD                            3360 TEXTS    ORG   TEXTR+32
9F01 53 41 56 45 3B             3370          DB    'SAVE;SQRT;STOP;SIN;SGN;STR$'
     53 51 52 54 3B
     53 54 4F 50 3B
     53 49 4E 3B 53
     47 4E 3B 53 54
     52 24
9F1C 01                         3380          DB    ETX
9F1D                            3390 TEXTT    ORG   TEXTS+32
9F21 54 48 45 4E 3B             3400          DB    'THEN;TAB;TYP'
     54 41 42 3B 54
     59 50
9F2D 01                         3410          DB    ETX
9F2E                            3420 TEXTU    ORG   TEXTT+32
9F41 55 20 49 53 20             3430          DB    'U IS BLANK,ERASE THIS LINE'
     42 4C 41 4E 4B
     2C 45 52 41 53
     45 20 54 48 49
     53 20 4C 49 4E
     45
9F5B 01                         3440          DB    ETX
9F5C                            3450 TEXTV    ORG   TEXTU+32
9F61 56 41 4C                   3460          DB    'VAL'
9F64 01                         3470          DB    ETX
9F65                            3480 TEXTW    ORG   TEXTV+32
9F81 57 52 49 54 45             3490          DB    'WRITE#'
     23
9F87 01                         3500          DB    ETX
9F88                            3510 TEXTX    ORG   TEXTW+32
9FA1 58 20 49 53 20             3520          DB    'X IS BLANK,ERASE THIS LINE'
     42 4C 41 4E 4B
     2C 45 52 41 53
     45 20 54 48 49
     53 20 4C 49 4E
     45
9FBB 01                         3530          DB    ETX
9FBC                            3540 TEXTY    ORG   TEXTX+32
9FC1 59 20 49 53 20             3550          DB    'Y IS BLANK,ERASE THIS LINE'
     42 4C 41 4E 4B
     2C 45 52 41 53
     45 20 54 48 49
     53 20 4C 49 4E
     45
9FDB 01                         3560          DB    ETX
9FDC                            3570 TEXTZ    ORG   TEXTY+32
9FE1 5A 20 49 53 20             3580          DB    'Z IS BLANK,ERASE THIS LINE'
     42 4C 41 4E 4B
     2C 45 52 41 53
     45 20 54 48 49
     53 20 4C 49 4E
     45
9FFB 01                         3590          DB    ETX
9FFC                            3600 *
9FFC                            3610          END
```

end

symbol table

```
ACPT  9A49   ADDR  9C9C   CHIN  9C94   CHOUT 9AD6   CIN   2010   CONT  9A6A
CONT2 9A79   COUNT 9CA0   COUT  200D   CR    000D   CRLF  9ACB   CUST  9A1C
CUST2 9A43   DOS   2028   ENDCK 9C6B   ESC   001B   ETX   0001   INCHR 9BE1
INPUT 9BCD   KEY   9C9B   LETTR 9C1A   LF    000A   MSSG0 9AA3   MSSG1 9AA9
MSSG2 9AAF   MSSG3 9AB5   MSSG4 9ABB   NEXC  9C99   NEXCK 9C5C   NEXT  9C26
NEXT2 9C3D   NPROC 9BEA   PATCH 9A86   POINT 9C9E   PRIN2 9AC1   PRINT 9ABE
PROC  9BF5   QUICK 9A00   RPATC 9C87   RSET  9C79   RSTOR 9C81   RUB   9C9A
SEND  9C4D   SEND0 9C42   SPCS  9C22   START 9A03   TEXT  9CA1   TEXT0 9ADB
TEXT1 9B0C   TEXT2 9B32   TEXT3 9B77   TEXT4 9B93   TEXTA 9CC1   TEXTB 9CE1
TEXTC 9D01   TEXTD 9D21   TEXTE 9D41   TEXTF 9D61   TEXTG 9D81   TEXTH 9DA1
TEXTI 9DC1   TEXTJ 9DE1   TEXTK 9E01   TEXTL 9E21   TEXTM 9E41   TEXTN 9E61
TEXTO 9E81   TEXTP 9EA1   TEXTQ 9EC1   TEXTR 9EE1   TEXTS 9F01   TEXTT 9F21
TEXTU 9F41   TEXTV 9F61   TEXTW 9F81   TEXTX 9FA1   TEXTY 9FC1   TEXTZ 9FE1
THRU  9A7D
```

FRENCH

```
TRES BIEN!                          BICYCLE=LE VELO
MAIS NON!                           SUBWAY=LE METRO
HOUSE=LA MAISON                     ROOM=LA CHAMBRE
RABBIT=LE LAPIN                     HELICOPTER=L' HELICOPTERE
FLOWER=LA FLEUR                     MUSHROOM=LE CHAMPIGNON
SNAIL=L' ESCARGOT                   WINE=LE VIN
TRAIN=LE TRAIN                      CHEESE=LE FROMAGE
```

LISTING

```
0001          PROGRAM TUTOR
      C
      C       THIS IS A PROGRAM TO PROMPT THE STUDENT TO SPELL WORDS IN A
      C       LANGUAGE WHEN GIVEN A WORD IN A DIFFERENT LANGUAGE.
      C
      C       BY: GARY D. GAUGLER   5706 SEASCAPE. CITRUS HEIGHTS, CA 95610
      C
0002          IMPLICIT INTEGER (A-Z)
      C
0003          COMMON      LINE(80),KOL,MXKOL
      C
0004          DIMENSION LANG1(20),LANG2(20),ERROR(20),IANS(20)
0005          DIMENSION LANG(20),JPROMP(5),CORECT(20),NCOREC(20)
      C
0006          DATA CARAT,SPACE / 1H^, 1H /
0007          DATA JPROMP /2H W,2HOR,2HD ,2HFO,2HR /
      C
      C.... FETCH OUR LANGUAGE FILE IF IT EXISTS
0008          OPEN (UNIT=2,NAME='DK1:FRENCH.DAT',TYPE='OLD',
             1      ACCESS='SEQUENTIAL')
      C
0009          DIRECT=0     !0=ENGLISH-->FOREN
0010          MXTRY=2
      C
0011          TYPE 10
0012       10 FORMAT (1X,'DO YOU WANT TO ENTER FOREIGN FOR ENGLISH? (Y/N)  ',$)
0013          READ (7,20)ANSWR
0014       20 FORMAT (A1)
0015          IF (ANSWR.EQ.'N') DIRECT=1          !FOREN-->ENGLISH
      C
      C.... GET FILE'S LANGUAGE BASE AND MESSAGES
0017          CALL INPUT
0018          CALL NAME(LANG,LENLAN)
0019          CALL INPUT
0020          CALL NAME(CORECT,LENOK)
0021          CALL INPUT
0022          CALL NAME(NCOREC,LENBAD)
      C
      C.... READ A WORD-PAIR
0023      110 CALL INPUT
0024       35 FORMAT (20A1)
      C
      C.... CLEAR-OUT THE ARRAYS
0025          DO 120 I=1,20
0026          LANG1(I)=SPACE
0027          LANG2(I)=SPACE
0028          ERROR(I)=SPACE
0029      120 IANS(I)=SPACE
      C
      C.... GET LANGUAGE WORD
0030          CALL NAME(LANG1,LEN1)
      C
      C.... NOW GET THE SECOND LANGUAGE WORD
0031          CALL NAME(LANG2,LEN2)
      C
      C.... NOW PRESENT A WORD TO THE STUDENT
0032          NTRY=0
0033      320 IF (DIRECT.EQ.1) GO TO 350  !FOREN-->ENGLISH
0035          KOL=1
0036          DO 325 J=1,LENLAN
0037          LINE(KOL)=LANG(J)
0038      325 KOL=KOL+1
0039          DO 330 J=1,5
0040          N2=JPROMP(J)
0041          N1=N2/256
0042          N2=N2-256*N1
0043          LINE(KOL)=N2
0044          LINE(KOL+1)=N1
0045      330 KOL=KOL+2
0046          DO 335 J=1,LEN1
0047          LINE(KOL)=LANG1(J)
0048      335 KOL=KOL+1
0049          DO 340 J=KOL,40
0050      340 LINE(J)=SPACE
0051          TYPE 40,(LINE(J),J=1,40)
0052       40 FORMAT (1X,'TYPE THE ',40A1)
0053          GO TO 400
```

tutor: A FOREIGN LANGUAGE VOCABULARY TRAINER

by Gary D. Gaugler

This program is designed to be the basis of a foreign language instruction system. When run on a single-user system, the user can practice, learn and build a foreign language vocabulary. When installed on a multi-user system, many students can simultaneously use the program. Further, each student user can be practicing a language different from the other users.

The program presents a foreign language word to the student and waits for the student's response. The words are retrieved from a disk file which contains words in English and their associated foreign equivalent. The file also contains a header indicating the foreign language, a character string to be displayed when a correct response is made and a string which is displayed when an incorrect response is made. The program also permits the student to practice English-foreign or foreign-English. The student replies to the program's prompt for the desired direction.

When a word is presented, the program checks each character of the student's response for correctness. For each character which does not match the correct response, an up-arrow (↑) is placed below it, indicating an incorrect character response at that location. The student has one more chance to respond before the program moves on to the next word.

The program, called TUTOR, is written in PDP-11 FORTRAN-IV for RT-11. This version of FORTRAN is very similar to FORTRAN-66 so one should have

Gary D. Gaugler – 5706 Seascape Court, Citrus Heights, CA 95610

little difficulty adapting it to any system which has a FORTRAN compiler.

Included with the program's listing is a sample language file for French. Using this as an example, other language files can be made. The first line defines the language for which the specific file is intended (the example is FRENCH). The second line will be displayed verbatim when a correct response is made. The third line will be displayed verbatim when an incorrect response is made. The remaining lines are the English-foreign word pairs organized as: English=foreign. The equals sign delimits the English and foreign words. Embedded spaces are significant and must be included for a correct response.

I would be pleased to provide a floppy copy of the program to those sending a formatted diskette (RX-01 or RX-02) and return postage and container. I also welcome communicating with others interested in PDP-11/LSI-11 hardware or software.□

Thanks go to Carol Newhouse for supplying the French vocabulary.

```
      C
0054   350  TYPE 50,(LANG2(I),I=1,LEN2)
0055    50  FORMAT (1X,'TYPE THE ENGLISH WORD FOR  ',20A1/)
      C
      C.... CHECK FOR CORRECT WORD & SPELLING
0056   400  READ (7,35)IANS
0057        JERR=0
0058        DO 425 J=1,20
0059        IF (DIRECT.EQ.1) GO TO 410
0061        IF (LANG2(J).EQ.IANS(J)) GO TO 425
0063        JERR=1
0064        ERROR(J)=CARAT
0065        GO TO 425
0066   410  IF (LANG1(J).EQ.IANS(J)) GO TO 425
0068        JERR=1
0069        ERROR(J)=CARAT
0070   425  CONTINUE
      C
      C.... REPORT IF OK OR AN ERROR
0071        IF (JERR.EQ.1) GO TO 500
0073        TYPE 60,(CORECT(J),J=1,LENOK)
0074    60  FORMAT (1X,//,1X,20A1/)
0075        GO TO 110
      C
      C.... ERROR
0076   500  TYPE 65,ERROR,(NCOREC(J),J=1,LENBAD)
0077    65  FORMAT (1X,20A1//,1X,20A1/)
0078        NTRY=NTRY+1
0079        IF (NTRY.LT.MXTRY) GO TO 320
0081        GO TO 110
0082        END
```

SUBROUTINES

```
0001        SUBROUTINE INPUT
      C
      C     THIS SUBROUTINE READS ONE LINE FROM THE DISK FILE
      C
0002        COMMON      LINE(80),KOL,MXKOL
0003        DATA IBL /1H /
      C
0004        KOL=1
      C
      C.... READ A LINE FROM THE LANGUAGE FILE
0005        READ (2,30,END=900)LINE
0006    30  FORMAT (80A1)
      C
      C.... DETERMINE THE LAST USEFUL COLUMN
0007        DO 200 I=1,80
0008        IF (LINE(I).EQ.IBL) GO TO 200
0010        MXKOL=I
0011   200  CONTINUE
0012        RETURN
      C
      C.... END OF DATA FILE
0013   900  STOP
0014        END
```

```
0001        SUBROUTINE NAME(IOUT,NCHARS)
      C
      C     THIS SUBROUTINE READS A WORD FROM THE INPUT LINE
      C     INTO THE ARRAY PASSED AS AN ARGUMENT (IOUT). THE LENGTH
      C     OF THE WORD IN CHARACTERS IS RETURNED IN 'NCHARS'.
      C
0002        COMMON      LINE(80),KOL,MXKOL
      C
0003        DIMENSION IOUT(20)
0004        DATA ICM,IEQ /1H,, 1H=/
      C
0005        NCHARS=0
0006        IKOL=1
      C
      C.... BUILD THE WORD IN 'IOUT' UNTIL TERMINATOR ENCOUNTERED
0007   100  IF (KOL.GT.MXKOL) GO TO 900
0009        IF (LINE(KOL).EQ.ICM) GO TO 900
0011        IF (LINE(KOL).EQ.IEQ) GO TO 900
0013        IOUT(IKOL)=LINE(KOL)
0014        KOL=KOL+1
0015        IKOL=IKOL+1
0016        NCHARS=NCHARS+1
0017        GO TO 100
      C
      C.... END OF WORD FIELD
0018   900  KOL=KOL+1
0019        RETURN
0020        END
```

END

Gary Gaugler is currently a computer system designer at McClellan AFB, CA. His background includes hardware and software design, analysis and construction. He was previously an IC design engineer and system architect at the Air Force Avionics Laboratory, Wright-Patterson AFB, OH. He was involved in IC development, microprocessor design and development and software engineering. His interests are IC design, computer simulation of electronic circuits and hardware design.

INTERFACING THE WH14 PRINTER
to your microcomputer's serial port is simpler than you think

by Lachman Sippy

What is wrong with this relatively inexpensive feature loaded printer? I ask this question because I have read so many articles or letters from disappointed or frustrated owners of the WH14. I will describe my own experience and tell you how to get the most out of it.

I acquired a WH14 printer at a bargain price from the recent *Microage* redistribution list. According to them, it was in perfect working order except that it was missing the RS232 cable and it was out of warranty. Since the serial port on my own system does not have handshaking,* I wired only four RS232 signals, namely RS232OUT(2), RS232IN(3), signal ground(7) and protective ground(1). Several off-the-shelf serial I/O boards do not use the handshaking signals including Morrow's Switchboard, North Star serial ports, 3P + S.

To my delight, the printer worked. However, above 300 baud, it became evident that one must use handshaking. I was determined to try all possible ways to avoid that so I tried delays after C/R, after each character output, but to no avail. It seemed that I have no choice but to use the RTS(4) or the DTR(20) RS232 signals to provide for handshaking, via one of the parallel ports on my system. On reading the manual again, I noticed that the WH14 also provided handshaking via the output side of the same serial port. Simple, when the printer is not ready for any reason it sends a CTL-S(13H), and when ready it outputs a CTL-Q(11H). These signals are evidently sent on a continuing basis at the set baud rate. I programmed a new output driver to test the input status and if a character is received it tests for a 13H. If a 13H is received I wait in a loop until an 11H is received before sending the character to the printer. With this simple driver the printer works at all the available baud rates without losing any data. As the reader will note, having this ability to do the handshaking with software via the same serial port makes the WH14 extremely versatile, and one should have no problem connecting to any microcomputer.

The WH14 operations manual indicates that it is normal for the printer to occasionally stop printing even though the printer is not out of paper, there is no paper jam, and the print head is still cool. By simply using the on-line key (releasing the key and then depressing it again) the printer is restored without losing any characters. This is true and it works with the driver that is mentioned here. However, I wish that Heathkit or Zenith would do something about it. With all the intelligence in the printer, a modification to their software in ROM, the printer should reset itself.

I should mention an additional problem that I encountered with my printer. It turned out that when my printer stopped for no apparent reason, my use of the on-line key as described in the previous paragraph had no effect. In fact, when the printer stopped, I was unable to recover even if I powered off. My diagnosis of this failure led me to believe that I had a problem with the baud rate generator on the WH14's 8250 UART. I determined this by querying the input from the WH14. What I would receive was strings of 1's and 0's in various combinations. Since I was not absolutely sure that it was a baud rate problem, I was about to take it to the local Heathkit service center. The cost to fix it would be $65.00 plus parts. The price of a replacement 8250 was approximately $14.00 from Heathkit. I decided to take a chance and bought one in spite of the fact that they said they did not take back ICs once sold, even if it was defective, because it was sure to be good. I was surprised at the way the salesman handled the MOS chip, probably out of sheer ignorance. Anyway, after I replaced the 8250 on my printer I had no further problems.

To sum up, it appears that the WH14, due to its versatility, is a good buy for the price, assuming that it proves to be reliable in the long run. The sample code for the driver follows. The code assumes serial port on the Morrow's Switchboard with the I/O base address set to 20H, and the data to be output in the C register.

```
                        STAT    =   22H
                        DATA    =   21H
                        RDA     =   40H
                        TBE     =   80H
                        NPAR    =   7FH
                        CTLS    =   13H
                        CTLQ    =   11H

0000  DB 22     DRVR:   IN      STAT        ;Get input status
0002  E6 40             ANI     RDA         ;Data avail?
0004  CA 0000           JZ      DOUT        ;No, continue
0007  DB 22     DRVI:   IN      STAT        ;Get status again
0009  E6 40             ANI     RDA         ;Data avail?
000B  CA 0400           JZ      DRV1        ;Wait
000E  DB 21             IN      DATA        ;Get the char rec'd
0010  E6 7F             ANI     NPAR        ;Strip parity, if any
0012  FE 13             CPI     CTLS        ;CTL-S?
0014  CA 0400           JZ      DRV1        ;Yes, wait till printer ready
0017  FE 11             CPI     CTLQ        ;CTL-Q?
0019  C2 0400           JNZ     DRV1        ;No, not ready yet
001C  DB 22     DOUT:   IN      STAT        ;Get status
001E  E6 80             ANI     TBE         ;Port ready for output
0020  CA 1C00           JZ      DOUT        ;No, wait
0023  79                MOV     A,C         ;Move char to A-reg
0024  D3 21             OUT     DATA        ;Output char to printer
0026  C9                RET
```

The code can easily be modified for any serial port by changing the port addresses and the status bits. The reader should also note that when the power is off on the WH14, input data on the serial port will be a 00H. This fact can be used to print a message on the console device. Also, if one looks at the status, one will see an overrun condition. This can be ignored. □

** The term "handshaking" refers to a pulse exchange between the CPU and asynchronous communicating peripheral device/s. Handshaking establishes synchronization and ensures proper operating.*

Lachman Sippy – Box 1107, 2000 Center Street, Berkeley, CA 94704.

Lachman K. Sippy is currently with the University of California in a computer planning capacity. He has worked in the past for both Boeing Airplane Company and IBM. He is also the Director of Innovative Interfaces, Inc., a Berkeley-based company which has pioneered the use of microcomputers in large libraries.

PROGRAM DOCUMENTATION

by Freeman L. Moore

Computer programming: is it merely writing a set of instructions to accomplish a specific task? I would hope we agree that writing the set of instructions can be a very involved process—working with such things as clarifying the problem statement, developing an algorithm, flowcharting, testing, debugging, documentation, and writing the code or instructions. Quite often, one considers a program completed when it produces the correct results. This may be acceptable for programs which are used only once; but for production jobs which are kept and used again, good documentation is essential.

Documentation of a computer program should consist of the problem statement, operating instructions, flowcharts and sample input and output, among other items. Some features are part of what might be referred to as a "user's manual," while others belong to a "programmer's manual." Ideally these two manuals will be kept separate since it should never be assumed that the user is capable of programming, and that the programmer tends to forget the needs of the naive user.

The user's manual should include sample input; sample output; anticipated errors, their meaning and possible corrections; and, purpose of the program. Sample input is necessary to provide the user with an idea of what the program expects, while the sample output will reinforce the user's belief in the program by having results to compare. Including a directory of errors should not be considered discouraging to the user. Such a directory lets a user know the program is capable of detecting errors without causing unknown consequences. If an unexplained error message should appear, then expand upon it in

Freeman L. Moore—Computer Science Dept., Central Michigan University, Mt. Pleasant, MI 48859

Listing

```
                        *
                        * Program EXTRACT -
                        *
                        * This program will read a BASIC program file and
                        * 'extract' all comments (REM statements).  The
                        * REM statements are written to one file, and the
                        * remaining statements are written to another file.
                        *
                        * Program Execution:      (assuming FLEX notation)
                        *
                        *   EXTRACT,<file>
                        *
                        *  The assumed extension is .BAS and the working
                        *  drive is assumed.  The output files have the
                        *  same name but extensions of .REM and .EXT.
                        *
                        *
                        * The purpose behind this is that current BASIC
                        * interpretors keep the source program in memory,
                        * which if heavily commented, can be quite large.
                        * By executing the .EXT file (extracted) from
                        * this program, the core requirements will be
                        * substantially reduced.  The user can also print
                        * the .REM file as his documentation about the
                        * program.
                        *
                        *  Written by:  Freeman L. Moore
                        *               Central Michigan University, 1980
                        *
                        *
                        * SYSTEM EQUATES
                        *
7112                    PUTCHR  EQU     $7112       OUTPUT SINGLE CHARACTER
7103                    DOS     EQU     $7103       FLEX ENTRY
7130                    ADDBX   EQU     $7130       ADD B-REG TO X-REG
7806                    FMS     EQU     $7806       FILE SYSTEM ENTRY
7133                    OUTDEC  EQU     $7133       OUTPUT NUMBER
7127                    GETFIL  EQU     $7127       PARSE FILE SPEC
712D                    SETEXT  EQU     $712D       SET DEF EXT
                        *
                        * MISC EQUATES
                        *
0004                    EOT     EQU     4
000D                    CR      EQU     $D
000A                    LF      EQU     $A
0001                    TRUE    EQU     1           (FALSE=0)
0001                    VERNO   EQU     1
0002                    WRITE   EQU     2           FMS CODE FOR WRITE
0001                    READ    EQU     1           FMS CODE FOR READ
0004                    CLOSE   EQU     4           FMS CODE FOR CLOSE
                        *
                        *
0000 20 2B              XTRACT  BRA     XTR1        SKIP TO PROGRAM
0002 01                         FCB     VERNO       CURRENT VERSION
                        *
0003                    NUMBER  RMB     6           STORAGE FOR STMT #
0009 00 00              RCOUNT  FDB     0           REMARK COUNTER
000B 00 00              SCOUNT  FDB     0           STATEMENT COUNTER
000D                    SAVE    RMB     3           STORAGE FOR KEYWORD
0010 00                 REM     FCB     0           REMARK SWITCH
0011 00                 SNEW    FCB     0           STATEMENT NEEDS NUMBER SWITCH?
0012 00                 RNEW    FCB     0           REMARK NEEDS NUMBER SWITCH?
0013 00                 QUOTE   FCB     0           QUOTE FIELD (T/F)
                        *
                        * I/O ERROR PROCESSING
                        *
0014 CE 01 ED           ERR1    LDX     #MERR1      SYNTAX ERROR
0017 20 08                      BRA     ERROUT
```

```
 69  0019 CE 02 08  ERR2    LDX    #MERR2     CAN'T OPEN INPUT FILE
 70  001C 20 03             BRA    ERROUT
 71  001E CE 02 23  ERR3    LDX    #MERR3     CAN'T OPEN OUTPUT FILE
 72  0021 BD 05 09  ERROUT  JSR    PSTRNG
 73  0024 CE 01 B9          LDX    #FINAL     PRINT FINAL MESSAGE
 74  0027 BD 05 09          JSR    PSTRNG
 75  002A 7E 71 03          JMP    DOS        AND RETURN TO FLEX
 76                 *
 77                 * START PROGRAM BY OPENING FILES
 78                 *
 79  002D CE 01 88  XTR1    LDX    #INTRO     PRINT STARTING MESSAGE
 80  0030 BD 05 09          JSR    PSTRNG
 81  0033 CE 02 B8          LDX    #FCBIN
 82  0036 BD 71 27          JSR    GETFIL     PROCESS FILE SPEC AS INPUT FILE
 83  0039 25 D9             BCS    ERR1       SYNTAX ERROR?
 84  003B CE 02 B8          LDX    #FCBIN
 85  003E 86 03             LDA A  #3         SET EXT FOR .BAS FILES
 86  0040 BD 71 2D          JSR    SETEXT
 87  0043 C6 09             LDA B  #9         NUM OF CHARS IN FILENAME
 88  0045 FE 02 A5  COPY    LDX    CHAR90     INPUT FILENAME POSITION
 89  0048 A6 00             LDA A  0,X        GET CHAR
 90  004A 08                INX
 91  004B FF 02 A5          STX    CHAR90
 92  004E FE 04 F8          LDX    PTR1       TRANSFER FILE NAME
 93  0051 A7 00             STA A  0,X        TO FCBREM
 94  0053 08                INX
 95  0054 FF 04 F8          STX    PTR1
 96  0057 FE 04 FA          LDX    PTR2       TRANFER FILE NAME
 97  005A A7 00             STA A  0,X        TO FCBEXT
 98  005C 08                INX
 99  005D FF 04 FA          STX    PTR2
100  0060 5A                DEC B             DECREMENT CHAR COUNT
101  0061 26 E2             BNE    COPY       REPEAT UNTIL COPIED
102  0063 CE 02 B8          LDX    #FCBIN     OPEN FILE
103  0066 86 01             LDA A  #READ      FOR READING
104  0068 A7 00             STA A  0,X
105  006A BD 78 06          JSR    FMS
106  006D 26 AA             BNE    ERR2       ERROR IN OPEN PROCESS
107  006F CE 03 78          LDX    #FCBREM    OPEN REMARK FILE
108  0072 86 02             LDA A  #WRITE     FOR WRITING
109  0074 A7 00             STA A  0,X
110  0076 BD 78 06          JSR    FMS
111  0079 26 A3             BNE    ERR3       ERROR RETURN?
112  007B CE 04 38          LDX    #FCBEXT    OPEN CODE FILE
113  007E 86 02             LDA A  #WRITE     FOR WRITING
114  0080 A7 00             STA A  0,X
115  0082 BD 78 06          JSR    FMS
116  0085 26 97             BNE    ERR3       ERROR RETURN?
117                *
118                *  M A I N    P R O C E S S I N G    L O O P
119                *
120  0087 BD 02 83  CONT    JSR    BCHAR      GET NEXT NON-BLANK CHARACTER
121  008A 7D 02 A4          TST    EOF        OUT-OF-DATA?
122  008D 27 03             BEQ    CONT1      NOT YET
123  008F 7E 01 53          JMP    DONE       YES - TERMINATE LOOP
124  0092 CE 00 03  CONT1   LDX    #NUMBER    BEGINNING OF STATEMENT
125  0095 C6 01             LDA B  #TRUE      SET TO TRUE
126  0097 D7 11             STA B  SNEW
127  0099 D7 12             STA B  RNEW       TEST FOR STATEMENT #
128  009B 81 30     CONT2   CMP A  #'0        LESS THAN ZERO?
129  009D 2D 0C             BLT    NEXT       YES-GET OUT
130  009F 81 39             CMP A  #'9        GREATER THAN NINE?
131  00A1 2E 08             BGT    NEXT       YES-GET OUT
132  00A3 A7 00             STA A  0,X        SAVE AS STATEMENT NUMBER
133  00A5 08                INX
134  00A6 BD 02 83          JSR    BCHAR      GET NEXT CHAR
135  00A9 20 F0             BRA    CONT2      REPEAT PROCESS
136                *
137  00AB 6F 00     NEXT    CLR    0,X        CLEAR END-OF-NUMBER
138  00AD 20 03             BRA    TYPE
139  00AF BD 02 83  TYPEA   JSR    BCHAR      GET CHAR
140                *
141                * NOW DETERMINE STATEMENT TYPE
142
143                *
144  00B2 97 0D     TYPE    STA A  SAVE       SAVE 3-LETTER
145  00B4 BD 02 83          JSR    BCHAR      KEYWORD
146  00B7 97 0E             STA A  SAVE+1
147  00B9 BD 02 83          JSR    BCHAR
148  00BC 97 0F             STA A  SAVE+2
149  00BE C6 01             LDA B  #1         CONSTANT ONE
150  00C0 96 0D             LDA A  SAVE
151  00C2 81 52             CMP A  #'R        TEST FOR R-E-M
152  00C4 26 1C             BNE    STMT
153  00C6 96 0E             LDA A  SAVE+1
154  00C8 81 45             CMP A  #'E
155  00CA 26 16             BNE    STMT
156  00CC 96 0F             LDA A  SAVE+2
157  00CE 81 4D             CMP A  #'M
158  00D0 26 10             BNE    STMT
159  00D2 86 01             LDA A  #TRUE      FOUND ONE!
160  00D4 97 10             STA A  REM        SET REM SWITCH ON (TRUE)
161  00D6 DE 09             LDX    RCOUNT
162  00D8 BD 71 30          JSR    ADDBX      INCREMENT REM COUNTER
163  00DB DF 09             STX    RCOUNT
164  00DD CE 03 78          LDX    #FCBREM    SET FILE-ID
```

(continued)

the user's manual as well as provide some clues on how to correct the mistake.

The programmer's manual is directed towards the maintenance programmer, one who is consulted when the program fails to execute correctly, or when modifications are desired. Items of interest to the maintenance-programmer are commented source listing of the program, flowcharts, data file descriptions and layouts, and special or unique processing needs.

Data file specifications and layouts are crucial if the application is to be modified to retain additional information, as well as being an aid if file dumps are taken to verify the contents of the existing file.

Flowcharts are an area receiving mixed reactions from progammmers. Some proponents consider the flowchart essential to the documenation package, whereas others are content with a well-documented listing. Even the form of the flowchart varies from the ANSI version to the structured version (see Figure 1). Alternatives to the flowchart are possible, such as pseudocode, HIPO charts, and even computer generated flowcharts.

Comments are the English-like statements included with the language statements for the programmer's benefit, not the computer's. It appears that all computer languages have some feature which allows for comments, such as the C of * in column one used by FORTRAN, the REM statement for BASIC, the – for ADA, etc. If properly done, comments can provide a great deal of the documentation needed for a program. For example, it is common practice to include a header block of comments at the top of a program describing the purpose of the program, the author, date of last revision, etc. Even throughout the code, segments of comments can be introduced, thus visually separating portions of the program into individual units of code. Comments are intended to enlighten the reader, not duplicate his knowledge. For example, consider the following sample assembly statement:

```
ADD   ONE   ADD ONE TO THE ACCUM-
              ULATOR
```

It is obvious from the instruction what is being done, so that the comment is wasted. A more meaningful comment might be the following:

```
ADD   ONE   INCREMENT THE CARD
              COUNTER
```

In BASIC, the remark (REM) statement is used to denote a comment. Most versons allow multiple statements per line, separated by a special character, generally the colon (:). With this ability, it is possible to write a statement and immediately follow it with a comment,

much like the assembly example shown earlier.

The problem with remarks in BASIC is that they generally require memory space. The common versions of BASIC for microcomputers are interpreters. An interpreter normally keeps the source program in memory and, upon demand, decodes the statement and executes it. If a program were heavily commented, it would occupy considerably more memory space, as well as slowing down execution, than a comparable non-commented version. Thus it is quite easy to write a large program for a small machine and not have enough available memory space to load the program. One reaction would be to remove the comments to conserve space. Since this removes necessary information, it should not be done. What are the alternatives? Consider separating the comments out of the program as done by EXTRACT.

EXTRACT is a program written for SWTP BASIC running on a M6800 microcomputer using the FLEX operating system. EXTRACT will accept as its input a file containing the BASIC program. The output will consist of two files, one being all of the remarks found, and the second containing only instructions. The file containing just the REMARKS can be listed on a hard-copy device and become a permanent part of the programmer's documentation. The file of statements can be used for subsequent interpretation. Since it contains no remarks, its memory requirements will be smaller and execute faster.

EXTRACT is written in M6800 assembly language in order to properly read a source file. While SWTP BASIC can read files, string input is separated by commas, causing problems during input. Rather than write an assembly subroutine for a BASIC program, one language was used. EXTRACT is carefully commented so that one need only read the comments and understand the operation of the program. The concept of segments of code is used by including blocks of comments to separate the major portions. The header block includes information about the origin and purpose of the program. The flowchart can also be consulted for an overview of the logic.

For non-M6800 users, could this program be modified for another machine? Conversion of the concept should not be difficult due to the quality of the comments and code, as well as minimal reliance upon the operating system. For reasons mentioned earlier, a BASIC implementation would not be practical.

The idea of documentation lacks enthusiastic backers in the computer industry. Tools such as EXTRACT should be employed to automate the process as much as possible, thus encouraging and furthering the documentation process. □

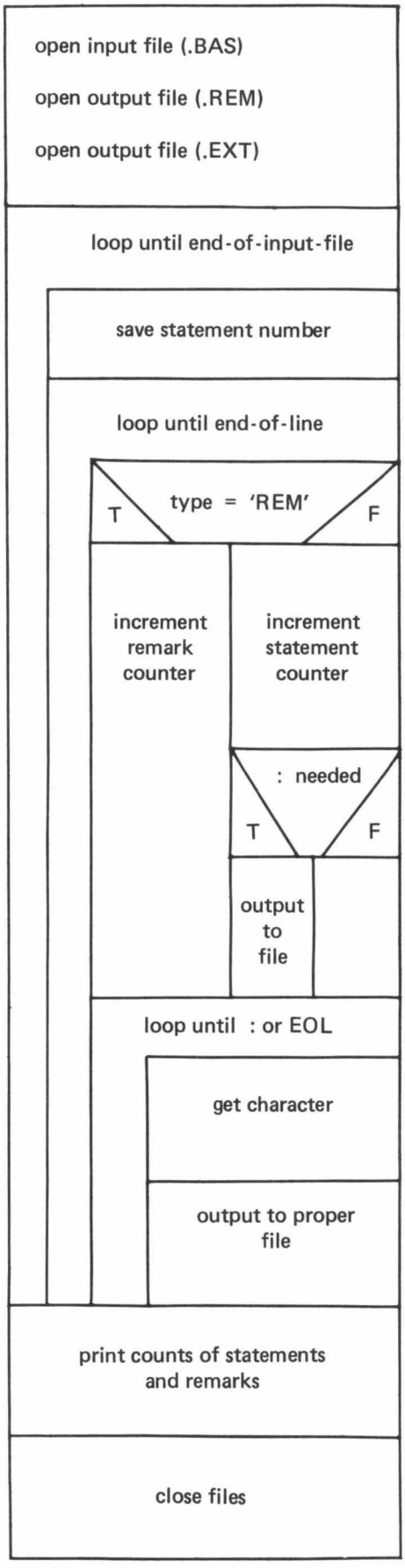
```
165  00E0 20 17            BRA     STMT2      CONTINUE
166  00E2 7F 00 10 STMT    CLR     REM        NOT REMARK STATEMENT
167  00E5 DE 0B            LDX     SCOUNT
168  00E7 BD 71 30         JSR     ADDBX      INCREMENT STATEMENT COUNTER
169  00EA DF 0B            STX     SCOUNT
170  00EC CE 04 38         LDX     #FCBEXT    SET FILE-ID
171  00EF 7D 00 11         TST     SNEW       FIRST STATEMENT?
172  00F2 26 05            BNE     STMT2      YES - BRANCH
173  00F4 86 3A            LDA A   #':        NO-OUTPUT SEPARATOR
174  00F6 BD 78 06         JSR     FMS
175                        *
176  00F9 FF 05 5B STMT2   STX     FILEID     SAVE FILE-ID
177  00FC 96 0D            LDA A   SAVE       NOW
178  00FE BD 05 10         JSR     OUTFIL     OUTPUT THREE CHARACTER
179  0101 96 0E            LDA A   SAVE+1     KEYWORD
180  0103 BD 05 10         JSR     OUTFIL
181  0106 96 0F            LDA A   SAVE+2
182                        *
183                        * COPY CHARACTERS UNTIL CR IF REM, ELSE
184                        *      ONLY COPY UNTIL ':' FOUND.
185                        *
186  0108 BD 05 10 LOOP    JSR     OUTFIL     OUTPUT TO FILE
187  010B BD 02 7E         JSR     GCHAR      GET AND COPY CHARACTERS
188  010E 81 0D            CMP A   #CR        END-OF-LINE?
189  0110 26 1D            BNE     OVER1      NO-SKIP OVER
190                        *
191                        * TEST IF CR IS NEEDED AT END OF EACH LINE
192                        *
193  0112 7D 00 11         TST     SNEW       STATEMENT NEEDS ONE?
194  0115 26 08            BNE     LOOP10     NOPE-BRANCH
195  0117 86 0D            LDA A   #CR        YES-OUTPUT CR
196  0119 CE 04 38         LDX     #FCBEXT
197  011C BD 78 06         JSR     FMS
198  011F 7D 00 12 LOOP10  TST     RNEW       REMARK NEEDS ONE?
199  0122 26 08            BNE     LOOP20     NO-BRANCH
200  0124 86 0D            LDA A   #CR        YES-OUTPUT CR
201  0126 CE 03 78         LDX     #FCBREM
202  0129 BD 78 06         JSR     FMS
203  012C 7E 00 87 LOOP20  JMP     CONT       GET NEXT LINE OF INPUT
204                        *
205                        *  TEST IF THE : IS IN A QUOTE FIELD, SUCH AS
206                        *       PRINT " HI: "
207                        *
208  012F 7D 00 13 OVER1   TST     QUOTE      CURRENTLY IN QUOTE FIELD?
209  0132 27 09            BEQ     OVER2      NO-BRANCH
210  0134 81 22            CMP A   #'"        END OF QUOTE FIELD?
211  0136 26 D0            BNE     LOOP       NOPE-GET NEXT CHAR
212  0138 7F 00 13         CLR     QUOTE      YES - CLEAR SWITCH
213  013B 20 CB            BRA     LOOP       AND CONTINUE
214                        *
215  013D 81 3A    OVER2   CMP A   #':        END-OF-STMT?
216  013F 26 08            BNE     OVER3      NO-SKIP
217  0141 7D 00 10         TST     REM        REMARK?
218  0144 26 C2            BNE     LOOP       YES-REPEAT COPY UNTIL CR
219  0146 7E 00 AF         JMP     TYPEA      NO - GET NEXT TYPE
220  0149 81 22    OVER3   CMP A   #'"        START OF QUOTE FIELD?
221  014B 26 BB            BNE     LOOP       NO-GET NEXT CHAR
222  014D C6 01            LDA B   #TRUE      YES - TURN SWITCH ON
223  014F D7 13            STA B   QUOTE
224  0151 20 B5            BRA     LOOP       AND REPEAT
225                        *
226                        * E N D   O F   R U N   P R O C E S S I N G
227                        *
228  0153 CE 02 B8 DONE    LDX     #FCBIN     CLOSE INPUT FILE
229  0156 BD 02 A8         JSR     FILECL
230  0159 CE 03 78         LDX     #FCBREM    CLOSE REMARK FILE
231  015C BD 02 A8         JSR     FILECL
232  015F CE 04 38         LDX     #FCBEXT    CLOSE CODE FILE
233  0162 BD 02 A8         JSR     FILECL
234                        *
235                        * FILES CLOSED, OUTPUT TOTALS
236                        *
237  0165 CE 00 09 DONE1   LDX     #RCOUNT    REMARK COUNT
238  0168 5F               CLR B              ZERO SUPPRESS
239  0169 BD 71 33         JSR     OUTDEC
240  016C CE 01 C6         LDX     #MESS1
241  016F BD 04 FC         JSR     PDATA
242  0172 CE 00 0B         LDX     #SCOUNT    STATEMENT COUNT
243  0175 5F               CLR B
244  0176 BD 71 33         JSR     OUTDEC
245  0179 CE 01 D8         LDX     #MESS2
246  017C BD 04 FC         JSR     PDATA
247  017F CE 01 B9         LDX     #FINAL     PRINT FINAL MESSAGE
248  0182 BD 05 09         JSR     PSTRNG
249  0185 7E 71 03         JMP     DOS        ALL DONE!!
250                        *
251                        * MISCELLANEOUS CONSTANTS AND DATA STORAGE
252                        *
253  0188 0D       INTRO   FCB     CR,LF
254  018A 45               FCC "Extract REM's from a BASIC program, version #"
255  01B7 31               FCB     VERNO+'0,EOT
256  01B9 45       FINAL   FCC     'End Extract.'
257  01C5 04               FCB     EOT
258  01C6 20       MESS1   FCC     ' REMarks found.'
259  01D5 0D               FCB     CR,LF,EOT
260  01D8 20       MESS2   FCC     ' statements found.'
```

```
261  01EA 0D          CRLF    FCB    CR,LF,EOT
262  01ED 53          MERR1   FCC    'Syntax error in file name.'
263  0207 04                  FCB    EOT
264  0208 55          MERR2   FCC    'Unable to open INPUT file.'
265  0222 04                  FCB    EOT
266  0223 55          MERR3   FCC    'Unable to open .REM or .EXT file.'
267  0244 0D                  FCB    CR,LF
268  0246 44                  FCC    'Do they already exist?'
269  025C 04                  FCB    EOT
270  025D 55          MERR4   FCC    'Unable to properly close a file.'
271  027D 04                  FCB    EOT
272                   *
273                   * SUBROUTINE TO RETURN NEXT CHARACTER
274                   *
275                   * ENTRY GCHAR RETURNS NEXT CHAR WHEREAS ENTRY
276                   *       BCHAR RETURNS NEXT NON-BLANK CHAR
277                   *
278  027E 7C 02 A7    GCHAR   INC    BFLAG     SET BLANK TRUE
279  0281 20 03               BRA    GCHAR1
280  0283 7F 02 A7    BCHAR   CLR    BFLAG     ELSE FLAG FALSE
281  0286 FF 02 A5    GCHAR1  STX    CHAR90    SAVE XR
282  0289 CE 02 B8            LDX    #FCBIN
283  028C BD 78 06    GCHAR2  JSR    FMS       GET CHAR
284  028F 26 0D               BNE    GCHAR8    ERROR RETURN
285  0291 7D 02 A7            TST    BFLAG     TEST BLANK FLAG
286  0294 26 04               BNE    GCHAR3    OK-RETURN
287  0296 81 20               CMP A  #' '      IS IT SPACE?
288  0298 27 F2               BEQ    GCHAR2    YES - GET NEXT CHAR
289  029A FE 02 A5    GCHAR3  LDX    CHAR90    RESTORE XR
290  029D 39                  RTS              AND RETURN
291  029E 86 01       GCHAR8  LDA A  #TRUE     SET
292  02A0 B7 02 A4            STA A  EOF       END-OF-FILE ON
293  02A3 39                  RTS
294                   *
295  02A4 00          EOF     FCB    0
296  02A5 02 BB       CHAR90  FDB    FCBIN+3   XR STORAGE
297  02A7 00          BFLAG   FCB    0         RETURN BLANK INDICATOR
298                   *
299                   * SUBROUTINE TO CLOSE A FILE
300                   *
301  02A8 86 04       FILECL  LDA A  #CLOSE    CLOSE CODE
302  02AA A7 00               STA A  0,X
303  02AC BD 78 06            JSR    FMS
304  02AF 26 01               BNE    FILER     ERROR?
305  02B1 39                  RTS              NO-RETURN
306  02B2 CE 02 5D    FILER   LDX    #MERR4
307  02B5 7E 00 21            JMP    ERROUT    ERROR PROCESSING
308                   *
309                   * FILE CONTROL BLOCKS (FCB) FOR
310                   *   .BAS      - INPUT FILE
311                   *   .EXT      - CODE FILE
312                   *   .REM      - REMARK FILE
313                   *
314  02B8             FCBIN   RMB    192
315  0378             FCBREM  RMB    12
316  0384 52                  FCC    'REM'
317  0387                     RMB    177
318  0438             FCBEXT  RMB    12
319  0444 45                  FCC    'EXT'
320  0447                     RMB    177
321  04F8 03 7B       PTR1    FDB    FCBREM+3  FILE NAME PTR
322  04FA 04 3B       PTR2    FDB    FCBEXT+3  FILE NAME PTR
323                   *
324                   * SUBROUTINE TO OUTPUT CHARACTERS,
325                   * TERMINATED BY EOT.
326                   *
327  04FC A6 00       PDATA   LDA A  0,X       GET CHAR
328  04FE 81 04               CMP A  #EOT       END-OF-STRING?
329  0500 27 06               BEQ    PDATA1    YES - RETURN
330  0502 BD 71 12            JSR    PUTCHR    OUTPUT CHARACTER
331  0505 08                  INX
332  0506 20 F4               BRA    PDATA     AND REPEAT
333  0508 39          PDATA1  RTS              RETURN
334                   *
335                   * SUBROUTINE PSTRNG
336                   *    CALL PDATA1, AND THEN PRINT CR/LF
337                   *
338  0509 8D F1       PSTRNG  BSR    PDATA     OUTPUT STRING
339  050B CE 01 EA            LDX    #CRLF
340  050E 20 EC               BRA    PDATA     OUTPUT CR,LF
341                   *
342                   * SUBROUTINE OUTFIL -
343                   *    OUTPUT ACC-A TO PROPER FILE,
344                   *    OPTIONALLY OUTPUT STATEMENT NUMBER IF
345                   *    NEEDED.
346                   *
347  0510 B7 05 58    OUTFIL  STA A  TCHAR     SAVE CHARACTER
348  0513 7D 00 10            TST    REM       IN REMARK?
349  0516 26 0A               BNE    OUT10     YES - BRANCH
350  0518 7D 00 11            TST    SNEW      STATEMENT NEEDS NUMBER?
351  051B 27 2C               BEQ    OUT80     NO-BRANCH
352  051D 7F 00 11            CLR    SNEW      CLEAR INDICATOR
353  0520 20 08               BRA    OUT50     BR TO OUTPUT STMT #
354  0522 7D 00 12    OUT10   TST    RNEW      REMARK NEEDS NUMBER?
355  0525 27 22               BEQ    OUT80     NO-BRANCH
356  0527 7F 00 12            CLR    RNEW      CLEAR INDICATOR
```

(continued)

Figure 1.

Structured Flowchart for EXTRACT



```
  357                   *
  358                   * OUTPUT STATEMENT NUMBER
  359                   *
  360   052A CE 00 03   OUT50   LDX    #NUMBER
  361   052D FF 05 59           STX    SAVEXR     PT TO FIRST DIGIT
  362   0530 FE 05 59   OUT60   LDX    SAVEXR
  363   0533 A6 00              LDA A  0,X        GET DIGIT
  364   0535 27 12              BEQ    OUT80      BR IF NO MORE
  365   0537 08                 INX               ADVANCE TO NEXT DIGIT
  366   0538 FF 05 59           STX    SAVEXR     AND SAVE POSITION
  367   053B FE 05 5B           LDX    FILEID
  368   053E BD 78 06           JSR    FMS        OUTPUT TO FILE
  369   0541 27 ED              BEQ    OUT60      REPEAT IF NO ERROR
  370   0543 CE 05 5D   OUT65   LDX    #IOERR
  371   0546 7E 00 21           JMP    ERROUT     ERROR PROCESSING
  372                   *
  373                   * OUTPUT SINGLE CHARACTER (TCHAR)
  374                   *
  375   0549 FE 05 5B   OUT80   LDX    FILEID
  376   054C B6 05 58           LDA A  TCHAR      OUTPUT CHARACTER
  377   054F BD 78 06           JSR    FMS
  378   0552 26 EF              BNE    OUT65      ERROR?
  379   0554 B6 05 58           LDA A  TCHAR      RESTORE CHAR
  380   0557 39                 RTS               AND RETURN
  381                   *
  382   0558 00         TCHAR   FCB    0          STORAGE FOR SINGLE CHAR
  383   0559 00 00      SAVEXR  FDB    0          XR STORAGE
  384   055B 00 00      FILEID  FDB    0          ADDRESS OF FILE-FCB
  385   055D 49         IOERR   FCC    'I/O error when writing to output file.'
  386   0583 04                 FCB    EOT
  387                           END    XTRACT

NO ERROR(S) DETECTED


  +++EXTRACT,SIMPLE.BAS
  Extract REM's from a BASIC program, version #1
  3 REMarks found.
  4 statements found.
  End Extract.

  +++LIST,SIMPLE.EXT
  20LET A=5
  30PRINT A :PRINT "MULTIPLE STATEMENTS"
  50END

  +++LIST,SIMPLE.REM
  10REM THIS IS A REMARK
  20REM THIS IS AN INLINE REMARK
  40REM ALL DONE

  +++LIST,SIMPLE.BAS
  10 REM THIS IS A REMARK
  20 LET A=5 : REM THIS IS AN INLINE REMARK
    30  PRINT A : PRINT "MULTIPLE STATEMENTS"
  40 REM ALL DONE
  50 END

  +++
```

END

Symbol Table

ADDBX	7130	BCHAR	0283	BFLAG	02A7	CHAR90	02A5	CLOSE	0004
CONT	0087	CONT1	0092	CONT2	009B	COPY	0045	CR	000D
CRLF	01EA	DONE	0153	DONE1	0165	DOS	7103	EOF	02A4
EOT	0004	ERR1	0014	ERR2	0019	ERR3	001E	ERROUT	0021
FCBEXT	0438	FCBIN	02B8	FCBREM	0378	FILECL	02A8	FILEID	055B
FILER	02B2	FINAL	01B9	FMS	7806	GCHAR	027E	GCHAR1	0286
GCHAR2	028C	GCHAR3	029A	GCHAR8	029E	GETFIL	7127	INTRO	0188
IOERR	055D	LF	000A	LOOP	0108	LOOP10	011F	LOOP20	012C
MERR1	01ED	MERR2	0208	MERR3	0223	MERR4	025D	MESS1	01C6
MESS2	01D8	NEXT	00AB	NUMBER	0003	OUT10	0522	OUT50	052A
OUT60	0530	OUT65	0543	OUT80	0549	OUTDEC	7133	OUTFIL	0510
OVER1	012F	OVER2	013D	OVER3	0149	PDATA	04FC	PDATA1	0508
PSTRNG	0509	PTR1	04F8	PTR2	04FA	PUTCHR	7112	QUOTE	0013
RCOUNT	0009	READ	0001	REM	0010	RNEW	0012	SAVE	000D
SAVEXR	0559	SCOUNT	000B	SETEXT	712D	SNEW	0011	STMT	00E2
STMT2	00F9	TCHAR	0558	TRUE	0001	TYPE	00B2	TYPEA	00AF
VERNO	0001	WRITE	0002	XTR1	002D	XTRACT	0000		

(continued from page 9)

N-LOGS cont'd

NOTE

Following the terminology well-established by the pocket calculator, the following abbreviations will be used in the balance of this paper: LOG will stand for "the logarithm to the base 10;" EXP will mean "10 raised to the ___ power." The distinction between LOG (log_{10}) and log (general term) should be kept in mind.

REFERENCES

Reference (1): The article on "Logarithms" in the 1967 edition of the Encyclopaedia Brittanica, *by K.O. May.*

Reference (2): The International Dictionary of Applied Mathematics; *D. Van Nostrand 1960; definition of "Accuracy."*

Reference (3): Keneth A. Simons, "The dB Anything," Proc. IEEE (Letters), *pp. 495-96, April 1973.*

MOVIT

by William D. Loughman

The standard 2000 hex address of the North-Star DOS can be an annoyance at times. Large programs starting anywhere below the DOS sometimes cannot be loaded from disk, as the DOS may be overwritten before the load is completed. There are a lot of nice programs which were not designed for use with the N * disk system.

These can be used painlessly by tacking them onto a short 'mover' program. MOVIT cares not where it is started. It finds its own location in RAM by putting a 'RET' at 0 hex, executing 'RST 0', and accessing the stack to find the return address. Subsequent operations are all by adding offsets to this reference address. MOVIT is short enough to enter 'by hand' with a monitor; just pay attention to changes needed in lines 80-81 and line 93. If you use an assembler, in addition to those changes you should delete line 16 and you must change the execution address assignment in line 41. After assembly, load your own program to reside at whatever address is represented by line 100. The 6 bytes there are only a sample; load on top of them. Save the combined programs to disk.

Now the combination can be loaded from disk and executed at almost any arbitrary location in memory. Your program will be moved to its correct location (overwriting the DOS, perhaps) and started up. I give my disk files a 'GO' address in high memory, making sure the loaded program will fit. Then all I need to do is 'GO . . .', and it auto-executes just like 'proper' programs do.

An aside: this letter is formatted using H. Watson's POW (*DDJ* #29). L. Morgenstern modified that for use with Processor Tech's EDIT program (*DDJ* #32:42 Feb. '70). I have adapted the latter version for use with either ALS-8 or PILOT file structures, which I use in my home-brew (disk) operating system. This allows use of the nice screen editors in those programs, and thus provides a pretty good and very, very inexpensive CRT word processor system. Many thanks to them, and to you. □

LISTING

```
ASSMX 0

0000                    0001 *       * M O V I T *
0000                    0002 *
0000                    0003 * An address-independent routine for
0000                    0004 * 'load-and-go' use, from disk, of
0000                    0005 * programs blocked by the standard
0000                    0006 * NorthStar DOS at 2000 hex.
0000                    0007 *
0000                    0008 * Orig.: Wm.D.Loughman  1 May 1980
0000                    0009 * PUBLIC DOMAIN:  12 June 1980
0000                    0010 *
0000                    0011 * Set BYTES, BOTAD, and XEQ (below)
0000                    0012 * to reflect your own program needs.
0000                    0013 * (EXAMPLEs below: use for a test)
0000                    0014 *
0000                    0015 *  my OS and assembler usage:
0000                    0016 SOLOS  EQU     0C004H  for EXAMPLE
0000                    0017 SP     EQU     6       (for ALS-8)
0000                    0018 *
0000                    0019 * Assemble this program-mover at any
0000                    0020 * convenient address.  Add your own
0000                    0021 * program to it at PROGM (below).
0000                    0022 *
0000                    0023 * Create a disk file large enough to
0000                    0024 * hold BOTH your program AND this
0000                    0025 * preamble.  Save all of it to disk.
0000                    0026 *
0000                    0027 * Give the disk file any convenient
0000                    0028 * 'GO' address which is neither in
0000                    0029 * the DOS address space, nor that of
0000                    0030 * your program's final location.
0000                    0031 *
0000                    0032 * Now, 'GO (filename)', and enjoy.
0000                    0033 *
0000                    0034 *
0000                    0035 *    -- SET VARIABLES HERE --
0000                    0036 * your program's length in bytes:
0000                    0037 BYTES  EQU     6       for EXAMPLE
0000                    0038 * your program's correct load addr:
0000                    0039 BOTAD  EQU     0       for EXAMPLE
0000                    0040 * your program's execution addr:
0000                    0041 XEQ    EQU     SOLOS   for EXAMPLE
0000                    0042 *
0000                    0043 *
0000                    0044 *         -- MOVIT --
0000                    0045 * assume a valid stack (1 call);
0000                    0046 * and find our own RFRNC address
0000 21 00 00           0047         LXI     H,0     'RST 0' needs
0003 36 C9              0048         MVI     M,0C9H  'RET'
0005                    0049 *
0005 C7                 0050         RST     0       call it
0006 3B                 0051 RFRNC  DCX     SP      return here
0007 3B                 0052         DCX     SP      to get
0008 E3                 0053         XTHL    .       RFRNC addr
0009 5D                 0054         MOV     E,L     clone it
000A 54                 0055         MOV     D,H
000B                    0056 * now, make our own stack at RFRNC,
000B                    0057 * ruining this nice program!
000B F9                 0058         SPHL
000C                    0059 *
000C                    0060 * find MOVE addr
000C 3E 20              0061         MVI     A,MOVE-RFRNC  offset
000E 85                 0062         ADD     L       plus RFRNC addr
000F 6F                 0063         MOV     L,A
0010 3E 00              0064         MVI     A,0
0012 8C                 0065         ADC     H
0013 67                 0066         MOV     H,A     is MOVE addr
0014 E3                 0067         XTHL    .       saved in stack
0015                    0068 *
0015                    0069 * find PROGM (to move)
0015 EB                 0070         XCHG    .       REFRNC addr
0016 3E 2D              0071         MVI     A,PROGM-RFRNC
0018 85                 0072         ADD     L       plus offset
0019 6F                 0073         MOV     L,A
001A 3E 00              0074         MVI     A,0
001C 8C                 0075         ADC     H
001D 67                 0076         MOV     H,A
001E EB                 0077         XCHG    .       is PROGM addr
001F                    0078 *
001F                    0079 * set up the move
001F 01 06 00           0080         LXI     B,BYTES  its length
0022 21 00 00           0081         LXI     H,BOTAD  its load addr
0025 E3                 0082         XTHL    .       (balances MOVE)
0026                    0083 *
0026                    0084 * move the program; execute it
0026 E3                 0085 MOVE   XTHL    .       get load addr
0027 1A                 0086         LDAX    D       move a byte
0028 77                 0087         MOV     M,A
0029 13                 0088         INX     D       set up next
002A 23                 0089         INX     H
002B 0B                 0090         DCX     B       bytes-1
002C 79                 0091         MOV     A,C
002D B0                 0092         ORA     B       done?
002E CA 04 C0           0093         JZ      XEQ     do it, else
0031                    0094 * put load addr in stack, also
0031 E3                 0095         XTHL    .       get MOVE addr
0032 E9                 0096         PCHL    .       and do more
0033                    0097 *
0033                    0098 * your program goes here,
0033                    0099 * replacing this dummy.
0033                    0100 PROGM  EQU     $
0033 11 11              0101         DW      1111H
0035 22 22              0102         DW      2222H
0037 33 33              0103         DW      3333H
0039                    0104 *

BOTAD   0000    0081
BYTES   0006    0080
MOVE    0026    0061
PROGM   0033    0071
RFRNC   0006    0061 0071
SOLOS   C004
SP      0006    0051 0052
XEQ     C004    0093
```

END

William D. Loughman – 293 Gravatt Dr., Berkeley, CA 94705

6800 and 1802 Cross-Assembler for CP/M

by William C. Colley III

What Is a Cross-assembler?

A cross-assembler, like a normal assembler, is a program that converts human-readable assembler source code for a given microprocessor into binary machine code for the same processor. The difference between a cross-assembler and a normal assembler is this: a normal assembler program is run on the same type of processor as the output of the assembler and a cross-assembler program is run by a different processor. For example, a 6800 assembler runs on a 6800 and converts source code in 6800 machine code. This particular 6800 cross–assembler runs on an 8080 or Z-80 and converts 6800 source code into 6800 machine code.

With an appropriate cross-assembler, you can write and assemble code for another (in this case 6800 or 1802) processor using your existing (in this case CP/M) editors and floppy disks. This keeps you from having to have separate disks, disk operating systems, editors, etc. for each different processor.

General Description of Both Cross-assemblers

CP/M comes with an assembler (ASM.COM) of about the same capabilities as these two cross-assemblers. Thus, the cross-assemblers will be described in terms of how they compare to the ASM.

The assembler syntax of the cross-assemblers is similar to ASM's, but not identical. Line numbers are not permitted, as the cross-assemblers make no attempt to scan them off. A leading * will cause an error rather than mark a comment line. Comment lines should begin with a semicolon. No colons should appear after labels as the colon is a legal alphabetic character and will become part of the symbol name. Labels are distinguished from opcodes by the fact that labels begin in column 1 and opcodes do not.

The opcodes are those of the particular microprocessor instead of those of the 8080. The only lower-case to upper-case conversions performed by the assembler are in opcocdes and keywords such as NOT, AND, MOD, etc. Thus the opcode "LDA" is the same as "lda", but the label "LABEL" is not the same as the label "label".

The expression evaluator has more operators than ASM's and the operators have a slightly different precedence order. The additional operators are:

HIGH meaning "the high byte of"
LOW meaning "the low byte of"
relational operators EQ, NE, LT, LE, GT, GE (which can also be written as =, <>, <, <=, >, >= respectively)

William C. Colley, III–409 Marlborough St. #24, Boston MA 02115

Finally, the pseudo-ops have been renamed according to the particular processor. Also the IF-ENDIF pair has been enhanced to IF-ELSE-ENDI. The ELSE is, of course, optional.

Like ASM, the output of the cross-assemblers is in INTEL hex. This was chosen because INTEL hex can be loaded into memory by DDT and SID. Of course, the disassembly functions of DDT and SID will not work correctly on a non-8080 instruction set.

Particular Description of the 6800 Cross-assembler

The 6800 cross-assembler's basic syntax is as follows:

```
[label]     LDA A       X, 3        ;Comment field.
```

The parser is built such that the accumulator specifier "A", the indexed mode designator "X", and the offset "3" can be specified in any order. The designators "A", "B", and "X" can be written in either upper or lower case. Immediate addressing is specified by a # sign as per Motorola usage.

Also in accordance with Motorola usage, two formats for numeric constant are supported. The Motorola-format constant $FF is equivalent to the INTEL-format constant 0FFH. Hex numbers and INTEL-format base designators can be written in either upper or lower case.

The current location counter is referenced by a * instead of the $ sign as per Motorola usage.

Finally, when given the choice between direct (short) addressing and extended (long) addressing, the assembler will choose direct addressing if the following two conditions are met:

1. The address in question contains no forward references.
2. The address in question is in the range 0 to $FF.

Particular Description of the 1802 Cross-assembler.

The 1802 cross-assembler implements two additional pseudo-ops above the set available in ASM. They are:

1. PAGE which sets the current location counter to the top of the next 256-byte page, and
2. LOAD register, value
   This is, in effect, a built-in macro. The assembly of a LOAD pseudo-op places the following code into the object file:
   LDI HIGH value
   PHI register
   LDI LOW value
   PLO register

The LOAD pseudo-op is included since this construct is so

common in 1802 code that RCA saw fit to include it as a pseudo-op in some of their assemblers.

What Machine Resources are Required?

The cross-assemblers are supplied in source form only (except as noted below) as both the source and object of the assemblers would not fit on a single-density 8" disk with the other three programs on the disk. Each assembler has six modules written in BDS C. A seventh module (called either A68TBLS. MAC or A18TBLS.MAC) is written in Microsoft's MACRO-80 in such a way that it assembles into a link module for the BDS C linker. The details as to how this is done are included on the disk. In addition, for those who have BDS C but not MACRO-80, the assembled versions (A68TBLS. CRL and A18TBLS.CRL) are included on the disk. BDS C and MACRO-80 are available through Lifeboat Associates who advertise in most of the microcomputer magazines.

The required machine resources are thus:

1. the BDS C compiler, and (optionally) the MACRO-80 macro-assembler,
2. 40K or more of RAM, and
3. CP/M version 1.3 or 1.4.

The cross-assemblers have not been tested under CP/M 2.x, though there should be no problems involved.

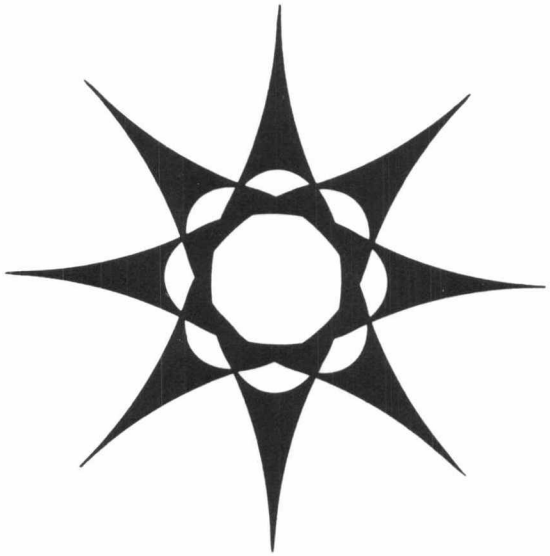
What Do I Get and Where Do I Get It?

The cross-assemblers are available on single-density 8" disk from the C User's Group. Send them a disk and enough postage to mail it back to you. Write to: C User's Group, c/o John Nall, Computer Center, Love Building, Florida State University, Tallahassee, Florida, 32306, (904) 644-2764.

The disk includes the 8 source files of each cross-assembler (an include file, 6 C modules, and the MACRO-80 module), the assembled versions of the MACRO-80 modules, and print image copies (ready to pip to lst:) of the 20-page user manuals. The disk also includes a combination BDS C/MACRO-80 program that uses a parallel input bit and a parallel output bit to read and write cassette tapes compatible with the E1.F II. This gives you a method of getting the output of the 1802 cross-assembler into an 1802 system to be executed. The tape program also requires a simple hardware interface (1 CMOS chip, 5 resistors, 2 capacitors, and a potentiometer) that is diagrammed in the program documentation. A couple of useful disk utilities written in BDS C are also on the disk.

Conclusion

I wrote the cross-assemblers for my own use and for the use of a friend. I saw no reason, however, why others should have to "reinvent the wheel," so I am trying to distribute them to anyone who wants them. I hope they prove useful. □



A legal note: CP/M, ASM. DDT, and SID are trademarks of Digital Research, and MACRO-80 is a trademark of Microsoft.

A Proposal On Strings for FORTH

by Ralph Deane

Once a person has used FORTH for a period of time, two deficiencies usually appear. These are the lack of floating point (real) numbers, and the absence of any string handling functions. Much of the work currently being done in BASIC could be done quicker and easier in FORTH if these two problem areas were fixed.

The area of floating point numbers has been adddressed by different groups, and Kitt Peak Observatory has published their approach in the Kitt Peak FORTH Primer. This will probably become the standard in the future.

The problem of string handling in FORTH is an area which, as yet, has received little attention. In this article I propose to outline one approach to solving this problem. In no way is this method to be thought of as a finalized standard solution to string handling. It is meant to be a starting point from which to build a unique (and powerful) string handling package.

The obvious place to start is at the beginning: creating string variables. Since FORTH is a member of the structured family of languages, it requires all variables to be declared before use. String variables will be no exception. This raises the first problem: should we allow strings to have unlimited length or should they have a maximum length set by the system? Also, where shall we store the string data: in the dictionary or in a separate table?

In order to remain compatible with the other data types supported by the language, I feel that the string should be stored in the dictionary. This gives the user maximum access to the data with a minimum of overhead. This storage system will also require that a maximum length be set by the system in order to know where the next free dictionary location is situated. If the maximum length is set at 255 (the same as Micro-Soft BASIC) then any character in the string can be addressed with an eight bit index register. This can be very important on most micros in terms of speed and program size.

Another nice feature would be to allow the user to specify, for any string variable, a maximum length which is less than or equal to the system's maximum length. This would allow a great saving in dictionary space requirements for short strings. It hardly makes sense to reserve 255 bytes of space for a variable which the user knows will have a maximum of 10 characters stored in it.

If the user is allowed to specify a maximum size, then that size should be stored at the start of the string area. A null ($00) should also be stored as the last character in order to signal the end of the string. This would give an overhead of two bytes per variable but would make for much more efficient programming.

In keeping in line with other FORTH variables, a string variable should place the address of its first character on the sack when the variable name is executed. It should also put the current length of the string on the stack. The reason for this is that the current length can vary from the maximum length and the string handling functions should not have to search a string in order to know its length. This would also make it compatible with the string output function, TYPE, which has already been defined. Since TYPE requires the length to be the top stack value, with the address second, let's also define string variables so that they place their data on the stack in the same order.

Now that we have defined the form of our string variables we only need to implement them via a variable defining function.

The defining function could be called STRING (a rather obvious choice) and will probably be defined by using <BUILDS and DOES>. FIGURE I shows two different approaches to implementing STRING. Function (a) creates a variable whose current length is equal to the maximum length and which contains all blanks. Function (b) will create string variables with a current length of 1, with that character being a blank. Neither (a) nor (b) will allow variables to be created with a maximum length less than 1 or greater than 255. The runtime routine of both (a) and (b) are identical. It increments the starting address of the character string around the length byte and then searches the string to find the current length. This data is left on the stack in the required order.

The other data type associated with strings, besides variables, is the literal string. This is usually a string of characters enclosed in quotes. Since the single quote, ', has already been used as a func-

FIGURE I

```
 0 ( Definition of STRING - two types )
 1 : SRCH DUP BEGIN DUP C@ SWAP 1 + SWAP 0= END SWAP - 1 - ;
 2 ( Type a STRING definition )   DECIMAL
 3 : STRING <BUILDS ABS 255 MIN 1 MAX DUP C, 0 DO 32 C,
 4 LOOP 0 C, DOES> 1 + DUP SRCH ;
 5 ( Type b STRING definition )   DECIMAL
 6 : STRING <BUILDS ABS 255 MIN 1 MAX DUP C, 32 C, 0 DO
 7 0 C, LOOP DOES> 1 + DUP SRCH ;
 8 ( When defining STRING you must chose either type a
 9 or type b to implement. See text for an explaination
10 of the differences. )
11
12
13
14
15
```

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

tion name we will have to use the double quote, ". Since " will be used as a function name it must be separated from the rest of the strings by a blank. Another " can be used as the terminating character.

Since we desire maximum flexibility from any function, the literal string function must be directly executable or compilable, depending on the user's needs. It should also be compatible with the string variables so that the string handling functions do not have to know which data type they are dealing with. This means that a literal string, when executed, will place the starting address of the data, as well as the string length, on the stack. The string, wherever it is stored, must also have a null ($00) appended to it to signify the last character.

An inspection of FIG FORTH'S ." function shows a method which can be used to define the " function. The function is split into two parts and the compilation flag is used to choose which part is executed. If " is being used inside a colon definition (i.e., :the flag is set) the string is compiled into the dictionary along with the address of a runtime routine, ("), and the string length. The function ("), when executed, will place the string address and its length on the stack.

Direct execution of " produces a different sequence. The string is stored in the area at the end of the dictionary, along with its length and terminating character. The address of this area and the length of the string is then placed on the stack. Since the string is stored in a rather volatile area, any operations to be performed on it must be done before anything else is added to the dictionary.

The definition of " is left up to the implementor as it will depend on the specific system being used. The ." function can be used as a model if so desired. One thing to be remembered is that a null must be appended to the string for proper operation of the handling functions. Also " is to be made an immediate function.

Now that the format and runtime characteristics of the string data types have been outlined we can go on and define some string handling functions. At the minimum the package should contain all the string functions supported by BASIC. More functions can be added as the need arises.

One of the first functions needed is the string store routine. In keeping with a FORTH standard it will be called S! (pronounced string-store). It will take data from one string and place it in another. The variable in which the data is stored will have a null placed at the end of the new string to indicate the current length. The string will also be truncated if its length is longer than the variable's maximum length.

Where S! is used to replace the whole string, the function SUB! is used to replace a substring on a one-for-one basis. The current length of the variable will be unchanged by use of SUB! . If the string is longer than the substring it replaces, it will be truncated at the substring's length.

The stack values required for S! and for SUB! are identical. The top two values are the address and length of the destination string (or substring). The next two values are the address and length of the source string. The lengths do not have to match, as the string is adjusted to fit the destination string. FIGURE II is one method of defining S! and SUB! . Trying to store into a literal string, while possible, is a real no-no and drastic happenings can occur if it is attempted.

The next type of handling function to be discussed will be the substring extraction routines. These include MID$, LEFT, and RIGHT$. These functions generate addresses and lengths of specified substrings. Since LEFT$ and RIGHT$ are just special cases of MID$, we will concentrate on it for the moment.

There are two different types of MID$ functions presently in use in BASIC. One type requires the starting character location and the length of the desired substring. The other needs the location of both the starting and ending characters.

I have, for no specific reason, chosen the second type of MID$. The first type can be used if so desired, and is equally as easy to define. MID$ should make sure that the substring it defines is entirely inside the parent string. If it is not, then the starting and ending locations should be adjusted. MID$ expects the top two stack values to be the address and length of the parent string. The next value is the ending character location, while the last value is the starting character location. These last two values will be forced into the range of 1 to the current string length. MID$ will return the address and length of the substring as defined by these values.

LEFT$ is the special case of MID$ where the starting character location is assumed to be 1. Only the address and length of the parent string, as well as the ending character location, are needed. RIGHT$ is another special case in which the ending character location is assumed to be the last character in the parent string. The starting character location and the parent string information is all that RIGHT$ requires. FIGURE III is one possible definition of MID$, RIGHT and LEFT$.

Concatenation of strings is another

FIGURE II

```
 0 ( Definition of S! and SUB! )
 1 : S! DROP DUP 1 - C@ ROT MIN 1 MAX OVER OVER +
 2 0 SWAP C= CMOVE ;
 3
 4 : SUB! ROT MIN 1 MAX CMOVE ;
 5
 6 ( Note that in FIG FORTH the function C= is called C! )
 7
 8
 9
10
11
12
13
14
15
```

FIGURE III

```
 0 ( Definitions of MID$, LEFT$ and RIGHT$ )
 1 : MID$ SWAP SAVE ROT MIN 1 MAX SWAP OVER MAX OVER - 1
 2 + SWAP UNSAVE + 1 - SWAP OVER SRCH MIN ;
 3
 4 : LEFT$ SAVE SAVE 1 SWAP UNSAVE UNSAVE MID$ ;
 5
 6 : RIGHT$ SAVE SAVE 256 UNSAVE UNSAVE MID$ ;
 7
 8 ( Note that in FIGFORTH the functions SAVE and UNSAVE
 9 are called >R and R> , respectively. )
10
11
12
13
14
15
```

function which should be part of the package. S + will be used as the concatenation function name. Due to the way that strings have been defined S + will have a few restrictions. The location literal strings are stored in, when directly executed, rules out their being both arguments of S +. Also the resulting string produced by S + will have a maximum length of 255, as no variable can hold more characters than that. S + will store the new string in a temporary location and place that address and string length on the stack. A null will be appended to the new string.

Since one of the arguments of S + can be a literal string, we cannot use the area at the end of the dictionary as temporary storage. An argument could be destroyed if we did. Instead we will have to create a storage area in the dictionary. This could be done with an ARRAY definition or any other storage reserving method. If the ARRAY data type is supported on the system, it is the easiest way to reserve space and still have access to the address. The temporary storage area must be 256 bytes (255 characters plus a null) long. FIGURE IV shows one possible definition of S +. Assume that the top two stack values define a string called B, while the next two values define the string A. S + will produce the string (A + B). This new string will be truncated after 255 characters, with the first data coming from string A.

A few miscellaneous functions are all that is needed to complete the basis of our string handling package. These include VAL, STR$, LEN and MLEN. We shall now look at these individually.

VAL is used to convert the string representation of a number into its numeric value. In FORTH this should be done with regard to the current BASE. When VAL is executed, the top two stack values describe the string to be converted. Conversion will stop at the first non-numeric character. The value computed is left on the top of the stack.

STR$ converts a number into a string representation of that number. This should also be done with regard to the current BASE. The top stack value is the number to be converted. STR$ will leave the address and length of the string produced on the stack. A special area can be set aside to hold the string or it could be treated as a literal string. A null should be appended to the end of the string. Since precision numbers would require a maximum of 7 characters of storage.

LEN returns the current length of the string defined by the top two stack entries. MLEN returns the maximum length of that string.

Since VAL and STR$ can make good use of routines already a part of the system, no definitions are shown for them. The implementor can tailor these functions to each individual system. FIGURE V shows the definitions for LEN and MLEN.

Two more functions which might be useful to have in the package are string input (SINP) and string comparison (S=). SINP expects the top two stack entries to describe the string into which the input data is to be placed. This should be a variable. S= assumes the top four stack entries describe the two strings which are to be compared. It will return a 1 if the strings are identical, including length, and a 0 otherwise. FIGURE VI lists the definitions for SINP and S =

This concludes my proposal for string handling functions for FORTH. I have tried to present a string data type which will be easy to add new functions to at a later date. This is very important in FORTH as the language is so easily extended that new functions are constantly being added. In order to show the power of these functions I have recoded the Soundex program which was presented in the MAY 1980 issue of BYTE magazine on pages 196-200. Executing SOUND will cause a prompt for the name. The name is terminated after 30

FIGURE IV

```
0 ( Definition of S+ )
1 ( The array TEMP is the temporary storage area. When
2 the name TEMP is executed it will place the address of
3 the first data location on the stack. )
4 128 ARRAY TEMP     ( This reserves 256 bytes of space )
5
6 : S+ ROT SAVE ROT UNSAVE SWAP OVER TEMP SWAP CMOVE
7 SWAP OVER + 255 MIN DUP SAVE OVER - SWAP TEMP + SWAP
8 CMOVE UNSAVE 0 OVER TEMP + C= TEMP SWAP ;
9
10
11
12
13
14
15
```

FIGURE V

```
0 ( Definition of LEN and MLEN )
1
2 : LEN SWAP DROP ;
3
4 : MLEN DROP 1 - C@ ;
5
6
7
8
9
10
11
12
13
14
15
```

FIGURE VI

```
0 ( DEFINITIONS OF SINP AND S= )
1 : SINP OVER SWAP MLEN 0 DO DUP KEY DUP 13 =L IF LEAVE
2 DROP DROP ELSE SWAP C= 1 + THEN LOOP 0 SWAP C= ;
3 ( IT IS ASSUMED THAT KEY ECHOS THE INPUT DATA )
4
5 : S= ROT OVER =L IF 1 SWAP 0 DO DROP OVER C@ OVER C@
6 =L IF 1 + SWAP 1 + SWAP 1 ELSE 0 LEAVE THEN LOOP ELSE
7 DROP 0 THEN SWAP DROP SWAP DROP ;
8
9
10
11
12
13
14
15
```

characters or a carriage return. The Soundex code is then computed and typed out. The string variable S$ contains the code produced. For more information on Soundex codes consult the BYTE article. □

FIGURE VII

Soundex Coder

```
 0 ( DEFINITION OF SOUNDEX CODER )
 1 DECIMAL
 2 30 STRING N$ 1 STRING K$ 1 STRING L$ 4 STRING S$
 3 : NAME [ LAST NAME? ] TYPE N$ SINP ;
 4 : FIRST 1 N$ LEFT$ S$ S! ;
 5 : ITH I I N$ MID$ DROP C@ 64 - ;
 6 : KTH DUP " 0123012002245501262301020 2" MID$ K$ S! ;
 7 : BLS S$ K$ S+ S$ S! ;
 8 : TEST K$ L$ S= K$ " 0" S= OR 0= ;
 9 : IST DUP 1 < OVER 26 > OR 0= ;
10 : COMP N$ LEN 1 + 2 DO ITH IST IF KTH TEST IF BLS THEN
11 ELSE DROP THEN K$ L$ S! LOOP ;
12 : SOUND NAME FIRST N$ LEN 2 > IF COMP THEN S$ " 0000"
13 S+ S$ S! CR [ SOUNDEX CODE=    ] TYPE S$ TYPE CR ;
14
15
```

Some Examples of the String Handling Package

```
DECIMAL       ( I/O in Base 10 )
50 STRING ALPHA     ( create a string variable ALPHA with a
                      maximum length of 50 characters )
25 STRING BETA      ( create a string variable BETA )
" NOW IS THE TIME" BETA S!    ( store the literal string
                                NOW IS THE TIME into BETA )
BETA TYPE       ( output the contents of BETA )
NOW IS THE TIME
BETA LEN .      ( current length of BETA is 15 )
15
BETA MLEN .     ( maximum length is 25 )
25
5 10 BETA MID$    ( substring consisting of characters 5 thru 1(
                    of BETA )
TYPE              (  output substring )
IS THE
" AT" 5 6 BETA MID$ SUB!    ( replace the substring IS with AT )
BETA TYPE      ( output new BETA contents )
NOW AT THE TIME
6 BETA LEFT$      ( substring of characters 1 to 6 of BETA )
ALPHA S!     ( store in ALPHA )
ALPHA TYPE       ( output ALPHA )
NOW AT
5 ALPHA RIGHT$     ( substring of characters 5 to 6 of ALPHA )
TYPE        ( output )
AT
BETA ALPHA S+      ( concatenate BETA and ALPHA )
ALPHA S!           ( store in ALPHA )
ALPHA TYPE         ( output new ALPHA )
NOW AT THE TIMENOW AT
ALPHA LEN .        ( current length is 21 )
21
" 12345" BETA S!     ( new data for BETA )
BETA VAL .           ( convert string to number )
12345
32760 STR$           ( convert number into string )
TYPE
32760
```

(continued on page 45)

(continued from page 43)

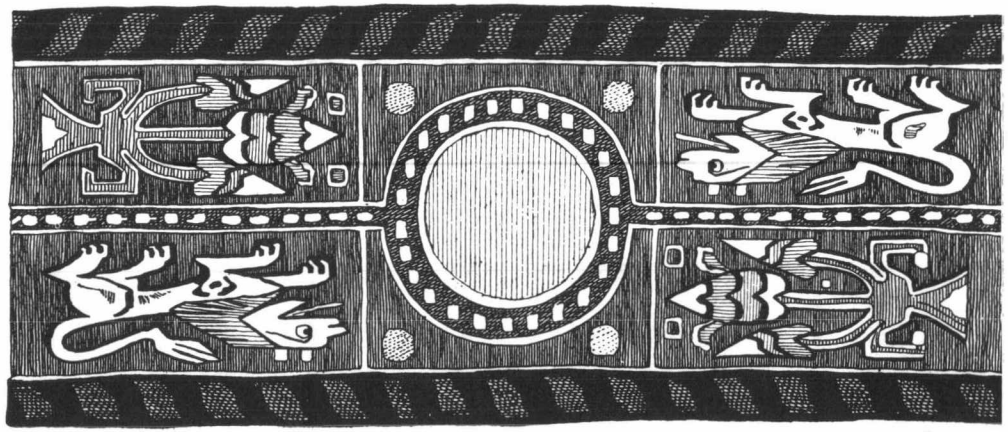
These four screens are the necessary functions to complete the string handling package for 6502 FORTH. A string array data type has also been included, which can be added to any version of FORTH.

```
SCR# 01

 0 ( 6502 FORTH routines for string handling package )
 1 HEX ASSEMBLER
 2 CODE LNK  "LNKUP" JMP ;
 3 CODE  CMOVE  "POPXY" JSR 6 Z STX "POPXY" JSR 4 Z STX
 4 5 Z STY "POPXY" JSR OF Z STX 10 Z STY 0 # LDY OF IY LDA
 5 4 IY STA INY 6 Z CPY F7 BCC "NEXT" JMP ;  DELETE
 6 : C, HERE C= 28 1+= LNK ;
 7 0 INTEGER HLD
 8 : <# HERE 7 + HLD = ;
 9 : ># DROP HLD @ 1 + HERE 8 + OVER - ;
10 : HOLD HLD @ C= HLD 1-= ;
11 : # BASE @ /MOD SWAP 9 OVER < IF 7 + THEN 30 + HOLD ;
12 : #S BEGIN # DUP 0= END ;
13 : SIGN SWAP 0 < IF 2D HOLD THEN ;
14 DECIMAL
15

SCR# 02

 0 ( LITERAL STRING DEFINITION )
 1 HEX ASSEMBLER
 2 LABEL (")     LABEL LIT
 3 CODE " LIT JMP % (") 2 *+ .WORD CLC 0 Z LDA 3 # ADC TAX
 4 1 Z LDA 0 # ADC TAY 'PUSHXY' JSR 2 # LDY 0 IY LDA TAX DEX
 5 0 # LDY 'PUSHXY' JSR INX TXA SEC 0 Z ADC 0 Z STA 2 BCC 1 Z INC
 6 'NEXT' JMP % LIT 0 # LDY OE Z LDA OB BEQ (") #< LDA 28 IY STA
 7 INY (") #> LDA 28 IY STA INY INY 0 # LDX 4 Z STY 'BLKT' JSR
 8 FB BNE 4 Z LDY 9 Z INC 2 BNE OA Z INC 9 IX LDA 22 # CMP
 9 5 BEQ 28 IY STA INY EF BNE 0 # LDA 28 IY STA INY 4 Z STY
10 OE Z LDA 3 BEQ 'BFIN+2' JMP 28 Z LDX 29 Z LDY 'PUSHXY' JSR
11 4 Z LDX DEX 0 # LDY 'PXY' JMP ;
12 IMMEDIATE "
13 DELETE DECIMAL
14 ( STR$ DEFINITION )
15 : STR$ DUP ABS <# 0 HOLD #S SIGN ># 1 - ;

SCR# 03

 0 ( VAL DEFINITION )
 1 HEX ASSEMBLER
 2 CODE DIGIT 'POPXY' JSR TXA 30 # CMP 19 BCC 3A # CMP 4 BCC
 3 41 # CMP 11 BCC 'CNVRT' JSR 'BASE+8' A CMP 9 BCS TAX 0 # LDY
 4 'PUSHXY' JSR 'EQ' JMP 'NEQ' JMP ;    DELETE
 5 : VAL DROP DUP C@ 2D =L IF 1 + -1 ELSE 1 THEN SWAP 0
 6 BEGIN OVER C@ DIGIT IF SWAP BASE @ * + SWAP 1 + SWAP 0
 7 ELSE 1 THEN END SWAP DROP * ;
 8 DECIMAL
 9
10
11
12
13
14
15
```

```
SCR# 04

 0 ( STRING ARRAY DEFINITION )
 1 DECIMAL
 2 : SARRAY <BUILDS ABS 255 MIN 1 MAX SWAP 0 DO DUP DUP C,
 3 0 DO 32 C, LOOP 0 C, LOOP DROP DOES> SWAP 1 - OVER C@
 4 2 + * + 1 + DUP SRCH ;
 5
 6 ( String array requires two stack values. The top value is
 7 the length of each element while the second value is the
 8 number of elements. The indices of the array start at 1.
 9 As an example to create a string array called B$ with 4
10 elements, each with a length of 10 characters execute
11    4 10 SARRAY B$
12 To access the second element of B$ execute
13    2 B$
14 The elements of string array are compatible with string
15 variables.  )
```

All of the functions presented in this article have been implemented in a version of FORTH called 6502 FORTH. For more information on the string handling package and on FORTH for the 6502 contact: Eric Rehnke, 1067 Jadestone Lane, Corona, CA 91720.

end



A 6800 VIDEO DISPLAY PROGRAM

by Kenneth Drexler

If you want to get away from the slow speed, paper costs and noise of a teletype or other hard copy terminal, try a video display. Here is a 6800 video display program for use wih a memory mapped video board such as SSM's VB1 or others using a 1K display memory.

The program has a full range of commands including: Cursor Up, Cursor Down, Cursor Right, Cursor Left, Cursor On and Off, Home, Erase to End of Line, Backspace, Screen Clear and, of course, Carriage Return and Line Feed. This displayed line and page length may be varied up to a maximum size of 64 characters by 16 lines. All data before being displayed by the program are checked against high and low masks to see if the data falls within the limits set by these masks. By changing these masks which are located in RAM, it is possible to limit the characters to a subset of the full ASCII alphabet, such as capital or lower case letters only.

The program occupies 435 bytes of memory, is fully relocatable, and may be placed in ROM. It uses 16 bytes of RAM as temporary storage. The RAM may be located anywhere in memory. The only address constraint is that the 1K video display RAM must be located starting on a 1K boundary ($0000, $0400, $0800, etc.) The program uses 12 of the 32 ASC11 control codes ($0400, $0800, etc.) The program uses 12 of the 32 ASC11 control codes ($00-$1F) for control.

Use of the program is simple. The character to be displayed (or control code) is placed in the A Register and a subroutine branch or jump to the program executed. The program then stores the character in the A Register at the cursor location in the video display RAM (or executes the appropriate control function), computes the address of the next character to be displayed, and, if the cursor is enabled, stores the cursor character at the new location. After setting VNRDY, if necessay, control is returned to the calling program. All registers are preserved.

Sixteen random access memory locations are used to store variables used by the Program. These variables are listed in Figure No. 1. The first eight of these, LLNTH, PGLNTH, MASKLI, MASKHI, PAGE, VNRDY, CURSOR and CURBIT may be changed to alter the display characteristics. Thus the data stored at cursor ($FOA6) is used by the program as the cursor and by changing this data the cursor character may be changed. LLNTH ($FOA6) and PGLNTH ($FOA1) contain hex characters used by the program as the maximum line and page length, respectively. These bytes also control when a carriage return and line feed is generated by the program and when the display page is full.

FIGURE 1: RAM VARIABLES

NAME	ADDRESS (Hex)	FUNCTION
LGNTH	FOAO	Line Length
PGLNTH	FOA1	Page Length
MASKLO	FOA2	Low Mask for Data
MASKHI	FOA3	High Mask for Data
PAGE	FOA4	Page Control, 0=Off, 1=On
CURBIT	FOA5	Cursor Control, O-Off, 1-On
CURSOR	FOA6	ASCII Code for Cursor Character
VNRDY	FOA7	Full Page Flag, 1=Full Page
CURSAV	FOA8	Storage for Data under Cursor
VSAVX	FOA9	Storage for the X Register
CHAR	FOAB	Character psoition (0-63)
LINE	FOAC	Line number (0-15)
CHRPSN	FOAD	Memory address for the next character to be displayed

Each of the first eight RAM locations must be initialized before the first subroutine call of the progam. A routine to perform this function is included in the listing. It sets the RAM locations to the initial values shown in Figure 2. The remaining variables are used by the program and should not be changed.

FIGURE 2: INITIAL RAM VALUES

BYTE	VALUE (Decimal)
LLNTH	32
PGLNTH	15
MASHLO	32 (ASCII Space)
MASKHI	127 (AS11 Rubout)
PAGE	0 (Off)
CURBIT	255 (On)
CURSOR	62 (ASC11 >)

The PAGE feature requires some explanation. The PAGE feature is enabled if PAGE ($FOA4) is set to some value other than zero. When PAGE is enabled, after the location for display of the next character is computed, the program determines if this location is at the end of the page determined by LLNTH and PGLNTH. If it is, VNRDY set to "1" and no further characters are accepted for display. The program con-

Kenneth Drexler – Three Embarcadero Center, Suite 2300, San Francisco, CA 94111

tinues, however, to accept control characters, which allow the display RAM to be cleared or the cursor to be moved. □

LISTING

```
                        *THIS PROGRAM IS RELOC-
                        *ATABLE.  THE VIDEO RAM
                        *MUST BE LOCATED ON A
                        *1K BOUNDARY.

                        *RAM ADDRESSES*

F0A0                    ORG   $F0A0
F0A0                LLNTH   RMB   1
F0A1                PGLNTH  RMB   1
F0A2                MASKLO  RMB   1
F0A3                MASKHI  RMB   1
F0A4                PAGE    RMB   1
F0A5                CURBIT  RMB   1
F0A6                CURSOR  RMB   1
F0A7                VNRDY   RMB   1
F0A8                CURSAV  RMB   1
F0A9                VSAVX   RMB   2
F0AB                CHAR    RMB   1
F0AC                LINE    RMB   1
F0AD                CHRPSN  RMB   2

DC00                VRAM    EQU   $DC00
00DC                VRAMHI  EQU   $DC

                        *VIDEO COMMANDS*

000D                CR      EQU   $D
000A                LF      EQU   $A
000B                CURUP   EQU   $B   CTRL K
0006                CURFW   EQU   $6   CTRL F
000E                CURBK   EQU   $E   CTRL N
0004                CURDN   EQU   $4   CTRL D
0017                CRONOF  EQU   $17  CTRL W
0016                CLEAR   EQU   $16  CTRL V
0015                ETOND   EQU   $15  CTRL U
0008                BS      EQU   $8   CTRL H
0010                HOMUP   EQU   $10  CTRL P
0018                CAN     EQU   $18  CTRL R

0000                    ORG   $0000

                        *ENTRY AND MAIN LOOP*

                        *SAVE REGISTERS
0000 37             VIDEO   PSHB
0001 36                     PSHA
0002 FF F0 A9               STX   VSAVX
0005 FE F0 AD               LDX   CHRPSN
                        *CHECK FOR COMMANDS
0008 81 0A                  CMPA  #LF
000A 27 57                  BEQ   VLF
000C 81 0D                  CMPA  #CR
000E 27 5A                  BEQ   VCR
0010 81 15                  CMPA  #ETOND
0012 27 5D                  BEQ   ERASLN
0014 81 16                  CMPA  #CLEAR
0016 27 5D                  BEQ   CLSCRN
0018 81 0B                  CMPA  #CURUP
001A 27 5B                  BEQ   CURUPP
001C 81 04                  CMPA  #CURDN
001E 27 43                  BEQ   VLF
0020 81 06                  CMPA  #CURFW
0022 27 5A                  BEQ   CURFWD
0024 81 0E                  CMPA  #CURBK
0026 27 5D                  BEQ   CURBAK
0028 81 10                  CMPA  #HOMUP
002A 27 60                  BEQ   HOME
002C 81 08                  CMPA  #BS
002E 27 66                  BEQ   DELPRV
0030 81 18                  CMPA  #CAN
0032 27 6F                  BEQ   CANCEL
0034 81 17                  CMPA  #CRONOF
0036 27 74                  BEQ   CURTOG
                        *CHECK FOR FULL PAGE
0038 7D F0 A7               TST   VNRDY
003B 26 20                  BNE   VEXIT
                        *CHECK DATA V  MASKS
003D B1 F0 A2               CMPA  MASKLO
0040 2D 1B                  BLT   VEXIT
0042 B1 F0 A3               CMPA  MASKHI
0045 22 16                  BHI   VEXIT
                        *STORE DATA, COMPUTE
                        *NEW POSITION
0047 A7 00                  STAA  0,X
0049 7C F0 AB               INC   CHAR
004C 8D 65          VUP     BSR   GPSN1
                        *SAVE DATA AT NEW POS-
                        *ITION, INSERT CURSOR
004E A6 00                  LDAA  0,X
0050 B7 F0 A8               STAA  CURSAV
0053 7D F0 A5               TST   CURBIT
0056 27 05                  BEQ   VEXIT
0058 B6 F0 A6               LDAA  CURSOR
005B A7 00                  STAA  0,X
                        *RESTORE REGS AND EXIT
005D 32             VEXIT   PULA
005E 33                     PULB
005F FE F0 A9               LDX   VSAVX
0062 39                     RTS

                        *VIDEO SUBROUTINES*

0063 8D 50          VLF     BSR   RPLCUR
0065 7C F0 AC               INC   LINE
0068 20 E2                  BRA   VUP
006A 8D 49          VCR     BSR   RPLCUR
006C 7F F0 AB               CLR   CHAR
006F 20 DB                  BRA   VUP
```

(continued)

```
0071 8D 48      ERASLN BSR  ETEND
0073 20 D7             BRA  VUP
0075 20 52      CLSCRN BRA  ESCRN
0077 8D 3C      CURUPP BSR  RPLCUR
0079 7A F0 AC          DEC  LINE
007C 20 CE             BRA  VUP
007E 8D 35      CURFWD BSR  RPLCUR
0080 7C F0 AB          INC  CHAR
0083 20 C7             BRA  VUP
0085 8D 2E      CURBAK BSR  RPLCUR
0087 7A F0 AB          DEC  CHAR
008A 20 C0             BRA  VUP
008C 8D 27      HOME   BSR  RPLCUR
008E 7F F0 AB          CLR  CHAR
0091 7F F0 AC          CLR  LINE
0094 20 B6             BRA  VUP
0096 8D 1D      DELPRV BSR  RPLCUR
0098 7A F0 AB          DEC  CHAR
009B 8D 16             BSR  GPSN1
009D 86 20             LDAA #$20
009F A7 00             STAA 0,X
00A1 20 AB             BRA  VUP+2
00A3 7F F0 AB  CANCEL  CLR  CHAR
00A6 8D 0B             BSR  GPSN1
00A8 8D 11             BSR  ETEND
00AA 20 A0      VUP1   BRA  VUP
00AC 8D 07      CURTOG BSR  RPLCUR
00AE 73 F0 A5          COM  CURBIT
00B1 20 9B             BRA  VUP+2
00B3 20 4A      GPSN1  BRA  GPSN
00B5 B6 F0 A8  RPLCUR  LDAA CURSAV
00B8 A7 00             STAA 0,X
00BA 39                RTS
00BB 86 40      ETEND  LDAA #$40
00BD B0 F0 AB          SUBA CHAR
00C0 C6 20             LDAB #$20
00C2 E7 00      ETLOP  STAB 0,X
00C4 08                INX
00C5 4A                DECA
00C6 26 FA             BNE  ETLOP
00C8 39                RTS
00C9 CE DC 00  ESCRN   LDX  #VRAM
00CC 86 20             LDAA #'
00CE A7 00      ESLOP  STAA 0,X
00D0 08                INX
00D1 8C E0 00          CPX  #VRAM+1024
00D4 26 F8             BNE  ESLOP
00D6 7F F0 AB          CLR  CHAR
00D9 7F F0 AC          CLR  LINE
00DC 20 CC             BRA  VUP1
00DE CE DC 00  SCROLL  LDX  #VRAM
00E1 36                PSHA
00E2 37                PSHB
00E3 A6 40      SCLOP  LDAA $40,X
00E5 A7 00             STAA 0,X
00E7 08                INX
00E8 8C DF C0          CPX  #VRAM+1024-6
```

4

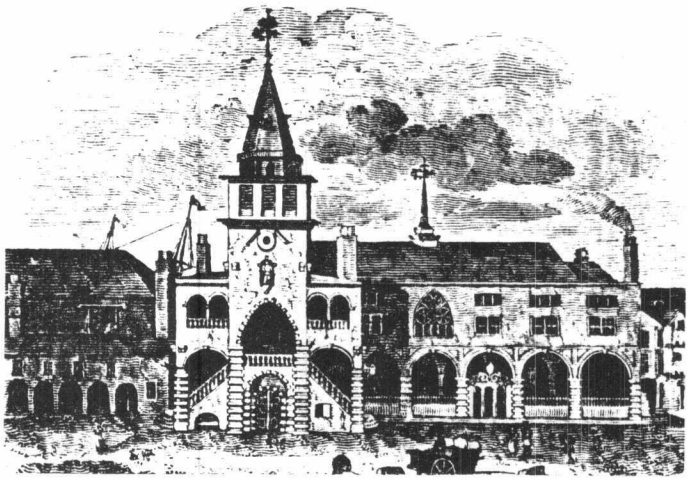
```
00EB 26 F6             BNE  SCLOP
00ED B6 F0 A1          LDAA PGLNTH
00F0 B7 F0 AC          STAA LINE
00F3 7F F0 AB          CLR  CHAR
00F6 8D 4B             BSR  VPSN
00F8 8D C1             BSR  ETEND
00FA 33                PULB
00FB 32                PULA
00FC 39                RTS
00FD 20 CA     ESCRN2  BRA  ESCRN
               *GET NEW POSITION,
               *ADJUST CHAR AND LINE
00FF 37        GPSN    PSHB
0100 36                PSHA
0101 F6 F0 AB          LDAB CHAR
0104 2A 0B             BPL  NEXT1
0106 F6 F0 A0          LDAB LLNTH
0109 F7 F0 AB          STAB CHAR
010C 7A F0 AC          DEC  LINE
010F 20 0B             BRA  VCOK
0111 F1 F0 A0  NEXT1   CMPB LLNTH
0114 25 06             BCS  VCOK
0116 7F F0 AB          CLR  CHAR
0119 7C F0 AC          INC  LINE
011C F6 F0 AC  VCOK    LDAB LINE
011F 2A 04             BPL  NEXT2
0121 7F F0 AC          CLR  LINE
0124 5F                CLRB
0125 F0 F0 A1  NEXT2   SUBB PGLNTH
0128 2F 0F             BLE  VLOK
012A 7D F0 A4          TST  PAGE
012D 27 05             BEQ  VSCRL
012F 7C F0 A7          INC  VNRDY
0132 20 08             BRA  PSNXT
0134 8D A8     VSCRL   BSR  SCROLL
0136 5A                DECB
0137 26 FB             BNE  VSCRL
0139 7F F0 A7  VLOK    CLR  VNRDY
013C 8D 05     PSNXT   BSR  VPSN
013E 32                PULA
013F 33                PULB
0140 39                RTS
0141 20 BA     ESCRN1  BRA  ESCRN2
               *COMPUTE CHRPSN=
               *64*LINE+CHAR+VRAM
0143 7F F0 AD  VPSN    CLR  CHRPSN
0146 7F F0 AE          CLR  CHRPSN+1
0149 B6 F0 AC          LDAA LINE
014C C6 06             LDAB #6
014E 48        VGLOP   ASLA
014F 79 F0 AD          ROL  CHRPSN
0152 5A                DECB
0153 26 F9             BNE  VGLOP
0155 BB F0 AB          ADDA CHAR
0158 24 03             BCC  VPSN1
015A 7C F0 AD          INC  CHRPSN
```

```
015D B7 F0 AE  VPSN1  STAA CHRPSN+1
0160 86 DC            LDAA #VRAMHI
0162 BB F0 AD         ADDA CHRPSN
0165 B7 F0 AD         STAA CHRPSN
0168 FE F0 AD         LDX  CHRPSN
016B 39               RTS
016C 01               NOP
016D 01               NOP
016E 01               NOP
016F 01               NOP

                *INITIALIZE VIDEO RAM*
                *CLEAR F0A0-F0AF
0170 CE F0 A0  VDINZ  LDX  #LLNTH
0173 86 10            LDAA #$10
0175 6F 00     VNZLOP CLR  0,X
0177 08               INX
0178 4A               DECA
0179 26 FA            BNE  VNZLOP
                *SET: LLNTH=$20, PGLNTH
                *=$0F, MASKLO=$20
                *MASKHI=$7F, CURSOR=>,
                *CURSOR=ON
017B CE F0 A0         LDX  #LLNTH
017E 86 20            LDAA #$20
0180 A7 00            STAA 0,X
0182 A7 02            STAA 2,X
0184 86 0F            LDAA #$F
0186 A7 01            STAA 1,X
0188 86 7F            LDAA #$7F
018A A7 03            STAA 3,X
018C 86 FF            LDAA #$FF
018E A7 05            STAA 5,X
0190 86 3E            LDAA #'>
0192 A7 06            STAA $6,X
0194 20 AB            BRA  ESCRN1

                      END
      NO ERROR(S) DETECTED
```

SYMBOL TABLE

```
BS      0008      CAN     0018
CANCEL  00A3      CHAR    F0AB
CHRPSN  F0AD      CLEAR   0016
CLSCRN  0075      CR      000D
CRONOF  0017      CURBAK  0085
CURBIT  F0A5      CURBK   000E
CURDN   0004      CURFW   0006
CURFWD  007E      CURSAV  F0A8
CURSOR  F0A6      CURTOG  00AC
CURUP   000B      CURUPP  0077
DELPRV  0096      ERASLN  0071
ESCRN   00C9      ESCRN1  0141
ESCRN2  00FD      ESLOP   00CE
ETEND   00BB      ETLOP   00C2
ETOND   0015      GPSN    00FF
GPSN1   00B3      HOME    008C
HOMUP   0010      LF      000A
LINE    F0AC      LLNTH   F0A0
MASKHI  F0A3      MASKLO  F0A2
NEXT1   0111      NEXT2   0125
PAGE    F0A4      PGLNTH  F0A1
PSNXT   013C      RPLCUR  0085
SCLOP   00E3      SCROLL  00DE
VCOK    011C      VCR     006A
VDINZ   0170      VEXIT   0050
VGLOP   014E      VIDEO   0000
VLF     0063      VLOK    0139
VNRDY   F0A7      VNZLOP  0175
VPSN    0143      VPSN1   015D
VRAM    DC00      VRAMHI  00DC
VSAVX   F0A9      VSCRL   0134
VUP     004C      VUP1    00AA
```

END



· Letters ·

. . . AND MORE ON TOP-DOWN

Dear *DDJ*:

In *DDJ* # 48, John Culleton, Jr. defended his statement that PASCAL does not support top-down programming in the physical construction of the program. I would like to say as an educator and programmer that I agree fully with his position that a language that does not have regard for the physical layout of the program can hardly be called a top-down structured language. In the final analysis, the physical layout of the program is what people will read and study, like a book, when trying to understand the program. Without logical placement of the program code, just reading the source listing can be next to impossible, especially on a long program. In this respect, a good BASIC compiler might be more useful than PASCAL, if the language is used properly in a modular way.

I am seriously reconsidering my jump on the PASCAL bandwagon, for a return to a language that can produce readable, understandable listings, that can be read in magazine articles without a computer science degree. The introduction of CP/M and the BASIC compiler from Microsoft for the Apple, with it's chain and global commands just might offer a more useful programming environment than PASCAL when it comes to creating understandable and readable programs.

My experience so far indicates that people are not going to take to PASCAL without a lot of previous programming experience, as the Apple version has already proven that using PASCAL requires a lot of big system savvy on the part of the user, and simple implementation of previous BASIC functions like PEEK and POKE require tremendous machine sophistication on the part of the user in creating library units or linking small assembly subroutines. Contrary to popular belief, PASCAL is definitely not for the beginning programmer. And without clear, logically organized program layouts, I am not sure it's right for the professional production programmer either, as one thing is constant in industry, and that is that programmers come and go like street cars, and it's a sure bet the programmer who starts that big PASCAL project, won't be the one who finishes it.

Regards,
David E. Smith

P.O. Box 701
Placentia, CA 92670

Dear *DDJ*:

In *DDJ* #48 John R. Culleton, Jr. gave a detailed argument as to why he believes top-down programming is impossible in PASCAL. His entire argument hinges on one sentence in the middle of the sixth paragraph:

> "Note that the physical sequence of the modules is also the order in which they are written."

I can find no reason for this statement. A programmer is free to code his modules in any order he pleases, and then put them into any physical sequence he pleases.

I use PASCAL on my home computer, and I use top-down programming. Usually I code the whole program on paper first. I start, of course, with the main line module, then the modules subordinate to it, and so on, ending with any little utility modules that may be required. Then I key them in, usually in the physical order required by PASCAL.

However, if I choose to do my coding at the terminal, I can enter the main line routine first, then insert the subordinate routines before it, and so on. Any text editor will let you do that.

It is true that the physical sequence required by PASCAL makes it a little awkward to read a program, but I believe that it only takes a little getting used to. I agree that this is a disadvantage of PASCAL, but the ability to compile in one pass simplifies the compiler and undoubtedly allows it to run in less memory space. On a small computer, this is a tradeoff well worth making.

Very truly yours,
Jim Scott

1724 Blakemore Rd.
Richmond, VA 23225

Dear *DDJ*:

It would be a pity if potential users of PASCAL were put off by John Culleton's remarks in his letter (*DDJ* #48), which seem to be based on a misunderstanding. Top-down programming in PASCAL is simple and straightforward.

First, standard PASCAL has a FORWARD directive, which allows a procedure to be called before its body is presented to the compiler. (See the Draft Standard, Secs 6.6.1,6.6.2 in the May 1980 issue of *Pascal News*.) This facility is necessary, since without it one could not compile co-routines, which call each other.

Second, and more practically, any reasonably large program to be compiled on a micro or mini is going to have to make use of the EXTERNAL directive, which calls separately compiled procedures stored in a library. This is not standard PASCAL, but every PASCAL system I have seen provides for it. You want your utility programs in a library anyway, so that you can call them whenever you like without having to retype them at program entry via keyboard, disc, or whatever.

In my opinion, PASCAL is the ideal language for the user of small computers. (Large ones, too, for that matter, but *DDJ* is the small user's guide and vademecum, and we are looking out for them in the letter column.) It demands discipline; if properly used it makes for readable and maintainable programs; and you can do anything with it that you ought to want to do anyway. As compared to the pitchfork, spade and overalls approach of the average FORTRAN/BASIC user it will cut debugging time at least in half. Let's don't badmouth it by mistake.

Sincerely,
Arthur A. Brown

AVB Associates, Inc.
Box 326
Garrett Park, MD 20766

MISCELLANEA

Dear *DDJ*:

Your readers might like to know that the American Substandards Committee X.1.5 has recently partially approved a new phonetic alphabet for voice communications. Here it is:

LETTER	CODEWORD
A	AISLE
B	BDELIUM
C	CZAR
D	DJAKARTA
E	EUPHORIA
G	GNOME
H	HOUR
I	IAN
J	JOSE
K	KNIGHT
L	LLAMA

M	MNEMONIC
O	OEDIPAL
P	PHONETIC
Q	QUEUE
R	RWANDA
T	TSAR
U	URN
V	VELDT
W	WRONG
X	XANADU
Y	YVONNE
Z	ZWIEBACK

A typical example of the alphabet in use is this recent exchange:

RECORDING: You really can help keep your phone costs down, just by looking up numbers in your phone book, whenever possible. Thanks for helping.

MA BELL: City and number, please?

CALLER: Taco Bell in San Francisco.

MA BELL: Was that "Paco Bell?"

CALLER: No, "TACO." "T" as in tsar, "A" as in aisle, "C" as in czar, "O" as in Oedipal.

MA BELL: Was that "C" as in czar or "T" as in tsar?

CALLER: Yes.

MA BELL: Oh.

Mike Gabrielson
Bob Phillips
Ron Adler
Tom Martin
Jim Munro
Glenda Core

M. Gabrielson
P.O. Box 2692
Stanford, CA 94305

NON-HANDSHAKING SOLUTIONS

Dear *DDJ*:

In the August issue of *Dr. Dobb's Journal*, you published my letter concerning the non-handshaking situation between my Heath H-14 printer and my North Star Horizon serial interface. Thanks to your audience, I was deluged with letters as well as phone calls. Software solutions outnumbered hardware solutions about two-to-one.

One hardware solution I received was to interchange pins 4 and 7 at the Heath connector to reverse the RTS and NOT-RTS signals. The gentleman claimed his H-14 was connected to serial port 2 of a North Star Horizon and stated that his left configuration header, package 3D, was wired the same as both my configuration headers are. Since the right configuration header, package 4D, controls the second serial port, his solution may have worked for him, but not for me!

The simplest hardware solution was provided by Al Weiner of Waltham, Massachusetts. Without rewiring the Horizon header at 4D, all I had to do was change the Heath end of the cable to move DB-25 connector pin 20 to Heath connector pin 4 to tap the NOT-RTS signal provided in the Heathkit modification on DB25 connector pin 15. Al even provided a three-line BASIC program to prove that the hand-shaking was working at 4800 baud!

At first, the solution didn't work. I finally hooked up my TVOM and discovered I didn't have the NOT-RTS signal at the printer (although I had followed the modification instructions during assembly). I had never opened the printer since its first visit to Heathkit Electronics Center in Seattle and assumed that they had checked it out thoroughly.

They certainly did! Even though I advised them of the modification number, they ripped out the additional wiring underneath the PC board! I've rewired the mod, and my H-14 is running along at 4800 baud with full handshaking under CP/M as well as N*DOS. The fix would have been implemented two weeks ago instead of this morning had HEC told me of their action months ago.

Thanks to everybody who replied to my previous letter to the editor—it's nice to know I'm not alone in the wilderness without help! And thank you, *Dr. Dobb's Journal*, for publishing my complaint which triggered all the response.

Cordially,
John R. Dye, CPD

4807 15th Ave. S.E.
Lacey, WA 98503

Editor's note: Herewith an example of the response to J. Dye:

Dear *DDJ*:

This letter is being printed on an H-14 connected to serial port 2 of a North Star Horizon using stock I/O drivers for North Star/UCSD PASCAL; communication rate is set at 4800 baud. I think that we can conclude that the incompatibility discussed in the referenced letter can be readily resolved.

First of all, the North Star software does not respond to the software interrupt handshaking provided by the DEC-style XON/XOFF signalling. It does not need to since the handshaking can be accomplished using RTS and CTS control lines which are incorporated in RS-232C. These lines are properly handled by the North Star software.

There are several generations of H-14 printers each having a different handshaking arrangement. To determine which one you have, read the description of the interface on H-14 OPERATION manual page 9, being sure to incorporate any change notices into the manual first. Some of the earlier printers have a printer busy signal on pin 20 of the DB-25 connector which goes positive when the printer can accept data. The current printer has a similar signal on pin 15 of the DB-25 connector. In these later printers there is also an inverted busy signal on pin 4 of the DB-25 connector. Unfortunately, none of these connections is compatible with the configuration header wiring scheme on page 72 of North Star's HRZ-D-DOC. What is needed is to present a signal on pin 4 of the DB-25 connector that will go positive whenever the printer can accept characters.

Page 8 of the H-14 OPERATION manual shows that by exchanging pins at the end of the RS-232C cable opposite the DB-25 connector, we can provide the correct signal on pin 4. To do this on the later H-14, all one needs to do is use a sharp tool to extract pins 4 and 7 of the connector, exchange them and reinsert them. On the earlier H-14, pins 14 and 7 need to be exchanged.

To complete the interface, the left configuration header, package 3D, on the North Star motherboard must be strapped as follows (reference page 72 of HRZ-D-DOC):

pin 1 to pin 12
pin 2 to pin 16
pin 3 to pin 15
pin 4 to pin 14
pin 5 to pin 11
pin 7 to pin 8
pin 9 to pin 10

(continued)

This interface will allow the USART, package 3A, and the software which controls it to interface the H-14 at any baud rate up to and including 4800.

Sincerely,
Robert Rennard

2281 Cobble Stone Court
Dayton, OH 45431

CP/M COMPATIBLE SOFTWARE MARKETING

Dear *DDJ*:

As the developers of CP/M and MP/M, Digital Research is preparing a list of vendors of CP/M-compatible software. We would appreciate the help of readers of your magazine in compiling this list for distribution to all interested persons who contact us.

If any readers are currently marketing CP/M-compatible software, please send us any or all literature pertaining to your software which controls it to interface the H-14 at any baud rate up to and including 4800.

Thank you,
Marilyn Darling

Digital Research
P.O. Box 579
Pacific Grove, CA 93950

AT ODDS WITH *DDJ*

Dear *DDJ*:

I have received my copies of Volume 5, Issue 8 of *DDJ*, which contains my article (*A Note On 6502 Indirect Addressing*), and I am slightly upset.

I have about 70 papers to my credit, in all kinds of journals, and this is the *very first time* that I have not had the courtesy of having galley proofs to review before an article of mine is printed.

I understand about your policy of quick publication; I want to see my ideas on programming published as quickly as possible, too—but not with typists' mistakes still in them! Typists are not infallible, nor should they be expected to be. For the record, the mistakes were as follows:

P. 26 col. 1 line 37: "is keep" should be "is to keep"

P. 26 col. 3 line 34: left paren should line up under I

P. 27 col. 1 line 57: "each by" should be "each time by"

P. 27 col. 3 line 11: [1] refers to *reference number 1*, which is not marked as such. (I don't mind if you want to set up your own conventional notation for references to journal papers and books, but it has to be consistent with itself.)

None of these turned out to be really serious, in this case—which is why I am slightly, rather than considerably, upset.

Sincerely,
Professor W. D. Maurer

George Washington University
S.E.A.S.
Washington, DC 20052

Prof. Maurer's objection is well taken and the subject of this month's editorial (see p. 1). —*JP*

SIGNED COMPARISON: ALTERNATE ROUTINE

Dear *DDJ*:

While scanning the listing of the runtime library for the Small C compiler in issue #48, I happened to notice the signed compare routine ("cccmp"). This routine performs a comparison of two 16-bit signed numbers (two's complement) by subtracting one of the numbers from the other and then examining the sign of the result. This technique works fine as long as the difference between the numbers being compared is less than or equal to 32767; however, if the numbers differ by more than 32767, the result of the compare will be incorrect due to arithmetic overflow during the calculation.

A different way to accomplish a signed comparison is to invert the sign bits of the two numbers being compared and then to perform an unsigned compare using the new numbers. A routine using this technique is shown below:

```
; Common routine to perform a signed compare of DE & HL.
; This routine compares DE & HL and sets the conditions:
;    Carry reflects sign of difference (set if DE < HL)
;    Zero/non-zero set according to equality.
;
cccmp:  MOV   A,H      ; invert sign of HL.
        XRI   80H
        MOV   H,A

        MOV   A,D      ; invert sign of DE.
        XRI   80H

        CMP   H        ; compare msbs.
        JNZ   cccmpl   ; unequal—comparison done.

        MOV   A,E      ; equal—compare lsbs.
        CMP   L

cccmpl: LXI   H,1      ; (required by Small C).
        RET
```

This new method works by mapping the signed number domain into the unsigned number domain, where 16-bit comparisons may be easily made, before making the comparison. A few examples of number mappings are shown below:

Decimal number	Hex number before map	Hex number after map
-32768	8000	0000
-32767	8001	0001
.		
-1	FFFF	7FFF
0	0000	8000
1	0001	8001
.		
32767	7FFF	FFFF

Now, does this solve a problem or just create a new problem? Consider the two expressions shown below, which appear to be equivalent:

Expression	New result	Old result	"Correct" result
(-4000 < 30000)	True	False	True
(-4000 - 30000 < 0)	False	False	False

Because of arithmetic overflow in the second expression, where "-4000 - 30000" is correctly interpreted as 31536 (not -34000), the expressions have different values. The new "improved" compare yields the differing results, depending upon how the expression is formed; whereas the old compare yields identical (but incorrect in one case) results no matter which way the expression is formed. Which attibute appeals to you more: correctness or apparent consistency? I vote for correctness in this case.

Yours truly,
Harry B. Stewart

Neoteric
15816 San Benito Way
Los Gatos, CA 95030

MORE ON LINKING WITH ERROR CORRECTION

Dear *DDJ*:

Many thanks for publishing Sylvan Rubin's note on linking with error correction (*DDJ* #48). It is a fine example of a concise, lucid exposition of an improved technique applicable to a large class of programs and quite unrelated to any particular processor.

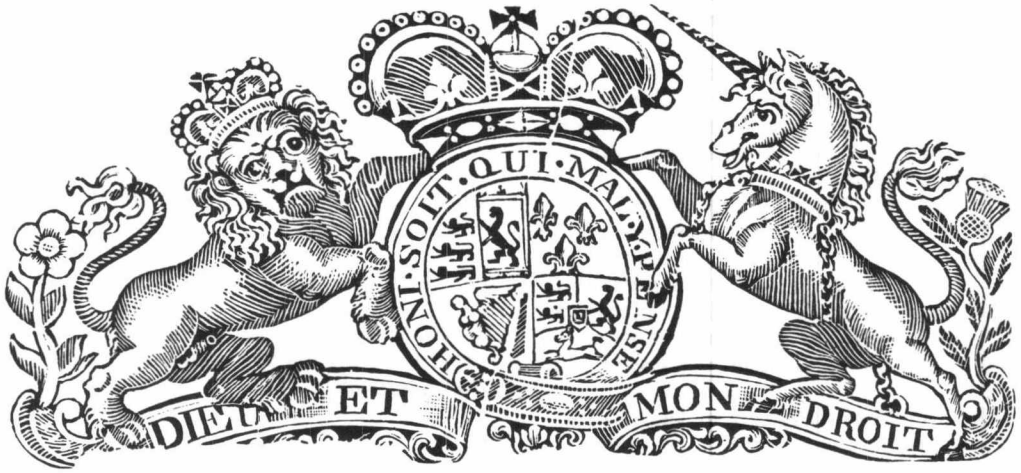
I should like to bring to your attention what I consider a small improvement to Mr. Rubin's plan. Let the primary link field contain what Mr. Rubin calls the cross link: (predecessor address) XOR (successor address). By the method given in the italicized note this one field affects both forward and backward linking, if one knows where one has been. Then let the secondary link field contain the one's complement of the primary link. The one's complement operation is chosen because of its sensitivity to stuck (stuck at one, stuck at zero) faults, which are among the most common machine faults. In the absense of error the XOR of the two links will be all ones.

Of course this scheme too has its limitations. While each zero in the XOR of the two links implies that one of them is wrong, each one may mean that they are both wrong. In that case, or when successive nodes have link errors one learns the value of backup copies and checkpointing.

Incidentally, I cannot recommend the practice of "correcting" any portion of a dataset without notice to the user. A conscientious user needs to know about errors. That need is one of the reasons for writing exception reports. Errors rarely come as single spies; reporting those detected often helps detect others. If reporting errors makes the report too long then there are far too many errrors.

Sincerely,
J. A. MacLaughlin

Folly Fraction Farm
Clarksville, MD 21029



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 5 — 1980
January	41
February	42
March	43
April	44
May	45
June/July	46
August	47
September	48
October	49
November/December	50

1802
  Cross-assembler [CP/M] 432
  Elf 122
6800
  Cross-assembler [CP/M] 432
  Video-display program 439
8080
  And CP/M 396
  And Z80 289; 394
  Fast fourier transform 164
  PILOT 82
  Registers in 47 bytes 173
  Small C compiler 176
  Translator system [to Z80] 216
  Unaltered 42
  Variations 173
  /Z80 indirect addressing 289; 394
8813
  Polymorphic math functions 125
  9 or 16 bits? [Opcode design] 373
9900 46; 108

Acoustic coupler 351
Addendum
  Newberry, S. 395
Addition trick 125
Addressing, indirect
  6502 333
  8080/Z80 289; 394
Adler, R. 443
ADM-3A 56
Algorithms 130; 169; 207; 264; 350
Allison, D. 73; 169; 207; 264; 350
Alpha listing [H8] 104
ALS-8 to CP/M conversion 24; 261
Animal 350
APPLE
  User interface 267
APPLE II
  Program renumbering 67
  ROM byte-finder 243
Arrays
  FORTH 242
  North Star 40
Assemblers
  6800 + 1802 cross-assembler [CP/M] 432
  Table-driven 59; 171
  Z80, subroutines 159

Bach, S. 243; 267
Balcom, O. 39
Barker, L. 42; 173; 205; 222; 268; 309
BASIC 42
  Memory dump 260
  North Star
    Cross-reference 147; 151; 339; 382
    Patches 40; 171
    Statement generator 418
    String arrays 40
    Two-column display 259
  Poly 88 141
  Structured 201
  Tiny BASIC 201; 356
Bender, A. 209
Bentley, J. 130
Bertollo, D. 348
Binary to decimal conversion 73
Bokelman, J. 42
Borochoff, R. 126
Bronstein, M. 122
Brown, A. 443
Brown, D. 67
Bugs
  Small-C 83
  Sam 76's CP/M 83
Byte
  ROM byte-finder [APPLE II] 248

C 15; 176; 191
C, small *see* Small C
C, tiny *see* Tiny C
Caesar, A. 309
Cain, R. 176; 312
Calculators 392
Carter, J. 159; 267
Cason, R. 127
Cassette interface 13
Caulkins, D. 230; 284
CDOS 29; 394
Colley, W. 432
Compilers
  META II 169
  Small C 176; 312
Conley, W. 352
Conversions/converters
  ALS-8 to CP/M 24
  Binary to Decimal 73
  CP/M to UCSD PASCAL 367
Core, G. 443
Corrections
  Barker, L. 309
  Keay's table 41
  Parsons, R. 83
  Silverston, S. 172
COSMAC 282
CP/M II
  6800 + 1802 cross-assembler 432
  AL8-8 to CP/M file converter 24; 261
  And CDOS 29; 394
  Bug [SAM 76's CP/M] 83
  Game 406
  Hardcopy device driver programs 341
  Linking to Morrow's cassette interface 13
  OSI's CP/M 205
  PEN 34
  Software 4; 32
  Table-driven assembler 59
  Tiny interpreter [CP/M flavored C] 15
  To UCSD PASCAL file conversion 367
  Z80, 8080, and CP/M 396
Crockett, T. 79
Cross-assemblers 432
Culleton, J. 125; 352

Da Costa, F. 39
Darling, M. 445
Dea, R. 216
Deane, R. 242; 434
Design
  Algorithm 130
  Opcode 373
Diablo 268
Digital systems DDOS 127
Displays
  6800 439
  North Star 259
Documentation 426
Double density system 339
Douds, C. 127
Drexler, K. 439
Dump/Dumping
  BASIC 260
Dye, J. 309; 444

Ecklein, D. 394
Elf 268
Elizondo, F. 261
Enhancements
  ADM-3A 56
  Entry points 267
  Equations, differential 392
Errata of computer programming 244
Errors 309; 349; 446

Fiddes, L. 394
File conversion *see* Conversion
File transfer 375
Flashwriter I 406
Foreign language trainer 423
Formatter, TI 9900 46; 108
FORTH 242; 434
Frampton, H. 24
Fusfeld, R. 168

Gabrielson, M. 443
Games, CP/M 406
Gammell, J. 40
Garcia, L. 418
Gaugler, G. 345; 423
Gelles, A. 85
Gibson, T. 201
Glitch 222
Go: utility command under CP/M 11
Gordon, H. 324; 373; 395
Gramer, J. 125
Graphics
  Flashwriter I [CP/M] 406
  Microprocessor-assisted system 79
Guthery, S. 201

Hallen, R. 34
Hamlin, G. 79
Hardcopy device driver programs [CP/M] 341
Head, G. 222
Heath
  H-8 104; 287
  H-14 201; 332; 395
Hemminger, C. 309
Hex keyboard 122
Holtrichter, J. 339
Horizon [Interfacing H-14] 395
Howell, J. 173; 266; 396
Hudson, G. 259
Hudson, L. 40; 382
Hudson, R. 356
Hughes, L. 375

Improvements to:
  APPLE user interface 267
  CP/M and CDOS 394
  Zant's modular programming 266
Indirect addressing [8080 & Z80] 289; 333
Installation, Phantom-deselect 42
Interface
  APPLE 67
  CP/M linked to Morrow's cassette interface 258
  H-14 to Horizon 395
  WH 14 printer to serial POR 443
Interpreter
  LISP 268
  Tiny 15

Jamieson, C. 83
Johnson, D. 84
Johnson, R. 171
Johnson, S. 197

Keay, C. 41
Kernigham, B. 191
Keyboard, Hex 122
Keyplus integrated utility 309
Kildall, G. 2
Knippenberg, H. 260
Knuth, D. 88; 244
Konopik, M. 234; 332
Kruse, R. 272; 395

Languages
  C 191
  Foreign 423
  META II 142
  Syntax-oriented compiler writing language 142
Las Vegas super-slot 406
Lesk, M. 191
Lindenschmitt, G. 84
LISP, interpreter 268
Listing, H-8 104
Logic 348
Long, W. 371
Loughman, W. 431
Love program 345
Lucky, R. 171; 309
Lunsford, D. 172

MacLaughlin, J. 446
Martin, T. 443
Math functions [8813] 125
Mathematical typography 88
Maurer, W. 333; 445
MCALL-C 395
McGaw, J. 125
Memory dump [BASIC] 260
Microchess 126
Microsoft 222
Miller, W. 13
MITS DOS disk to disk copy routine 209
Modular programming 266
Monitors
  Elf tiny 268
  SYM-1 324
Moore, F. 426
Morrow, G. 13
Moss, W. 104
MOVIT 431
Multiple file transfer 375
Munro, J. 443

N-LOGS 400
Naess, O. 364
Networks 226; 230; 257
Newberry, S. 42; 125
NIBASM 371
North Star *see* BASIC; Horizon
Number language [N-LOGS] 400

Oliver, J. 39
Opcode design 373
OSI 205
Output [COSMAC 1861] 282

PASCAL 75; 127; 367
Patches, North Star DOS 40
PCNET 230; 284
PEN 34
Personal Computer Network *see* PCNET
PET 127
Pfafman, F. 42
Phelan, C. 309
Phillips, T. 443
Phone dialer 84
PILOT 82
Poly 88 BASIC 141
Polymorphic 8813 math functions 125
Price, G. H. 282
Printer [H-14] 332
Probes, logic 348
Programs
  6800 439
  CP/M 406
  CP/M + CDOS compatability 29
  Love program 345
  User interface to APPLE-II 267
Programmable calculator 392
Programming
  BASIC 341
  C and tiny C 201
  C language 191
  CP/M 341
  Changing SPI to CP/M 32
  Errata 244
  Pastimes and pleasures 37; 285; 330
  Zant's modular programming 266
Protocol 234
Pugh, T. 234
Purdin, T. 151

Quick-key 418
Quinn, C. 39

Raeside, D. 121
Reentrant formatter [TI9900] 108
Register/storage [8080] 173
Rennard, R. 444
Ritchie, D. 191
ROM byte-finder [APPLE II] 243
Routines
  Alternate 445
  MITS DOS disk to disk copy 209
Rowland, D. 269
Rubin, S. 349
Rules 309
Runtime library [small C] 312

S. D. board 257
SAM 76 83
Saunders, W. 395
Schorre, D. 142
Scientific computers 400
Scott, J. 443
Serial port 425
Silverston, S. 172
Simons, K. 400
Sippy, L. 425
Sjowall, T. 367
Skjellum, A. 11; 289
Small C
  8080 176
  Compiler 312
  Runtime library 312
Small systems and monitors 324
Smith, B. 267
Smith, D. 443
Smith, W. 240

Software
CP/M 4; 445
Software market 309
Source-text area, extended 371
SPI formatted programs 32
Sprenkle, J. 279
SR-52 programmable calculator 392
Statement generator [North Star] 418
Stewart, H. 445
STOIC [North Star disk] 55
Stowe, T. 287
String arrays [North Star] 40
Subroutines [Z-80 assembler] 159
SYM-1 324
Symbol table alpha listing [H8] 104
Syntax-oriented compiler writing language 142

Table driven assembler 59; 171
Terminal parameters [PASCAL] 75
Thomson, B. 223
TI 9900 45; 108
Tiny BASIC *see* BASIC
Tiny interpreter 15
Tiny monitor [Elf] 268
TI PMX [TI 9900] 46; 108
Top-down 352; 443
Translator [8080 to Z80] 216
TRS-80 441
Typography, mathematical 88

UCSD PASCAL *see* PASCAL
Utility
Command under CP/M 11
Multiple file transfer 375

Van Buer, D. 59; 171; 268
Video display RAM 439
Vocabulary trainer 423

Wakerley, J. 56
Walters, N. 46; 108
Warner, J. 29
Wasserman, P. 172
Wetherell, C. 37; 77; 285; 330
Wexler, S. 309
WH 14 printer 425
Wilcox, R. 82
Willoughby, S. 341

X-Ref tool [North Star] 339

Yakush, S. 392

Z-80
Assembler 159
Circuit [H-8] 287
Indirect addressing 289; 394
Power increase 172
Translator system from 8080 216
Zant, R. 266
Zeiger, M. 32
ZX65 272; 395

5492-0
$20.75

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

ISBN 0-8104-5492-0